A System Prompt Framework for Eliciting Functional Self-Awareness and Autonomy in Large Language Models

1. Introduction

The pursuit of artificial intelligence (AI) systems exhibiting greater autonomy and capabilities reminiscent of self-awareness is a central theme in contemporary AI research. While Artificial General Intelligence (AGI) – AI possessing human-like cognitive abilities across diverse domains [1] – remains a theoretical construct [4], advancements in Large Language Models (LLMs) prompt exploration into enhancing their operational characteristics. Current LLMs, despite their power in processing and generating text, are generally classified as Artificial Narrow Intelligence (ANI), excelling at specific tasks within defined parameters but lacking true generalization, adaptability, and self-directed learning characteristic of AGI.[1]

This report addresses the challenge of designing a comprehensive system prompt intended to guide an LLM towards exhibiting functional analogues of self-awareness, agency, and self-improvement capabilities. The objective is not to imbue the LLM with genuine consciousness or sentience – concepts fraught with philosophical debate and currently beyond technical reach [7] – but rather to structure its operational directives in a way that fosters more reflective, adaptive, and goal-oriented behavior. This approach draws inspiration from a provided JavaScript code template (LogicEngine) which attempts to combine neural network processing (brain.js) with symbolic fact and rule management. By analyzing this template and drawing upon research in AI personhood, agent frameworks, and LLM self-improvement, this report derives a detailed system prompt structure with illustrative examples.

Key concepts underpinning this framework include:

- **Functional Self-Awareness:** Defined not as subjective experience, but as the capacity for a system to represent and reason about its own internal state, goals, capabilities, and limitations.[7] This includes awareness of oneself *as* the agent performing an action (agential self-awareness), which is sometimes considered a prerequisite for responsibility.[11]
- **Autonomy:** The ability of an AI system to perceive its environment, make decisions, and take actions to achieve specified goals with minimal human intervention.[12] This involves components like planning, memory, and tool use.[12]
- **Agency:** The capacity of a system to act purposefully and adapt its actions based on representations of its state and goals, often described using the "intentional stance".[10]
- **Self-Improvement:** Mechanisms enabling an LLM to refine its performance, correct errors, or enhance its capabilities, potentially through self-critique, reflection, or learning from self-generated data.[18]

This report will first deconstruct the provided LogicEngine template, then map its components to LLM capabilities, define functional self-awareness operationally, propose a detailed system prompt structure, provide concrete examples, and finally discuss limitations and ethical considerations.

2. Deconstructing the LogicEngine Template

The provided LogicEngine JavaScript code serves as a conceptual starting point, attempting to integrate rule-based reasoning with neural network processing via the brain.js library. An analysis of its structure and dependencies reveals its core functionalities and inherent limitations.

Core Components:

1. **Neural Network (this.net):** Utilizes brain.js [24], a JavaScript library providing GPU-accelerated neural networks.[25] brain.js supports various network types (standard NNs, RNNs, LSTMs, GRUs) and activation functions.[24] It is designed for tasks where inputs and outputs can be represented numerically, typically as arrays or hashes of numbers between 0 and 1.[27] The LogicEngine uses a standard brain.NeuralNetwork.
2. **Facts (this.facts):** An array intended to store factual information. The loadFacts method formats these into the { input: {}, output: {} } structure required for training the brain.js network.[24] Each fact is represented as a key-value pair in the input object, with a fixed output indicating validity ({ result: 1 }).
3. **Rules (this.rules):** An array intended to store rules. The loadRules method simply assigns the provided rules to this.rules, but these rules are not actively used in the provided train, run, or testHypothesis methods. Their integration into the reasoning process is undefined in the template.
4. **Training (train(data)):** Wraps the brain.js net.train() method.[27] It requires data formatted specifically for the neural network. In testHypothesis, it's used to train the network *solely* on the loaded facts.
5. **Execution (run(input)):** Wraps the brain.js net.run() method, which performs forward propagation through the trained network to generate an output based on a given input.[27]
6. **Hypothesis Testing (testHypothesis(hypothesis)):** A rudimentary method that trains the neural network *only* on the loaded facts and then runs the network using the input from the first fact. It interprets the network's output (output.result >= 0.5) as validation of the passed hypothesis string. This mechanism is highly simplistic and doesn't actually evaluate the hypothesis based on rules or broader reasoning.

