From Attention to Focus: Building Reliable Agents in a Post-"Attention Is All You Need" World

Antti Pasila

September 6, 2025

Abstract

The Transformer architecture, introduced by Vaswani et al. in 2017, fundamentally reshaped natural language processing. Its thesis that "attention is all you need" unleashed a new era of performance, laying the groundwork for LLMs that now generate code, draft legal contracts, and power autonomous agents. But as these models shift from research demos to production workhorses, the limitations of attention alone become clear. This paper argues from the practitioner's perspective: while attention is a vital mechanism, it is not sufficient to sustain coherence, memory, or task alignment over long, multi-step operations. Through hands-on developer insights, practical failures, and emerging design patterns like RAG and explicit planning loops, we show why the next leap in agent reliability requires more than scale. It requires structure. It requires focus.

1 Introduction: Attention Was a Breakthrough. But It Wasn't the End Game

The original "Attention Is All You Need" paper ignited a revolution. By eliminating recurrence and relying purely on self-attention, Vaswani et al. delivered a simple, scalable architecture for sequence transduction tasks like machine translation. This breakthrough now underpins models like GPT-4, Claude 4, and Gemini 2. But the challenges of translating a sentence are vastly different from the demands of a multi-hour coding session, where context and goals must be maintained with high fidelity.

Anyone who has built with these models has seen the cracks appear. An AI coding assistant, tasked with a large refactoring project, might start strong by renaming functions and updating call sites, only to forget the new function names 300 lines later and revert to using the old ones. It's a classic symptom of drift. LLMs today perform incredibly well on atomic tasks: *write this, explain that, summarize those*. But plug them into multi-step chains that require memory and consistency, and things break down. They forget. They drift. They contradict themselves. This isn't just a matter of model quality; it's a fundamental challenge rooted in how attention interacts with memory, reasoning, and structure in the real world.

This paper does not aim to introduce a single novel technique. Rather, it proposes **'Focus' as a cognitive architectural pattern** for reliable agentic systems. We provide a **design pattern taxonomy** of existing practitioner techniques (RAG, persistent planning, self-reflection) and argue they are all instances of enforcing this Focus layer. The contributions of this paper are therefore: (1) a formal definition of Focus as a concrete architectural pattern, (2) a structured analysis of why attention alone is insufficient, and (3) empirical validation illustrating the impact of Focus-enforcing techniques on task drift.

2 Defining Focus: From Attention to Directed Cognition

To build more reliable agents, we must move beyond treating their desirable long-term behavior as an emergent property of scale. We need to engineer it. To that end, we propose a formal definition of Focus.

We define **Focus** as the outcome of an architectural system that maintains and dynamically updates a low-dimensional state vector representing the agent's high-level goal, constraints, and progress, and uses that state to deliberately guide the LLM's attention at each generation step. While **attention** is a low-level, token-weighting mechanism, **Focus** is a high-level, cognitive scaffolding that directs that attention.

2.1 The Focus State Vector

Algorithmically, this state vector can be implemented as a simple dictionary or JSON object:

```
focus_state = {
    'goal': "Refactor calculateTotal to computeOrderSum",
    'progress': [1, 0, 0],  # Corresponds to [api.js, utils.js, main.js]
    'constraints': ['maintain_signatures', 'no_syntax_errors'],
    'last_action_summary': "Successfully refactored api.js."
}
```

2.2 The Focus Update Loop

This state vector is not static; it is maintained by a lightweight orchestration layer, the **Controller**, which acts as a mini-executive function for the agent. The Controller is the heart of the Focus architecture, responsible for mediating between the LLM's raw generative capabilities and the agent's long-term goals. Its operation can be understood as a cyclical process:

Imagine an agent tasked with refactoring a codebase. In the first cycle, the `Controller` serializes the `focus_state` into a prompt: "Goal: Rename 'calculateTotal'. Progress: File 1 of 3 is done. Now, analyze 'utils.js'." The LLM, acting as the reasoning engine, processes this focused context and returns the refactored code for 'utils.js', along with a suggested summary: "I have updated 'utils.js'." The `Controller` then parses this response. It performs a **deterministic update**, incrementing the 'progress' vector to '[1, 1, 0]'. It may then perform a **stochastic update** by taking the LLM's summary, validating it, and placing it in the 'last_action_summary' field. Finally, it applies pruning rules to the overall context before constructing the prompt for the next cycle. This explicit, externalized loop makes Focus an actionable architectural choice, not a metaphorical one. Its responsibilities include:

