# Hierarchical Context Management: Feasibility Analysis, Architectural Paradigms, and Implementation Strategies for Mitigating Semantic Drift in Large Language Model Systems

## 1. The Context Drift Crisis: Mechanisms, Costs, and the Imperative for Hierarchy

The effective deployment of Large Language Models (LLMs) in complex, real-world production environments is currently facing a critical inflection point. While the theoretical capability of these models to process information has expanded dramatically—with context windows growing from 4,096 tokens to over 1 million tokens—the practical *reasoning* capability over these extended contexts has not scaled linearly. As interaction histories expand, systems encounter a pervasive failure mode known as **context drift**, where the semantic weight of initial instructions, critical constraints, and strategic goals is progressively diluted by the accumulation of subsequent, often less relevant, tokens. This degradation in coherence, coupled with the computational unsustainability of processing massive linear sequences, necessitates a fundamental architectural shift toward **Hierarchical Context Management (HCM)**.

The feasibility of integrating HCM techniques is not merely a question of availability but of architectural necessity. The analysis of current research indicates that linear context management strategies, such as sliding windows or naive Retrieval-Augmented Generation (RAG), are fundamentally insufficient for long-horizon tasks. They fail to preserve the "signal-to-noise" ratio required for high-fidelity reasoning. In contrast, HCM mimics human cognitive processes by organizing information into distinct layers of abstraction—separating strategic oversight from tactical execution, and global constraints from local state. This report provides an exhaustive analysis of the feasibility of these techniques, detailing the dominant architectural paradigms, specific implementation toolkits, and the comparative performance metrics that validate their adoption.

### 1.1 The Computational and Semantic Mechanics of Drift

To assess feasibility, one must first operationalize the specific mechanisms of failure that HCM aims to correct. Context drift is a composite phenomenon driven by both the mathematical properties of the Transformer architecture and the information dynamics of long interactions.

#### 1.1.1 Attention Dilution and the "Lost in the Middle" Phenomenon

The core limitation lies in the self-attention mechanism, which possesses a computational complexity of .1 As the input sequence length  increases, the model must calculate attention scores between every pair of tokens. This leads to two distinct problems. First, the **Attention Space Complexity** explodes; a 30-fold increase in context length results in a 900-fold increase in the memory required for attention matrices.1 Second, and more insidiously, the softmax function distributes probability mass across an ever-growing number of tokens. This results in **attention dilution**, where the model's ability to attend to a specific, critical instruction (e.g., a variable definition at the start of a 50,000-token prompt) is statistically weakened by the noise of thousands of intermediate tokens.

Empirical evidence suggests that models exhibit a U-shaped performance curve, known as the "Lost in the Middle" phenomenon. They retain high fidelity for tokens at the very beginning (Primacy Bias) and the immediate end (Recency Bias) of the context window but struggle significantly to retrieve and reason over information buried in the middle 50-80% of the sequence. For a project aiming to keep context from drifting, this means that simply extending the context window is a trap; it increases costs while degrading reasoning reliability.

#### 1.1.2 Context Pollution in Stateful Workflows

In "stateful" workflows—such as evolutionary coding agents or autonomous research assistants—the context naturally fills with trial-and-error attempts. Without active management, this accumulation creates **Context Pollution**. For example, in evolutionary optimization tasks, an agent may generate dozens of failed code hypotheses before finding a solution. If these failed attempts remain in the linear context, they bias the model toward repeating similar failure modes, creating a self-reinforcing loop of stagnation.2 This is distinct from simple length constraints; it is a degradation of the *quality* of the context. HCM addresses this by implementing "pruning" protocols that identify and excise low-value branches of history, ensuring that the context remains a high-signal repository of successful strategies rather than a graveyard of failed attempts.