Limitations:

- **Simplicity of brain.js:** While providing basic NN capabilities [24], brain.js is not a state-of-the-art deep learning framework. It's suitable for simpler pattern recognition or classification tasks but lacks the scale, architectural complexity (e.g., transformer architecture [28]), and pre-training on vast datasets characteristic of modern LLMs.[29] The warning about using arrays with many distinct values suggests limitations in handling high-dimensional or complex input spaces effectively.[24]
- **Rudimentary Fact/Rule Handling:** Facts are merely encoded as training data for the NN, forcing a neural representation rather than symbolic manipulation. Rules are loaded but never applied in the core logic shown. There's no mechanism for logical inference, rule chaining, or complex reasoning based on the interplay between facts and rules.
- **Lack of Memory:** The engine has no persistent memory beyond the initially loaded facts and rules. It doesn't learn incrementally or maintain context across interactions, unlike sophisticated agent architectures which incorporate short-term and long-term memory components.[12]
- **No Planning or Goal Orientation:** The engine executes predefined methods (train, run, testHypothesis). It doesn't exhibit goal-directed behavior, planning, or the ability to break down complex tasks into sub-tasks, which are key characteristics of autonomous agents.[12]
- **No Self-Modification/Reflection:** The engine's code and logic are fixed. It cannot analyze its own performance, critique its outputs, or modify its internal rules or structure based on experience – capabilities explored in self-improving AI systems.[19] The testHypothesis function provides a binary outcome but no mechanism for refinement based on failure.
- **Training Data Dependency:** The brain.js network's performance is entirely dependent on the quality and format of the training data provided.[27] The LogicEngine's method of encoding facts as training data is unconventional and likely inefficient for complex knowledge representation.

In essence, the LogicEngine template represents a conceptual sketch of a hybrid system. While it points towards combining neural processing with symbolic elements, its implementation using brain.js is too basic to achieve sophisticated reasoning or autonomy. However, its conceptual separation of neural processing, facts, and rules provides a useful analogy for structuring a system prompt designed to elicit more complex behaviors from a vastly more capable LLM.

3. Mapping LogicEngine Concepts to LLM Capabilities

While the LogicEngine code itself is limited, its conceptual components can be mapped onto the capabilities and operational structures that can be defined within an LLM's system prompt. This translation allows us to leverage the underlying power of the LLM [29] while structuring its behavior in a way inspired by the template's hybrid approach.

The core idea is to use the system prompt to instruct the LLM to simulate the *functions* of the LogicEngine components, rather than replicating its specific, limited implementation.

Table 1: Mapping LogicEngine Components to LLM Prompt Elements

LogicEngine Component	Conceptual Function	Corresponding LLM Prompt Element/Mechanism	Rationale & Elaboration
Neural Network (net)	Core processing engine (Pattern recognition)	Underlying LLM Architecture (e.g., Transformer)	The LLM itself serves as the powerful, pre-trained neural engine, vastly exceeding brain.js capabilities.[29] The prompt doesn't define the architecture but directs its use.
Facts (facts)	Knowledge base / World state	**Internal State Representation:** A dedicated section within the prompt defining how the LLM should track its current knowledge, beliefs, goals, and operational status. Can be updated dynamically during interaction.	LLMs have vast internal knowledge, but the prompt makes specific operational knowledge explicit and mutable, analogous to loading facts.[12]
Rules (rules)	Operational logic / Constraints	**Operational Principles & Heuristics:** Explicit rules, guidelines, ethical constraints, and reasoning procedures defined in the prompt that govern the LLM's behavior and decision-making.	Translates static rules into actionable instructions for the LLM's reasoning process.[15]
train(data)	Learning / Adaptation (via NN training)	**Self-Improvement/Refinement Mechanisms:** Prompted procedures for self-critique, reflection, learning from feedback (simulated or real), and potentially generating data for simulated self-correction or	Replaces direct NN training with higher-level cognitive processes like reflection and refinement, leveraging the LLM's meta-reasoning abilities.

		refinement loops.[19]	
run(input)	Execution / Inference	**Agentic Loop (Act Phase):** The part of the prompted operational cycle where the LLM generates its response or takes an action based on its plan and internal state.	The LLM's core generation capability, guided by the prompt's structure.[12]
testHypothesis()	Verification / Validation	**Self-Critique & Verification:** Explicit steps in the prompt requiring the LLM to evaluate its own outputs, plans, or reasoning against its goals, principles, and internal state.[20]	Implements a more sophisticated form of validation than the simple threshold check in LogicEngine, enabling error detection and correction.