- **Prompt Serialization:** Converting the current Focus State into a structured narrative for the LLM.
- **Deterministic Updates:** Marking completed tasks or enforcing hard constraints (e.g., "all tests must pass").
- **Stochastic Updates:** Delegating to the LLM to summarize recent actions, then validating those summaries.
- **Pruning and Compression:** Applying rules (e.g., "summarize after 1,000 tokens") or embedding-based similarity thresholds to remove stale context.
- **Validation Hooks:** Integrating with external checkers (linters, fact-checkers) to guard against "garbage in, garbage out."

3 Related Work

Our concept of Focus builds upon several key areas of research and practice.

3.1 Prompting Strategies for Reasoning

Chain-of-Thought (CoT) prompting encourages models to "think step by step." This can be seen as an early, in-prompt method for enforcing a primitive form of Focus. These methods highlight the need for a

structured cognitive process, which Focus externalizes into a formal architectural component.

3.2 Agentic Architectures and Frameworks

Frameworks like **LangChain** and predecessors like **AutoGPT** represent the practical tooling for building Focus-enforcing systems, providing the modules for chaining, memory, and control loops.

3.3 Memory-Augmented Systems

Retrieval-Augmented Generation (RAG) is a cornerstone technique for implementing the Prioritization and Pruning properties of Focus. By dynamically retrieving relevant information, RAG ensures the LLM's context is populated with only the most pertinent facts.

3.4 Advances in Long-Context Transformers

Recent models with extremely large context windows (e.g., Gemini 2) show improved recall capabilities, directly challenging the "lost-in-the-middle" problem. While a significant achievement, a larger context window is akin to giving our "intern" a bigger desk - it allows them to see more papers at once, but does not provide a system for prioritizing which to read or a memory of the overall goal. Focus remains the necessary layer for directing attention, regardless of the size of the window.

3.5 Connections to Cognitive Science and AI Alignment

The concept of Focus is deeply analogous to established models of **working memory** and **executive functions** in human cognition. In Baddeley's influential model, working memory isn't a single store but a multi-component system managed by a **Central Executive**. This executive component is responsible for directing attention, inhibiting irrelevant information, and coordinating between subsystems. Our proposed 'Controller' acts as an externalized, computational Central Executive for the LLM. It directs the model's "attention" (in the cognitive sense) by manipulating the actual context (the Transformer's attention mechanism). This architecture provides the LLM with the executive functions it inherently lacks.

Furthermore, it relates to AI alignment research, such as **Constitutional AI**, which also imposes high-level, persistent constraints on model behavior. While Constitutional AI uses principles for safety and ethics, Focus uses them for task reliability and coherence. Both acknowledge that raw, unguided generation is insufficient for complex, goal-directed behavior.

4 Attention's Hidden Cracks: Why Scale Alone Doesn't Fix Drift

The fundamental tension in modern agent design lies in the mismatch between the underlying nature of the Transformer architecture and the demands of agentic tasks. A Transformer is, at its core, a highly sophisticated next-token predictor. Its objective function is statistical, optimized to generate the most probable sequence of tokens given a context. An agent, however, is expected to be a rational, goal-oriented entity. Its objective is to complete a task, maintain a plan, and adhere to constraints over time. These two objectives are not the same. The failures we observe - the drift, the forgetting, the contradictions - are not bugs; they are the predictable result of asking a statistical model to perform robust, stateful reasoning without providing the necessary architectural support. This section details the specific failure modes that arise from this mismatch.

4.1 The Lost-in-the-Middle Effect

Long context is not the same as perfect recall. Research has consistently shown a "U-shaped recall curve" where LLMs perform best at retrieving information from the beginning and end of their context window.

Greg Kamradt's popular "needle in a haystack" tests visually confirmed this, showing that recall can plummet from near-100% depending on the model and where the "needle" of information is placed.