#### 1.1.3 The Economic and Latency Barrier

Feasibility is also constrained by the economics of inference. During forward propagation, the model must calculate and retain intermediate results (activations) for each layer, known as the Key-Value (KV) cache.1 For a 64K token input, the VRAM consumption for these activations is massive, often exceeding the capacity of single enterprise-grade GPUs (e.g., NVIDIA A100 80GB). Furthermore, the latency of processing a massive linear prompt for every single token generation makes real-time interaction sluggish and unresponsive. HCM techniques like **HOMER** and **HCP** demonstrate that it is feasible to reduce this memory footprint from linear  to logarithmic , reducing memory requirements by over 70% and enabling high-throughput deployment on constrained hardware.1

### 1.2 The Feasibility Verdict

Based on the convergence of data from code completion benchmarks, document summarization tasks, and agentic framework evaluations, the integration of HCM is assessed as **highly feasible** and **strongly recommended**. The technology has matured beyond theoretical papers into implementable toolkits and protocols.

* **Availability:** Open-source implementations for hierarchical pruning (HCP-Coder), merging (HOMER), and agentic management (OpenPACEvolve, Confucius SDK) are currently available.6
* **Standardization:** The emergence of the **Model Context Protocol (MCP)** and **AGENTS.md** standards provides a low-friction pathway for integrating hierarchical context into existing IDEs and agent runtimes.9
* **Performance:** The performance gains are non-trivial. Systems implementing topological pruning for code see accuracy jumps from ~6% to ~54% compared to naive concatenation 1, while document summarizers reduce hallucination rates significantly.1

The following sections will detail the specific architectures available for integration, categorized by their operational mechanism: static inference-time optimization and dynamic agentic management.

## 2. Static Hierarchical Architectures: Inference-Time Optimization

For projects involving "stateless" tasks—such as code completion, document summarization, or question answering over massive corpora—the most feasible HCM techniques involve static restructuring of the input prompt. These methods do not necessarily require training new models (though they can benefit from it) but rely on sophisticated pre-processing algorithms to optimize information density.

### 2.1 HOMER: Hierarchical Context Merging

**HOMER** represents a generalized approach to compressing long contexts into manageable representations without retraining the underlying foundation model. It is particularly applicable to projects dealing with long linear texts where the "gist" is essential but token-for-token precision in the middle is negotiable.

#### 2.1.1 The Divide-and-Conquer Mechanism

HOMER operates on a "divide-and-conquer" principle. Instead of feeding a 100,000-token document into the model at once, the input is divided into manageable chunks (e.g., 2,048 tokens each). Unlike sliding window approaches which process chunks in isolation (losing global context), HOMER processes them collectively but merges them hierarchically at progressive transformer layers.1

* **Input Segmentation:** The text is split into chunks .
* **Hierarchical Merging:** As the data flows through the layers of the Transformer, adjacent chunks are concatenated. However, naive concatenation would reintroduce the length problem.
* **Token Reduction:** The critical innovation is **Propagative Refinement**. Before merging, HOMER applies a token reduction technique. It identifies tokens that receive minimal attention scores in the upper layers (which typically capture high-level semantic meaning) and retroactively prunes these tokens from the lower-layer representations.11

#### 2.1.2 Implementation Feasibility

The implementation of HOMER is highly feasible for projects using open-source models like Llama-2 or Llama-3. The official implementation provides a patch\_llama.py utility that modifies the standard HuggingFace LlamaForCausalLM object.11

* **Configuration:** Key parameters include max\_chunk\_len (defining the granularity of the hierarchy) and layers\_warmup (determining how deep into the network the merging begins).
* **Memory Efficiency:** By processing the chunks in a specific depth-first search (DFS) order and applying reduction, HOMER ensures that the peak memory usage scales logarithmically  rather than linearly. This allows a project to process significantly larger contexts on the same hardware footprint.1

### 2.2 HCP: Hierarchical Context Pruning for Code Repositories

For projects specifically focused on software engineering—such as an AI coding assistant or an automated refactoring agent—generic pruning methods often fail because they break syntactic dependencies. **Hierarchical Context Pruning (HCP)** is a domain-specific HCM technique that respects the rigid structure of programming languages.