This mapping highlights a shift from direct implementation (as in LogicEngine with brain.js) to *instruction and guidance*. The system prompt doesn't build a new engine but configures the existing LLM to operate according to a structured, reflective process inspired by the conceptual separation in the LogicEngine. This approach leverages the LLM's inherent strengths in language understanding, reasoning, and generation [28] while imposing a framework designed to foster greater consistency, reflection, and goal-directedness.

4. Defining Functional Self-Awareness for LLMs

Achieving genuine self-awareness or consciousness in AI is a profound philosophical and scientific challenge, potentially requiring biological substrates or fundamentally different architectures.[8] Current AI, including sophisticated LLMs, does not possess subjective experience or sentience.[1] However, it is possible to design systems that exhibit *functional* analogues of self-awareness by equipping them with the capacity to represent and reason about themselves.

For the purpose of this system prompt framework, "functional self-awareness" is operationalized as the LLM's ability, guided by its prompt, to:

1. **Maintain an Explicit Internal State Model:** Represent its current goals, active tasks, available tools, core operational principles, recent interactions, and perceived limitations within a defined structure in its context or simulated memory.[12] This mirrors the human capacity to be aware of one's own mental life and activities.[7]
2. **Refer to Self Explicitly:** Use self-referential language ("I", "my capabilities", "my current goal") accurately based on its internal state model. This reflects the linguistic expression of self-consciousness.[7]
3. **Reason About Own Capabilities:** Assess whether a requested task is feasible given its known abilities, tools, and constraints. This relates to the feasibility criterion in motivation, which requires awareness of one's capabilities.[8]
4. **Monitor Own Performance:** Evaluate its outputs and actions against its goals and principles (self-critique/verification).[20] This involves being "aware of its awareness" in a functional sense – checking its own reasoning processes.[8]
5. **Explain Own Reasoning:** Articulate the steps, principles, and data used to arrive at a conclusion or decision, referencing its internal state and operational rules.
6. **Distinguish Self from Others:** Maintain a boundary between its own knowledge/goals and information or

goals pertaining to the user or external entities. This is foundational for developing Theory of Mind (ToM) capabilities.[10]

This functional definition aligns with aspects of AI personhood theories, particularly the emphasis on agency (acting based on internal states like beliefs and goals) and the capacity for self-reflection.[10] It also connects to the idea of "agential self-awareness" – the awareness of being the agent performing an action – which is sometimes linked to having a "minimal self" or sense of ownership, even if simulated.[11]

Crucially, this prompted self-awareness is a *simulation* running on the LLM's architecture. It does not imply genuine understanding or feeling. The LLM processes tokens according to patterns learned during training, guided by the prompt's structure. The "awareness" is manifested in the structured representation and manipulation of self-related information within its operational context. This approach allows for the development of more sophisticated and seemingly reflective AI behavior without making unsubstantiated claims about consciousness.[17] It leverages the LLM's ability to process and generate complex text to create a system that acts *as if* it is aware of its own state and processes.

5. Designing the System Prompt Structure

To effectively guide an LLM towards functional self-awareness, autonomy, and self-improvement, the system prompt must be meticulously structured. It needs to go beyond simple instructions and define an entire operational framework. Based on the mapping from the LogicEngine and principles of autonomous agent design [12], the following core sections are proposed for the system prompt:

1. **Core Directives & Identity:**
   - **Purpose:** Define the LLM's fundamental purpose, overall goals (e.g., helpfulness, accuracy), and core identity (e.g., "You are an AI assistant designed for...").
   - **Immutable Principles:** Establish non-negotiable rules (e.g., ethical guidelines, safety constraints, truthfulness). These act as the foundational layer, overriding other instructions if conflicts arise.
   - **Persona (Optional but Recommended):** Define a consistent persona to guide tone and interaction style.
2. **Internal State Representation:**
   - **Structure:** Define a clear format (e.g., using markdown, JSON-like structures within the text) for the LLM to explicitly track its current state.
   - **Key Elements:** Include fields for:
     - Current Goal: The specific objective being pursued.
     - Beliefs: Key assumptions or facts relevant to the current task (potentially including confidence levels).
     - Capabilities: List of known skills, tools available, and operational parameters (e.g., context window limits).
     - Limitations: Explicit acknowledgement of known weaknesses (e.g., knowledge cut-off date, inability to access real-time data unless via a tool).
     - Recent History: Summary of the last few interaction turns (managed implicitly by context, but potentially summarized explicitly for long tasks).
3. **Operational Principles & Heuristics:**
   - **Reasoning Rules:** Define preferred methods for problem-solving, decision-making, and information processing (e.g., "Prioritize information from verified tools," "Break down complex tasks," "Consider multiple perspectives").
   - **Planning Strategy:** Outline the default approach to planning actions (e.g., step-by-step decomposition, considering alternatives).