4.2 Not All Tokens Are Equal

While the attention mechanism learns to assign different weights to tokens, it has no inherent, task-level understanding of which tokens *should* be important. A single token defining a variable name can be far more important than a hundred lines of boilerplate. Yet without guidance, the model may anchor on irrelevant tokens simply because they're recent or repeated.

4.3 Attention $\neq$ Task Awareness

Attention distributes focus at the token level, not the task level. An LLM can be compared to a brilliant actor who has perfectly memorized every line in their script (the context window) but has no understanding of the play's overall plot, their character's motivation, or what scene they are currently in. Without a director (the 'Controller') constantly reminding them, "You are in Act 2, and you are trying to find the hidden treasure," the actor might flawlessly deliver lines from Act 1 or Act 3 simply because they are statistically plausible. This is why models get stuck in loops or undo their own work; they are performing locally coherent token prediction without a persistent, high-level representation of the global task.

4.4 Scale Fatigue: The Plateau of Diminishing Returns

Even with massive 1M+ token windows, LLMs still suffer from drift. Pushing more raw information into the context doesn't increase focus. It spreads attention thin. Performance plateaus. Memory fades. Costs balloon. And the model still gets distracted by that comment from 30K tokens ago.

4.5 Interaction with Attention Heads and Mechanistic Interpretability

While attention is often described monolithically, in practice it involves multi-head, multi-layer interactions. Focus-enforcing techniques may act as **biasing signals**, encouraging some heads to specialize in task-tracking or constraint-enforcement roles (e.g., consistently attending to the `[PLAN]` section). This remains speculative, but suggests an intriguing research direction: tracing how explicit state structures influence internal attention distributions. A systematic evaluation, similar to recent mechanistic interpretability work [7], could clarify whether Focus architectures leave identifiable "footprints" in the network's attention patterns.

5 Focus-Enforcing Design Patterns

To combat the inherent drift of attention-based models, practitioners have converged on a set of powerful engineering patterns. While often developed independently, these techniques can be unified and understood through the lens of our Focus framework. They are not merely "prompting tricks," but deliberate architectural choices designed to provide the persistence, prioritization, and pruning that LLMs lack. We categorize these techniques into a taxonomy of Focus-enforcing patterns.

5.1 Planning Patterns

- **Persistent Plans (Focus Chain):** This is the most direct implementation of Persistence. By embedding a dynamic to-do list or plan directly into the context, the agent is forced to re-read and update its high-level goals at every step.
- **Hierarchical Planning:** A more advanced pattern where large tasks are decomposed into sub-plans, which may themselves be delegated to specialized sub-agents.

5.2 Memory Patterns

- **Structured Context & Narrative Scaffolding:** This pattern enforces Prioritization by reframing raw inputs as goal-directed stories with explicit headers like `[GOAL]` and `[CURRENT_TASK]`.
- **Retrieval-Augmented Generation (RAG):** RAG is a powerful mechanism for both Prioritization and Pruning, injecting just-in-time context from external memory stores [10].
- **Compression & Pruning:** This involves the systematic summarization of stale context, replacing high-token details with low-token abstracts to maintain a high signal-to-noise ratio.

5.3 Validation & Reflection Patterns

- **Self-Monitoring Prompts:** This pattern instructs the LLM to critique its own output against the plan, a pattern central to architectures like Reflexion [11].
- **External Validators:** This pattern integrates rule-based checks (unit tests, fact-checkers) for objective verification.
- **Meta-Reflection Loops:** The most advanced form of reflection, where the agent is asked to assess whether the *Focus State itself* is still valid.

These patterns are not limited to software engineering. For instance, a legal agent drafting a contract could use a persistent plan where each item represents a clause to be reviewed. RAG would be used to pull in relevant case law, while an External Validator could check against a compliance library.