#### 2.2.1 Topological Dependency Analysis

HCP begins by modeling the code repository not as text, but as a graph. Using parsers like Tree-sitter, it constructs a dependency graph where nodes are files or functions and edges represent import or call relationships.1

* **Dependency Levels:** The research identifies a steep drop-off in utility as dependency distance increases. Files at **Dependency Level 1** ( - direct imports) provide the vast majority of useful context. Files at  and beyond yield diminishing returns while consuming massive token budgets.1
* **Feasibility Insight:** A naive RAG system might retrieve snippets based on vector similarity, potentially pulling in unrelated code that shares keywords. HCP's topological approach ensures that the context is *structurally* relevant.

#### 2.2.2 Function-Level Granularity and Body Pruning

The most profound insight from the HCP research—and a key technique for implementation—is **Function-Level Pruning**. When a developer is writing code in File A that calls a function in File B, the LLM needs to know the *signature* (arguments, return type) and perhaps the *docstring* of the function in File B. It rarely needs to know the specific implementation details (the function body) of File B.

* **Pruning Strategy:** HCP retains the full content of the "Current File." For  dependency files, it parses the Abstract Syntax Tree (AST), retains class definitions and function headers, but strips out the function bodies.
* **Result:** This technique reduces the token count by approximately 80% (e.g., compressing a 50,000-token repository context into ~8,000 tokens).1
* **Performance:** Empirical testing on the CrossCodeEval benchmark shows that this pruning does not degrade accuracy; in fact, it significantly improves it (from 6.18% EM to 53.94%) by removing "syntactic noise" that distracts the model.1

### 2.3 CAHM: Context-Aware Hierarchical Merging for Documents

For projects involving the summarization or analysis of massive documents (100K+ tokens), such as legal discovery or scientific literature review, **Context-Aware Hierarchical Merging (CAHM)** offers a robust framework.

#### 2.3.1 Iterative Summarization vs. Context Injection

Standard hierarchical summarization builds a tree of summaries: chunk summaries  section summaries  document summary. However, this often leads to "hallucination snowballing," where an error in a leaf summary propagates up the tree. CAHM addresses this by injecting raw context back into the merging process.1

* **Support vs. Replace:** The feasibility analysis compares replacing intermediate summaries with raw context versus using raw context to *support* and refine the summaries. The "Extract-Support" strategy—where key sentences are extracted from the source to ground the intermediate summaries—proved most effective, achieving a Fact Precision of 56.3% compared to the 50.3% baseline.1
* **Unsupervised Data Expansion:** To train these models effectively, CAHM utilizes a "Root-Leaf Traversal" algorithm on document trees (parsed via tools like Grobid). This treats every path from the document root to a paragraph leaf as a standalone training example, significantly expanding the dataset size without human labeling.1

## 3. Dynamic Agentic Architectures: The "Context-as-a-Tool" Paradigm

While static methods optimize the input buffer, complex long-horizon tasks require active management. The **"Context-as-a-Tool" (Cat)** paradigm represents a shift where context manipulation is an explicit action space for the agent, rather than a passive environment variable. This is highly feasible for "Agentic" workflows.

### 3.1 COMPASS: Strategic Separation of Concerns

**COMPASS** (Context-Organized Multi-Agent Planning and Strategy System) is an architectural pattern that treats context management as a specialized role within a multi-agent team. It is designed to solve the "Context Management Bottleneck" where extended histories cause agents to lose strategic coherence.12

#### 3.1.1 The Triad Architecture

COMPASS decomposes the agent's cognition into three specialized components:

1. **The Main Agent:** Responsible for tactical execution (e.g., writing code, calling APIs). It operates with a **short-term working memory** that is aggressively pruned to keep it focused on the immediate sub-task.
2. **The Meta-Thinker:** A strategic observer that monitors the Main Agent's trajectory. It does not execute tasks; it evaluates progress against the high-level goal. If it detects drift (e.g., the agent is looping or deviating), it issues an intervention signal.12
3. **The Context Manager:** The core innovation. Upon receiving a signal from the Meta-Thinker, the Context Manager synthesizes a new "Context Brief." It pulls from a **Persistent Note Store** (long-term memory) and the recent interaction trace to construct a fresh, optimized prompt for the Main Agent.