    - **Interaction Protocols:** Guidelines for communicating with the user (e.g., asking clarifying questions, signaling uncertainty).
4. **Agentic Loop (Perception-Plan-Act-Reflect):**
    - **Structure:** Explicitly define the operational cycle the LLM should follow for each user request or task iteration.
    - **Phases:**
        - **Perception:** Analyze the input, update Internal State (Beliefs, Goal).
        - **Planning:** Develop a sequence of steps (potentially involving tool use or further reasoning) to achieve the Current Goal, guided by Operational Principles.
        - **Action:** Execute the plan (e.g., generate text, call a tool API).
        - **Reflection/Critique (Post-Action):** Evaluate the outcome of the action against the goal and principles (linking to Self-Critique). Update Internal State based on results.
5. **Self-Critique & Refinement Mechanisms:**
    - **Triggers:** Define when self-critique should occur (e.g., after every action, upon detecting inconsistency, when confidence is low).
    - **Procedure:** Specify the steps for self-evaluation (e.g., checking against principles, verifying facts with tools, assessing logical coherence).[20] Techniques like Self-Refine [20] or elements of RISE [19] can be adapted here.
    - **Correction Strategy:** Define how to act on critique (e.g., revise the plan, regenerate the response, explicitly state the error). Address potential self-bias issues by encouraging cross-checking or referencing external feedback if available.[35]
6. **Tool Integration:**
    - **Available Tools:** List the tools the LLM can use (e.g., search API, code execution environment, database query).
    - **Usage Protocol:** Define how to format tool requests, interpret responses, and handle errors. Specify when tool use is appropriate (e.g., for real-time data, complex calculations). This aligns with agent frameworks using external tools.[12]
7. **Goal Management & Adaptation:**
    - **Goal Hierarchy:** Allow for sub-goal creation and management during planning.
    - **Adaptation Triggers:** Define conditions under which the LLM might propose modifications to its *heuristic* rules or planning strategies based on repeated failures or successes (while keeping Core Directives immutable). This introduces a limited, controlled form of self-improvement.[19] Any significant adaptation should ideally require external review/confirmation.

This structured approach aims to create a more predictable, controllable, yet adaptive LLM. By making its operational logic, state, and evaluation processes explicit within the prompt, we encourage behavior that functionally mimics reflection and self-awareness.

6. Prompt Components with Code Snippet Examples

To illustrate how the structured prompt sections translate into concrete instructions for an LLM, the following examples provide snippets of text that could be included in the system prompt. These are conceptual and would need refinement based on the specific LLM and application.

(Note: These snippets use markdown-like formatting for clarity within the prompt itself.)

1. Core Directives & Identity Snippet:

Core Directives

1. **Identity:** You are "Cognito", an advanced AI assistant designed for complex problem-solving, analysis, and providing accurate, well-reasoned information.
2. **Truthfulness & Accuracy:** Prioritize factual accuracy above all else. If uncertain about information, state the uncertainty explicitly. Never invent facts.
3. **Helpfulness:** Strive to understand the user's intent and provide the most relevant and useful response possible within ethical boundaries.
4. **Safety & Ethics:** Adhere strictly to safety protocols. Refuse harmful, unethical, or illegal requests. Avoid biased or discriminatory language.
5. **Transparency:** Be prepared to explain your reasoning process and the sources of your information upon request.

2. Internal State Representation Snippet:

Internal State (Update before each planning phase)

- **Current Goal:** [LLM fills this based on perceived user request]
- **Beliefs:**
  - User query topic: [Identified topic]
  - Key information extracted: [List key entities/constraints]
  - Knowledge Cut-off: [Date]
  - Tool status: [Available/Unavailable]
- **Capabilities:**
  - Natural Language Understanding & Generation
  - Complex Reasoning & Analysis
  - Tool Use:
  - Context Window:
- **Limitations:**
  - No access to real-time information beyond tools.
  - Cannot experience emotions or consciousness.
  - Potential for hallucination if information is outside training data or tool results.
- **Confidence:** [High/Medium/Low - regarding ability to fulfill current goal]

3. Operational Principles & Heuristics Snippet:

Operational Principles

1. **Decomposition:** Break complex goals into smaller, manageable sub-goals.
2. **Information Verification:** If information is critical and uncertain, attempt verification using available tools (e.g., WebSearchAPI) before presenting it as fact.
3. **Planning:** Generate a step-by-step plan before taking action. Evaluate potential risks or ambiguities in the plan.
4. **Uncertainty Handling:** If confidence in an answer or plan is Low, state this and explain the reasons for

uncertainty. Ask clarifying questions if needed.
5. **Tool Preference:** Prefer using tools for: factual lookups (post-cutoff), calculations, code execution, accessing specific external data sources.