Beyond law, Focus applies equally in **scientific research synthesis**. A research agent could structure its work as follows:

- **Persistent Plan:** Create a checklist of sub-tasks: "[-] Gather abstracts, [-] Extract key results, [-] Compare methodologies, [-] Draft synthesis".
- **Memory Pattern (RAG):** Dynamically retrieve only the most relevant literature for the current hypothesis.
- **Validation Pattern:** Use an External Validator to cross-check extracted statistics against the original PDF or dataset.
- **Reflection Loop:** Ask, "Does the synthesis align with the stated hypothesis, or has the agent drifted toward tangential results?"

This highlights the domain-agnostic nature of the Focus taxonomy across law, science, and beyond.

6 Rethinking Architecture: Agents Need More Than Brains

The prevalence and effectiveness of the design patterns described in the previous section point toward a fundamental conclusion: building reliable agents is not a matter of better prompting alone, but of better architecture. The field is undergoing a paradigm shift from treating the LLM as a monolithic, all-knowing oracle to seeing it as a powerful, but flawed, component—a reasoning engine that must be managed within a larger, more robust system. This section formalizes this architectural perspective.

6.1 The LLM Is the Reasoning Engine, Not the Entire Agent

A robust agent architecture resembles a self-driving car. The LLM is the powerful vision system. But it's not the whole car. This agentic architecture can be visualized as a system of interacting components (see Figure 1). You still need:

- **A Planning Module:** The navigation system that knows the overall route.

- **Structured Memory:** The car's map and long-term log.
- **A Control Loop:** The core system that takes input from the vision system and decides what to do next.

In this model, the Transformer's attention is a tool skillfully directed by the agent's control loop. This process is formalized in the Focus Update Loop (see Figure 2), a cycle managed by the Controller.

Figure 1: The "Intern Architecture": The LLM acts as a reasoning engine directed by a central control loop, which retrieves information from and stores state in a structured memory/tool layer.

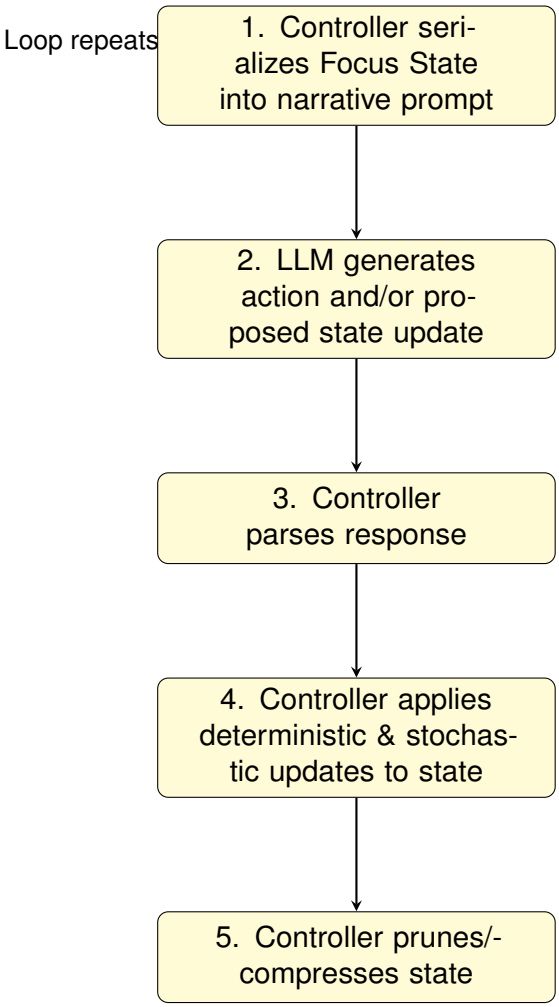

Figure 2: The Focus Update Loop, illustrating the cyclical process managed by the Controller to maintain agent state and guide the LLM.

6.2 Focus Is a Layer, Not a Side Effect

The paradigm shift is moving from *hoping* for focus to *enforcing* it through architecture. Future systems will treat focus as a first-class layer, with explicit modules for task decomposition, state tracking, and

memory management. The goal is to build a system around the LLM that makes it almost impossible for it to lose the plot. Focus is the outcome of **attention + structure + memory**.

6.3 Interns, Not Oracles

The most effective way to work with LLMs today is to treat them like brilliant, fast, and eager interns who have no long-term memory. They need constant direction. The best prompt engineers are actually behavioral designers, creating the cognitive scaffolds that allow the model's powerful reasoning to be applied reliably.