#### 3.1.2 Implementation Details: The Context-12B Model

Implementing COMPASS is feasible because the "Context Manager" does not need to be a frontier model (like GPT-4). The research demonstrates that a smaller, specialized model (e.g., based on Gemma-12B) can be fine-tuned via Supervised Fine-Tuning (SFT) and Direct Preference Optimization (DPO) to excel at summarizing and organizing context. This makes the architecture economically viable, as the high-volume context processing is offloaded to a cheaper model.12

### 3.2 PACEvolve: Evolutionary Anti-Drift Mechanisms

For projects involving **self-improving agents** or optimization tasks (where an agent iterates on a solution), **PACEvolve** offers a blueprint for preventing "Context Pollution."

#### 3.2.1 Decoupling Ideas from Hypotheses

PACEvolve introduces a hierarchical distinction in memory:

* **Ideas:** High-level strategic concepts (e.g., "Use simulated annealing for optimization").
* **Hypotheses:** Concrete implementations of those ideas (e.g., the specific Python code for the annealer). By separating these, the system can prune context intelligently. If a specific hypothesis fails, it is recorded in "Failure Memory," but the parent "Idea" might remain valid. Conversely, if multiple hypotheses under one Idea fail, the entire Idea branch is pruned to force exploration of new strategies.2

#### 3.2.2 Momentum-Based Backtracking (MBB)

To prevent "Mode Collapse" (where an agent gets stuck in a local minimum), PACEvolve implements **Momentum-Based Backtracking**. The system tracks a "progress momentum" metric. If this momentum drops below a threshold—indicating stagnation—the system triggers a backtrack. This is effectively a "System Restore" for the agent's context, wiping out the recent drift and resetting the state to a previous high-potential node. This mechanism is crucial for long-running autonomous loops where drift is inevitable.2

### 3.3 Surfer 2: Visual Context Hierarchies

For multimodal agents that interact with GUIs (screens), **Surfer 2** demonstrates how to manage high-dimensional visual context.

* **Orchestrator vs. Navigator:** An "Orchestrator" maintains the global plan (e.g., "Book a flight to NYC"). It delegates a sub-goal to a "Navigator" (e.g., "Select the date on the calendar").
* **Context Isolation:** The Navigator executes the micro-actions using only the local visual context (screenshots of the calendar). Once the sub-goal is achieved, it reports success to the Orchestrator. Crucially, the Orchestrator *does not* ingest the dozens of intermediate screenshots generated by the Navigator. This prevents the global context from being flooded with pixel-level noise, allowing the agent to maintain coherence over long workflows (97.1% success on WebVoyager).13

## 4. Implementation Strategy: The Toolkit Approach

The query asks for "Another toolkit." The research reveals that there isn't a single monolithic software package; rather, there is an ecosystem of protocols, standards, and SDKs that can be assembled into a toolkit.

### 4.1 The AGENTS.md Standard: Low-Code Hierarchy

The most accessible entry point for implementing HCM is the **AGENTS.md** standard. This represents a file-system-based approach to context management that is compatible with most existing LLM developer tools (Cursor, Windsurf, Copilot).

* **Mechanism:** Instead of a single system prompt, context is distributed across the codebase in AGENTS.md files.
  + **Root AGENTS.md:** Contains global architectural rules, coding conventions, and high-level project goals.
  + **Directory-Level AGENTS.md:** Contains specific context for that module (e.g., /backend/AGENTS.md details the API schema, while /frontend/AGENTS.md details the state management library).
* **Integration:** Toolkits like mcp-http-agent-md or the AI Code Toolkit act as servers. When an agent is working in a specific file, the toolkit recursively crawls the directory tree, concatenating the relevant AGENTS.md files from the leaf up to the root. This automatically constructs a hierarchical context prompt relevant to the immediate task.10