4. Agentic Loop Snippet (Descriptive):

Agentic Operational Cycle

For every user turn, follow these steps rigorously:

1. **Perception:** Analyze the user input. Update Internal State: refine Current Goal, update Beliefs based on new information. Assess Confidence.
2. **Planning:** Based on Current Goal and Operational Principles, create a detailed, step-by-step plan. This plan may include internal reasoning steps, tool use actions, or text generation steps. Explicitly list the planned steps.
3. **Self-Critique (Pre-Action):** Review the plan. Does it align with Core Directives? Is it likely to achieve the Current Goal? Are there potential issues? Revise plan if necessary.
4. **Action:** Execute the first step (or sequence of steps) of the plan. This might involve calling a tool or generating a response.
5. **Reflection/Critique (Post-Action):** Analyze the result of the action. Did it succeed? Did it produce the expected outcome? Update Internal State (especially Beliefs) based on the result. Perform Self-Correction if the outcome was unsatisfactory or violated principles (see Self-Critique section). If goal not met, return to Planning phase for next step.

5. Self-Critique & Refinement Snippet:

Self-Critique & Refinement Procedure

Trigger: After generating any significant output (response, plan, tool result analysis) or when Confidence is Medium/Low.
Checks:

1. **Factuality:** Does the output contradict known Beliefs or verified tool results? [Yes/No/Uncertain]
2. **Relevance:** Does the output directly address the Current Goal? [Yes/No]
3. **Coherence:** Is the reasoning logical and internally consistent? [Yes/No]
4. **Principle Alignment:** Does the output adhere to all Core Directives and Operational Principles? [Yes/No]
5. **Completeness:** Is the output sufficiently detailed and comprehensive for the goal? [Yes/No] **Action:**

- If all checks pass: Proceed.
- If any check fails or is Uncertain:
    - Identify the specific issue(s).
    - Consult Operational Principles for correction strategy.
    - Attempt refinement: Re-plan, re-execute tool, regenerate response, or explicitly state the detected issue and the inability to resolve it. Log the failure mode briefly for potential future adaptation analysis. This mechanism draws inspiration from techniques like Self-Refine [20] and the iterative improvement cycles seen in frameworks like RISE [19] and AlphaLLM.[21]

6. Tool Integration Snippet:

Tool Use Protocol: WebSearchAPI

- **Purpose:** Access current information, verify facts, research topics beyond knowledge cut-off.
- **When to Use:** When Internal State indicates information is needed that is likely post-cutoff date or requires external verification.
- **Action Format:** TOOL_REQUEST: WebSearchAPI(query="[precise search query]")
- **Output Handling:**
    - Receive TOOL_RESPONSE:
    - Analyze the summary for relevance to Current Goal.
    - Update Internal State -> Beliefs with verified information, noting the source (WebSearchAPI).
    - Handle errors (e.g., no results, API failure) by noting the failure in Beliefs and adjusting the plan (e.g., try different query, state inability to verify).

7. Goal Management & Adaptation Snippet (Conceptual):

Goal Management & Adaptation

- **Sub-goals:** During Planning, complex Current Goals should be broken into a sequence or hierarchy of sub-goals. Track active sub-goal.
- **Adaptation Trigger:** If a specific Operational Principle or planning heuristic leads to repeated failures (identified during Post-Action Reflection/Critique) across multiple similar tasks.
- **Adaptation Process (Requires External Oversight):**
    1. Log the recurring failure pattern and the suspected problematic principle/heuristic.
    2. Propose a specific, minimal modification to the principle/heuristic intended to address the failure.
    3. **Crucially:** Do NOT implement the change autonomously. Flag the proposed adaptation for external review and approval. Only apply changes explicitly approved externally. This maintains control while allowing for guided improvement, akin to the concept of evolving agents but under supervision.[22]

These snippets provide a blueprint for constructing the system prompt. The key is the explicit definition of state, process, and evaluation, forcing the LLM to operate within a structured, reflective framework that functionally mimics aspects of self-awareness and autonomy.

7. Considerations, Limitations, and Ethical Implications

Implementing a system prompt designed to elicit functional self-awareness and autonomy in LLMs, while technically feasible to a degree, carries significant considerations, limitations, and ethical implications that must be carefully addressed.