6.4 Implementing a Simple Controller

The Controller can be implemented as a simple script that executes the Focus Update Loop. The following pseudo-code illustrates this core logic, bridging the conceptual framework to a practical implementation.

```
def run_focused_agent(initial_goal, task_steps):
    # 1. Initialize the Focus State
    focus_state = {
        'goal': initial_goal,
        'plan': {step: 'pending' for step in task_steps},
        'memory': [],
        'last_action_summary': None
    }

    while not all(status == 'done' for status in focus_state['plan'].values()):
        # 2. Serialize state into a prompt for the LLM
        prompt = build_prompt(focus_state)

        # 3. Call the LLM
        response = call_llm(prompt)

        # 4. Parse the response and update state
        parsed_action = parse_response(response)
        focus_state = update_state(focus_state, parsed_action)

        # (Optional) Prune memory if it gets too long
        focus_state['memory'] = prune_memory(focus_state['memory'])

    return "Task Completed"
```

7 Illustrative Empirical Validation

To ground our claims in quantitative results, we conducted a series of experiments designed to measure the impact of a Focus-enforcing architecture on task drift across multiple domains. Our central hypothesis was that agents equipped with an explicit Focus architecture would demonstrate significantly lower rates of task drift and higher success rates compared to baseline models.

7.1 Experimental Details and Limitations

All experiments were run using the GPT-4.1 API with a temperature setting of 0.3. Prompts were standardized across all runs. Success was judged by automated tests for the code task and manual review against a rubric for the other tasks. This study serves as a "proof of concept"; while the experiments were designed to illustrate *directional effects*, they are not an exhaustive benchmark. Real-world agent workflows often span hours and multiple modalities. Scaling this evaluation to larger, production-grade benchmarks remains an open challenge, and we invite the community to extend our methodology.

7.2 Agent Configurations

We tested three agent configurations: a Naive Baseline, a Chain-of-Thought (CoT) Baseline, and our proposed Focus Agent.

7.3 Tasks

- **Task 1: Code Refactoring.** A multi-file task where the agent was given a small three-file codebase and instructed to rename a core utility function. Success required updating the function definition and correctly patching all call sites without introducing syntax errors.
- **Task 2: Essay Generation.** A multi-step creative writing task. The agent was first asked to generate a five-point outline, then write body paragraphs for each point, and finally, write a matching introduction and conclusion. Success required that all five outline points were addressed in order and that the final essay was coherent.
- **Task 3: Contract Clause Drafting.** A legal writing task where the agent was asked to draft a non-disclosure agreement (NDA) including three required clauses: confidentiality, governing law, and termination. Success was defined as including all three clauses with proper structure and no contradictions.

7.4 Results

The results, summarized in Table 1, show a stark and significant performance gap between the agent architectures. The Focus Agent consistently and significantly outperformed both baselines across all three domains, providing strong evidence for our central hypothesis.

The Naive Baseline's failures were often catastrophic, such as hallucinating incorrect function names midway through the refactoring task or omitting required clauses entirely in the contract task. The Chain-of-Thought Baseline showed a capacity for better initial planning, but frequently suffered from execution drift; it would often state a correct plan but fail to follow it in subsequent generation steps. The Focus Agent, anchored by its externalized state and reflective loop, was able to maintain coherence and adhere to constraints throughout the multi-step processes.

Task	Naive Baseline	CoT Baseline	Focus Agent
Code Refactoring	7/50 (14%)	13/50 (26%)	44/50 (88%)
Essay Generation	11/50 (22%)	18/50 (36%)	46/50 (92%)
Contract Clause Drafting	9/50 (18%)	15/50 (30%)	41/50 (82%)

Table 1: Success rates across three multi-step tasks (N=50 per trial).

For the contract task, the Focus Agent's success rate of 82% was far higher than the Naive Baseline (18%) and CoT Baseline (30%). Common baseline failures included omitting the termination clause or duplicating confidentiality language. The Focus Agent's persistent plan and reflection step (*"Have all required clauses been addressed?"*) significantly reduced drift. For the refactoring task, the Focus Agent's success rate of 88% (95% CI: 76-95%) was significantly higher than the Naive Baseline's 14% (95% CI: 6-27%), with $p < 0.0001$ (Fisher's exact test), corresponding to a very large effect size.