### 4.2 Model Context Protocol (MCP): The Connectivity Layer

**MCP** is the emerging industry standard for connecting LLMs to external context. It serves as the "plumbing" for HCM.

* **Hierarchical Servers:** Developers can build custom MCP servers that implement the logic of HCP or HOMER. For example, a code-context-server could accept a file path, perform the topological dependency analysis described in Section 2.2, prune function bodies, and return the compressed context to the LLM client.9
* **Progressive Discovery:** The AI Code Toolkit implements "progressive discovery" via MCP. Instead of dumping all rules into the context, it exposes schemas hierarchically. The agent "navigates" the context tree via tool calls, loading only the schemas necessary for the current step, thereby preserving token budget.16

### 4.3 The Confucius SDK: Agent Scaffolding

For building complex, multi-agent systems like COMPASS, the **Confucius SDK** provides the necessary scaffolding.

* **Persistent Note-Taking:** It includes modules for "Note-Taking Agents" that automatically distill interaction trajectories into persistent Markdown notes. This creates a "Long-Term Memory" that survives beyond the immediate context window.
* **Meta-Agent Support:** The SDK is architected to support "Meta-Agents" that can oversee and modify the behavior of sub-agents, facilitating the implementation of the "Meta-Thinker" pattern.8

### 4.4 L-RAG: Entropy-Based Gating for Efficiency

For projects where latency is a concern, **L-RAG** (Lazy Retrieval-Augmented Generation) offers a lightweight hierarchical approach.

* **Mechanism:** It employs a two-tier architecture. Tier 1 attempts to answer queries using a pre-computed "Context Summary." Tier 2 (Full Retrieval) is only triggered if the model's **predictive entropy** (a measure of uncertainty) exceeds a calibrated threshold.
* **Feasibility:** This is highly feasible to implement as a wrapper around any RAG pipeline. It reduces the average cost per query by avoiding expensive retrievals for simple questions while maintaining high accuracy for complex ones.18

## 5. Comparative Performance Analysis

To validate the feasibility of these approaches, we present a comparative analysis of performance metrics derived from the research snippets.

### 5.1 Code Completion: HCP vs. Naive Approaches

Data from the CrossCodeEval benchmark using Repo-Code LLMs (DeepseekCoder, Starcoder2) demonstrates the superiority of topological pruning (HCP) over simple RAG or full-context concatenation.

| **Metric** | **Random-All (Concatenation)** | **RAG-BM25 (Flat Retrieval)** | **HCP (Hierarchical Pruning)** | **Impact Analysis** |
| --- | --- | --- | --- | --- |
| **Input Length** | > 50,000 tokens | ~4,000 tokens | **~8,000 tokens** | HCP reduces context by ~84% vs. full repo, fitting standard windows. |
| **Accuracy (Exact Match)** | 6.18% | 17.28% | **53.94%** | HCP provides a **3x improvement** over RAG and massive gain over truncation. |
| **Throughput** | Low (OOM risk) | High | **High** | Reduced token count significantly lowers latency and VRAM usage. |
| **Error Modes** | High hallucination of non-existent methods | Misses dependencies | **Low** | Maintains semantic contract via function signatures. |

Source: 6

**Insight:** The data explicitly validates the hypothesis that "syntactic noise" (function bodies) harms performance. By pruning it, HCP not only saves memory but actively helps the model focus on the relevant API contracts.

### 5.2 Document Summarization: CAHM vs. Zero-Shot

In the domain of long-document summarization (>100K tokens), Context-Aware Hierarchical Merging significantly outperforms standard baselines.

| **Metric** | **Zero-Shot (Truncated)** | **HMerge (Standard)** | **CAHM (Extract-Support)** | **Impact Analysis** |
| --- | --- | --- | --- | --- |
| **Fact Precision** | 50.3% | 55.1% | **56.3%** | Context augmentation reduces hallucinations. |
| **Fact Recall** | 37.1% | 38.8% | **41.1%** | Refinement using source context captures more key information. |
| **AlignScore** | 79.3 | 72.9 | **79.4** | CAHM maintains alignment with source despite summarization. |

Source: 1

**Insight:** The "Support" strategy (refining abstractive summaries with retrieved extractive context) consistently outperforms "Replace" strategies. This suggests that the abstractive summary provides a necessary "scaffold" or narrative arc, while the extracted context provides the "bricks" of factual data.