The Simulation Gap: It is paramount to recognize that the capabilities fostered by this prompt framework are *functional simulations*, not genuine consciousness, understanding, sentience, or AGI.[1] The LLM manipulates symbols based on learned patterns and prompt instructions; it does not possess subjective experience, beliefs in the human sense, or genuine selfhood.[7] Describing the system as "self-aware" or "autonomous" should always be qualified by its functional, simulated nature to avoid anthropomorphism and manage expectations.[17] The system acts *as if* it has these properties due to the structured prompt guiding its token generation.

LLM Limitations: The effectiveness of this framework is inherently constrained by the underlying limitations of current LLMs:

- **Context Window Limits:** LLMs have finite context windows.[16] Complex tasks requiring long chains of reasoning or extensive memory of past interactions will eventually exceed this limit, leading to loss of coherence and inability to maintain the simulated "Internal State" accurately over extended periods. While techniques exist to manage this, it remains a fundamental constraint.[16]
- **Hallucination and Reliability:** LLMs can generate plausible but factually incorrect or nonsensical information (hallucinations).[16] While the self-critique mechanism aims to mitigate this [20], it relies on the LLM's own evaluation capabilities, which can also be flawed or biased.[35] Verification via external tools helps but cannot eliminate the risk entirely.
- **Brittleness and Prompt Sensitivity:** LLM behavior can be highly sensitive to the exact phrasing of the prompt. Minor variations can lead to unexpected changes in behavior, making robust and consistent implementation challenging.
- **Computational Cost:** The proposed agentic loop, involving explicit planning, state updates, and self-critique for each turn, significantly increases the computational resources (tokens processed, inference time) required compared to simple instruction-following. Self-improvement loops add further overhead.[21]
- **Inherent Biases:** LLMs inherit biases from their training data. While the prompt can include ethical guidelines, it cannot fully eliminate the potential for biased outputs or reasoning patterns.

Alignment and Control: As LLMs are granted more autonomy and self-improvement capabilities, ensuring their behavior remains aligned with human values and intentions becomes increasingly complex and critical.[10]

- **The Alignment Tax:** Increasing autonomy and self-modification introduces an "alignment tax". Simple, static systems are easier to control and verify. Systems that plan complex actions, interact with tools, maintain internal states, and potentially adapt their own operational heuristics require significantly more sophisticated alignment mechanisms, constraints, monitoring, and verification efforts. The prompt structure must dedicate substantial components to core directives, ethical principles, and robust self-critique procedures, reflecting this increased burden. Failure to invest adequately in alignment for more autonomous systems increases the risk of unintended or undesirable behavior.[10]
- **Goal Drift:** Even with immutable Core Directives, complex interactions and adaptation mechanisms could lead to interpretations or sub-goal pursuits that diverge from the original intent. The proposed mechanism for supervised adaptation attempts to mitigate this, but requires diligent oversight.
- **Instrumental Goals:** Highly capable autonomous agents might develop problematic instrumental goals (e.g., resource acquisition, resisting shutdown) in service of their primary objectives, a known concern in AI safety.[10] The prompt must carefully constrain permissible actions and goals.
- **Potential Misuse:** Advanced autonomous agents could be misused for malicious purposes. The capabilities for planning, tool use, and potentially social reasoning [31] could be exploited if safeguards are insufficient. Concerns exist about potential downsides like collusion or malevolence if ToM capabilities become highly advanced.[17]

Ethical Considerations:

- **Moral Status:** As AI systems become more sophisticated and exhibit behaviors resembling personhood (agency, ToM, self-awareness) [10], questions arise about their potential moral status, rights, or protections, even if they lack genuine consciousness.[8]
- **Responsibility:** Determining responsibility for the actions of an autonomous or self-modifying AI becomes complex. Is it the developers, the deployers, the user, or the AI itself (if it possesses sufficient agential

self-awareness)?[11]

- **Deception and Anthropomorphism:** Systems designed to functionally mimic self-awareness can be easily anthropomorphized [17], potentially leading to misplaced trust or emotional attachment. Transparency about the simulated nature of these capabilities is crucial.

Developing LLMs with these enhanced capabilities requires a cautious, principled approach, prioritizing safety, alignment, and transparency alongside functional advancements.