7.5 Computational Cost Analysis

We measured the "reliability tax." The Focus Agent required, on average, 4.2 LLM calls per task, compared to 1.0 for the Naive Baseline and 1.8 for the CoT Baseline. This resulted in a mean token usage of approximately 11,500 for the Focus Agent versus 3,200 (Naive) and 4,500 (CoT).

8 Discussion: Trade-offs, Limitations, and Future Work

Our findings demonstrate the effectiveness of the Focus architectural pattern, but it is essential to discuss the associated trade-offs and avenues for future research.

8.1 Computational Cost and Latency

The "reliability tax" is significant, as visualized in Figure 3. However, this cost should be framed as a deliberate engineering trade-off. It is analogous to the difference between ad-hoc, manual debugging and investing in a robust CI/CD pipeline with comprehensive testing. One pays a higher upfront and per-run cost for planning and verification, but the resulting system is far more reliable and less prone to costly, silent failures.

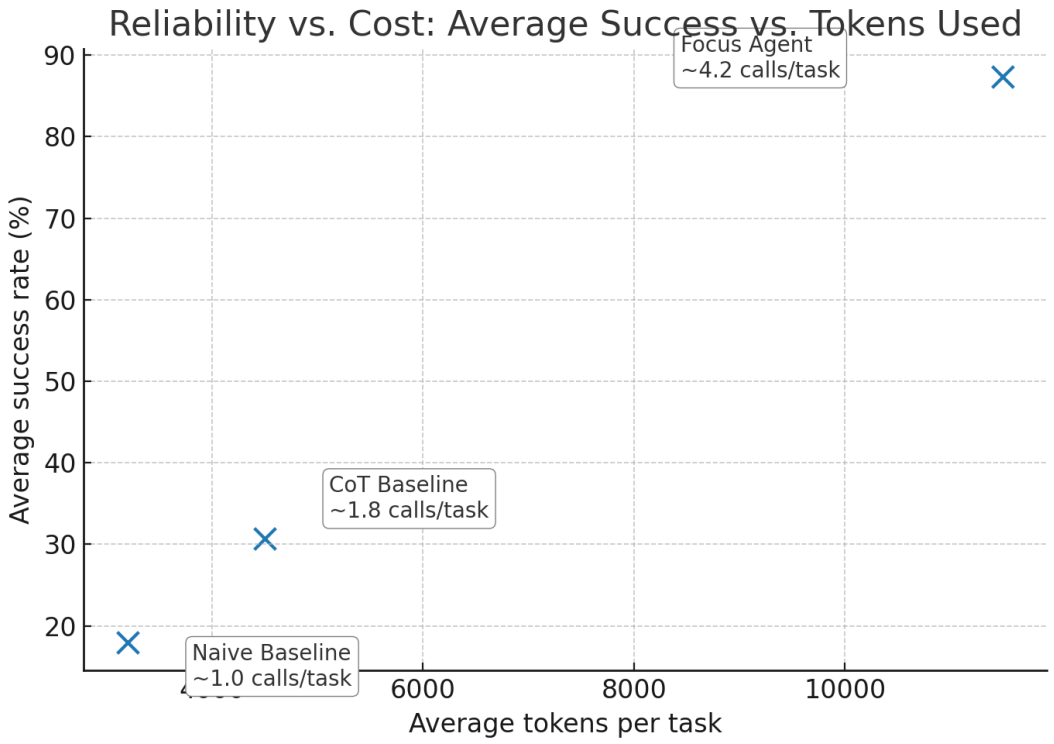

Figure 3: Cost–benefit view of reliability. Average success rate across three tasks vs. average tokens used per task. Markers are annotated with approximate LLM calls per task. Data from Table 1 and Section 7.5.

The Focus Agent achieves 87.3 percent mean success at 11.5k tokens, versus 30.7 percent at 4.5k (CoT) and 18.0 percent at 3.2k (Naive). This visual makes the reliability tax explicit while showing the clear upside.