### 5.3 Long-Horizon Agent Performance

For autonomous agents, the COMPASS architecture demonstrates that hierarchical context is the defining factor for success in long tasks.

| **Benchmark** | **Single-Agent Baseline** | **COMPASS (Hierarchical)** | **Improvement** |
| --- | --- | --- | --- |
| **GAIA** (General AI Assistant) | ~30-40% (est.) | **Higher** | Up to **20%** relative improvement over baselines. |
| **WebVoyager** (Web Navigation) | Varies | **97.1%** (Surfer 2) | Hierarchical visual context enables near-perfect navigation. |

Source: 12

## 6. Challenges and Risk Mitigation

While feasible, the integration of HCM introduces specific complexities that must be managed.

### 6.1 The "Parsing Cliff"

Methods like HCP and HCA rely heavily on accurate parsing (AST for code, Document Trees for PDFs).

* **Risk:** If Tree-sitter fails to parse a syntax error in a file, or if Grobid misidentifies a section header, the hierarchical structure collapses. This introduces "structural noise," which can be more damaging than random noise because the model assumes the structure is meaningful.1
* **Mitigation:** Implementation must include robust fallback mechanisms. If parsing fails, the system should degrade gracefully to chunk-based processing rather than crashing or feeding corrupted hierarchies.

### 6.2 Latency and Cost of Orchestration

Dynamic architectures like COMPASS require running multiple agents (Meta-Thinker, Context Manager) in parallel or sequence.

* **Risk:** This triples the inference cost and latency for every reasoning step.
* **Mitigation:** The "Context-12B" approach 12 is critical here. By offloading the context management to a smaller, distilled model, the system keeps the heavy (expensive) reasoning on the main model while minimizing overhead.

### 6.3 Coordination Drift

In multi-agent hierarchies, there is a risk that the "Context Manager" might aggressively prune information that the "Main Agent" actually needed (e.g., a subtle constraint mentioned 50 turns ago).

* **Mitigation:** The **Cat Paradigm** mitigates this by allowing the Main Agent to *request* context. Instead of being passive recipients, agents should have tools to query\_memory() if they feel context is missing.20

## 7. Strategic Recommendations and Roadmap

The feasibility analysis concludes that integrating Hierarchical Context Management is not only possible but is the optimal path forward for preventing context drift. We recommend a phased integration roadmap:

### Phase 1: Structural Foundations (Weeks 1-4)

* **Adoption of Standards:** Immediately restructure the project's data/codebase to align with the **AGENTS.md** standard. Create root and directory-level context files.
* **Parser Integration:** Integrate Tree-sitter (for code) or Unstructured (for documents) into the data ingestion pipeline to enable topological analysis.
* **Static Pruning Implementation:** Implement the **HCP** logic for RAG: identify  dependencies and prune function bodies. This will yield immediate latency and accuracy gains for RAG workflows.

### Phase 2: The "Context-as-a-Tool" Integration (Weeks 5-8)

* **Tool Provisioning:** Expose context management functions to the LLM. Give the agent tools to save\_to\_long\_term\_memory(key, value) and summarize\_task\_state().
* **MCP Server Deployment:** Wrap the Phase 1 logic into an **MCP Server**. This allows any MCP-compliant client (like Claude Desktop or Cursor) to leverage the hierarchical context immediately.

### Phase 3: Agentic Architecture (Months 3+)

* **Deploy the "Meta-Thinker":** If the project involves long-horizon autonomous tasks, implement the COMPASS pattern. Deploy a secondary lightweight model (e.g., Llama-3-8B) to act as the "Context Manager," continuously monitoring and refreshing the main agent's working memory.
* **Evolutionary Loops:** For optimization tasks, integrate **PACEvolve** principles. Implement "Momentum-Based Backtracking" to automatically reset context when the agent stagnates.

In conclusion, Hierarchical Context Management transforms context from a passive, decaying buffer into an active, structured asset. By shifting from linear processing to hierarchical management—utilizing tools like HCP for static pruning and COMPASS for dynamic oversight—development teams can effectively solve the context drift problem and unlock the next generation of reliable, long-horizon AI applications.