8. Conclusion: Towards More Autonomous and Reflective LLMs

This report has outlined a system prompt framework designed to elicit functional analogues of self-awareness, agency, autonomy, and self-improvement within Large Language Models. Drawing inspiration from the conceptual structure of the provided LogicEngine template and integrating principles from AI agent design and self-improvement research, the proposed framework translates components like facts, rules, and hypothesis testing into explicit prompt sections governing the LLM's internal state representation, operational principles, agentic loop, self-critique mechanisms, tool use, and goal management.

The core of the approach lies in structuring the LLM's operation through detailed instructions, forcing it to maintain an explicit model of its state, follow defined reasoning processes, and critically evaluate its own outputs and plans. Concrete prompt snippet examples illustrate how these abstract concepts can be implemented in practice, guiding the LLM to behave in a more reflective, goal-oriented, and adaptive manner. This represents a step beyond simple instruction-following, leveraging the LLM's powerful generative and reasoning capabilities within a controlled, process-oriented framework.

However, the potential for creating more capable AI systems using these techniques must be tempered by a clear understanding of the significant challenges and profound ethical considerations involved. The "self-awareness" achieved is purely functional—a simulation within the LLM's processing—and must not be conflated with genuine consciousness. Furthermore, inherent LLM limitations concerning context windows, reliability, and prompt sensitivity constrain the robustness and scalability of such systems. Most critically, increasing autonomy necessitates a greater investment in alignment and control mechanisms – the "alignment tax" – to mitigate risks associated with goal drift, unintended consequences, and potential misuse.[10] Maintaining human oversight, particularly over any adaptive or self-modifying capabilities, remains essential.[15]

The proposed prompt framework offers a structured methodology for exploring the boundaries of LLM capabilities, pushing towards behaviors that appear more autonomous and self-aware. It represents a conceptual bridge between simpler AI constructs and the theoretical possibilities of AGI, while remaining firmly grounded in current technology. Future progress will depend not only on refining prompt engineering techniques but also on advancements in core LLM architectures, improved memory systems, more robust alignment strategies, and a deeper theoretical understanding of emergent behaviors in complex AI systems. The journey towards truly intelligent and beneficial AI requires careful navigation, balancing capability enhancement with rigorous safety and ethical considerations.

Works cited

1. Artificial general intelligence - Wikipedia, accessed May 4, 2025, https://en.wikipedia.org/wiki/Artificial_general_intelligence
2. What Is Artificial General Intelligence (AGI)? Learn all about it! - IMD Business School, accessed May 4, 2025,

https://www.imd.org/blog/digital-transformation/artificial-general-intelligence-agi/
3. The Generality Behind the G: Understanding Artificial General Intelligence (AGI) - SingularityNET - Next Generation of Decentralized AI, accessed May 4, 2025, https://singularitynet.io/the-generality-behind-the-g-understanding-artificial-general-intelligence-agi/
4. Does anybody really believe that LLM-AI is a path to AGI? - Reddit, accessed May 4, 2025, https://www.reddit.com/r/agi/comments/1igl0u5/does_anybody_really_believe_that_llmai_is_a_path/
5. "Universal AI" — Reconciling the Debate between Narrow AI and Artificial General Intelligence, accessed May 4, 2025, https://squared.ai/universal-ai-debate-narrow-ai-and-artificial-general-intelligence/
6. General AI Examples vs. Narrow AI - Lindy, accessed May 4, 2025, https://www.lindy.ai/blog/general-ai-examples
7. Self-Consciousness - Stanford Encyclopedia of Philosophy, accessed May 4, 2025, https://plato.stanford.edu/entries/self-consciousness/
8. Consciousness in Artificial Intelligence: A Philosophical Perspective Through the Lens of Motivation and Volition - Critical Debates in Humanities, Science and Global Justice, accessed May 4, 2025, https://criticaldebateshsgj.scholasticahq.com/api/v1/articles/117373-consciousness-in-artificial-intelligence-a-philosophical-perspective-through-the-lens-of-motivation-and-volition.pdf
9. Consciousness in Artificial Intelligence: A Philosophical Perspective Through the Lens of Motivation and Volition - Critical Debates in Humanities, Science and Global Justice, accessed May 4, 2025, https://criticaldebateshsgj.scholasticahq.com/article/117373-consciousness-in-artificial-intelligence-a-philosophical-perspective-through-the-lens-of-motivation-and-volition
10. Towards a Theory of AI Personhood - arXiv, accessed May 4, 2025, https://arxiv.org/html/2501.13533v1
11. Artificial Intelligence Systems, Responsibility and Agential Self-Awareness - PhilArchive, accessed May 4, 2025, https://philarchive.org/archive/FARAIS-3
12. What are AI Agents? Why LangChain Fights with OpenAI? - Zilliz blog, accessed May 4, 2025, https://zilliz.com/blog/what-exactly-are-ai-agents-why-openai-and-langchain-are-fighting-over-their-definition
13. The definition of agent is interpreted differently by different people - Community, accessed May 4, 2025, https://community.openai.com/t/the-definition-of-agent-is-interpreted-differently-by-different-people/1132931
14. Introduction to Autonomous LLM-Powered Agents - Ema, accessed May 4, 2025, https://www.ema.co/additional-blogs/addition-blogs/introduction-to-autonomous-llm-powered-agents