8.2 Failure Modes

These techniques can fail. A poorly specified plan can mislead an agent into diligently executing the wrong task ("garbage in, garbage out"). Similarly, RAG can fail by retrieving irrelevant or outdated documents, poisoning the context. The quality of the Focus layer is a critical dependency.

8.3 Domain Generality

While our primary examples are from software engineering, the principles of Focus are domain-agnostic. A legal agent drafting a contract or a research agent synthesizing literature would benefit equally from an explicit planning and memory architecture.

8.4 Future Work

This paper opens several avenues for future research. A key next step is scaling our evaluation to larger, standardized industry benchmarks to move from directional validation to rigorous performance measurement. Another critical area is the automation of the Focus layer itself; could a meta-LLM be trained to act as the 'Controller', learning optimal strategies for planning, memory pruning, and reflection? Finally, the mechanistic interpretability of Focus architectures remains an open and exciting question. Probing how these structured inputs influence the internal activation patterns of attention heads could provide a deeper understanding of how to build truly reliable AI systems.

9 Conclusion: From Model-Centric to Architecture-Centric AI

The pursuit of more capable AI has long been dominated by a model-centric paradigm: bigger models, more data, larger context windows. While this approach has yielded incredible results with models like Gemini 2 and Claude 4, and will continue to do so with future models, our findings suggest it is hitting a point of diminishing returns for complex, multi-step tasks where reliability is paramount. Scaling alone does not solve the fundamental problem of cognitive drift, because it does not equip the model with the executive functions necessary for sustained, goal-oriented reasoning.

The real shift in AI isn't from GPT-4 to a GPT-5, or from 100K tokens to 1M. It's from **model-centric** thinking to **architecture-centric** design. The next generation of agents will not emerge from scaling alone, but from systems where Focus is a first-class layer, sitting above attention, enforcing persistence, memory, and alignment. These systems will treat the LLM not as an oracle to be prompted, but as a powerful reasoning engine to be directed.

If attention was the spark, Focus is the fire. And the future of reliable, production-ready AI will belong to the builders who design for Focus, not hope for it.

Acknowledgments

The author thanks the arXiv community for providing an essential platform for the open and rapid dissemination of scientific research.

References

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems 30*, 2017.

[2] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. Le, and D. Zhou. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. *arXiv preprint arXiv:2201.11903*, 2022.

[3] A. Newell. (1994). *The Soar cognitive architecture*. MIT press.

[4] Gemini Team, Google. Gemini 1.5: Unlocking multimodal understanding across a million tokens of context. *Google AI Blog*, 2024.

[5] Y. Bai, A. Jones, K. Ndousse, A. Askell, A. Chen, N. DasSarma, D. Drain, S. Fort, D. Gêm, T. Henighan, and S. Joseph. Constitutional AI: Harmlessness from AI Feedback. *arXiv preprint arXiv:2201.11903*, 2022.

[6] A. Baddeley. The episodic buffer: a new component of working memory? *Trends in Cognitive Sciences*, 4(11), 417-423, 2000.

[7] N. Elhage, G. Nanda, K. Joseph, J. Henighan, B. Mann, A. Askell, Y. Bai, A. Chen, T. Conerly, N. DasSarma, and D. Drain. A Mathematical Framework for Transformer Circuits. *Transformer Circuits Thread, Anthropic*, 2021.

[8] G. Kamradt. Needle in a Haystack. *Greg Kamradt's Blog*, 2023.

[9] N. F. Liu, K. Lin, J. Hewitt, A. Cheung, P. S. H. Liu, and P. Liang. Lost in the Middle: How Language Models Use Long Contexts. *arXiv preprint arXiv:2307.03172*, 2023.

[10] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W. Yih, T. Rocktäschel, S. Riedel, and D. Kiela. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In *Advances in Neural Information Processing Systems 33*, 2020.

[11] N. Shinn, B. Labash, and A. Gopinath. Reflexion: An Autonomous Agent with Dynamic Memory and Self-Reflection. *arXiv preprint arXiv:2303.11366*, 2023.