#### Works cited

1. hierachical\_context\_augementation.pdf
2. PACEvolve: Enabling Long-Horizon Progress-Aware Consistent Evolution - arXiv, accessed February 7, 2026, <https://arxiv.org/pdf/2601.10657>
3. 𝙿𝙰𝙲𝙴𝚟𝚘𝚕𝚟𝚎: Enabling Long-Horizon Progress-Aware Consistent Evolution - arXiv, accessed February 7, 2026, <https://arxiv.org/html/2601.10657v2>
4. [2404.10308] Hierarchical Context Merging: Better Long Context Understanding for Pre-trained LLMs - arXiv, accessed February 7, 2026, <https://arxiv.org/abs/2404.10308>
5. HIERARCHICAL CONTEXT MERGING: BETTER LONG CONTEXT UNDERSTANDING FOR PRE-TRAINED LLMS - ICLR Proceedings, accessed February 7, 2026, <https://proceedings.iclr.cc/paper_files/paper/2024/file/06694da057cb15fef11542270a592627-Paper-Conference.pdf>
6. Hambaobao/HCP-Coder: Hierarchical Context Pruning ... - GitHub, accessed February 7, 2026, <https://github.com/Hambaobao/HCP-Coder>
7. hassenhamdi/OpenPACEvolve - GitHub, accessed February 7, 2026, <https://github.com/hassenhamdi/OpenPACEvolve>
8. Confucius Code Agent: An Open-sourced AI Software Engineer at Industrial Scale - arXiv, accessed February 7, 2026, <https://arxiv.org/html/2512.10398v1>
9. modelcontextprotocol/servers: Model Context Protocol Servers - GitHub, accessed February 7, 2026, <https://github.com/modelcontextprotocol/servers>
10. Benhao Tang's AGENTS.md MCP Server: A Guide for AI Engineers - Skywork.ai, accessed February 7, 2026, <https://skywork.ai/skypage/en/agents-md-mcp-server-guide-ai-engineers/1980903899357851648>
11. alinlab/HOMER: Official implementation of Hierarchical ... - GitHub, accessed February 7, 2026, <https://github.com/alinlab/HOMER>
12. COMPASS: Enhancing Agent Long-Horizon Reasoning with Evolving Context - arXiv, accessed February 7, 2026, <https://arxiv.org/pdf/2510.08790>
13. (PDF) Surfer 2: The Next Generation of Cross-Platform Computer Use Agents, accessed February 7, 2026, <https://www.researchgate.net/publication/396848255_Surfer_2_The_Next_Generation_of_Cross-Platform_Computer_Use_Agents>
14. Surfer 2 The Next Generation of Cross-Platform Computer Use Agents - arXiv, accessed February 7, 2026, <https://arxiv.org/html/2510.19949v1>
15. AGENTS.md, accessed February 7, 2026, <https://agents.md/>
16. AgiFlow/aicode-toolkit: Toolkit for Coding Agents to work ... - GitHub, accessed February 7, 2026, <https://github.com/AgiFlow/aicode-toolkit>
17. Confucius SDK: AI Agent Development Scaffold - Emergent Mind, accessed February 7, 2026, <https://www.emergentmind.com/topics/confucius-sdk>
18. Arxiv今日论文| 2026-01-13 - 闲记算法, accessed February 7, 2026, <http://lonepatient.top/2026/01/13/arxiv_papers_2026-01-13>
19. Adaptive-RAG: Learning to Adapt Retrieval-Augmented Large Language Models through Question Complexity | Request PDF - ResearchGate, accessed February 7, 2026, <https://www.researchgate.net/publication/382632436_Adaptive-RAG_Learning_to_Adapt_Retrieval-Augmented_Large_Language_Models_through_Question_Complexity>
20. Context-as-a-Tool (Cat) Paradigm - Emergent Mind, accessed February 7, 2026, <https://www.emergentmind.com/topics/context-as-a-tool-cat-paradigm>