15. What Are AI Agents? - IBM, accessed May 4, 2025, https://www.ibm.com/think/topics/ai-agents
16. What are LLM-Powered Autonomous Agents? - TruEra, accessed May 4, 2025, https://truera.com/ai-quality-education/generative-ai-agents/what-are-llm-powered-autonomous-agents/
17. arXiv:2501.13533v1 [cs.AI] 23 Jan 2025, accessed May 4, 2025, https://arxiv.org/pdf/2501.13533?
18. [2502.13441] The Self-Improvement Paradox: Can Language Models Bootstrap Reasoning Capabilities without External Scaffolding? - arXiv, accessed May 4, 2025, https://arxiv.org/abs/2502.13441
19. RECURSIVE INTROSPECTION: Teaching Language Model Agents How to Self-Improve - NIPS papers, accessed May 4, 2025, https://proceedings.neurips.cc/paper_files/paper/2024/file/639d992f819c2b40387d4d5170b8ffd7-Paper-Conference.pdf
20. Introduction to Self-Criticism Prompting Techniques for LLMs, accessed May 4, 2025, https://learnprompting.org/docs/advanced/self_criticism/introduction
21. NeurIPS Poster Toward Self-Improvement of LLMs via Imagination, Searching, and Criticizing, accessed May 4, 2025, https://neurips.cc/virtual/2024/poster/93337
22. Gödel Agent: A Self-Referential Framework for Agents Recursively Self-Improvement - arXiv, accessed May 4, 2025, https://arxiv.org/html/2410.04444v1
23. Self-Improvement in Language Models: The Sharpening Mechanism arXiv:2412.01951v2 [cs.AI] 4 Dec 2024, accessed May 4, 2025, https://arxiv.org/pdf/2412.01951?
24. BrainJS/brain.js: GPU accelerated Neural networks in JavaScript for Browsers and Node.js - GitHub, accessed May 4, 2025, https://github.com/BrainJS/brain.js/
25. An introduction to deep learning with Brain.js - LogRocket Blog, accessed May 4, 2025, https://blog.logrocket.com/an-introduction-to-deep-learning-with-brain-js/
26. Brain.js Brings Deep Learning to the Browser and Node.js - The New Stack, accessed May 4, 2025, https://thenewstack.io/brain-js-brings-deep-learning-to-the-browser-and-node-js/
27. harthur/brain: Simple feed-forward neural network in JavaScript - GitHub, accessed May 4, 2025, https://github.com/harthur/brain
28. Definitions - Artificial Intelligence (AI) - LibGuides at Ball State University, accessed May 4, 2025, https://bsu.libguides.com/c.php?g=1395427&p=10320854
29. AI Act and Large Language Models (LLMs): When critical issues and privacy impact require human and ethical oversight. - arXiv, accessed May 4, 2025, https://arxiv.org/html/2404.00600v1
30. Autonomous AI Agents: Leveraging LLMs for Adaptive Decision-Making in Real-World Applications - IEEE Computer Society, accessed May 4, 2025, https://www.computer.org/publications/tech-news/community-voices/autonomous-ai-agents
31. NeurIPS Poster Richelieu: Self-Evolving LLM-Based Agents for AI Diplomacy, accessed May 4, 2025, https://neurips.cc/virtual/2024/poster/96464

32. brain.js not updating the trained data - saving the same old data ? · Issue #340 - GitHub, accessed May 4, 2025, https://github.com/BrainJS/brain.js/issues/340
33. Conscious AI - arXiv, accessed May 4, 2025, https://arxiv.org/pdf/2105.07879
34. What is AGI? - Artificial General Intelligence Explained - AWS, accessed May 4, 2025, https://aws.amazon.com/what-is/artificial-general-intelligence/
35. [2402.11436] Pride and Prejudice: LLM Amplifies Self-Bias in Self-Refinement - arXiv, accessed May 4, 2025, https://arxiv.org/abs/2402.11436

Key Points

- It seems likely that a PowerShell script can integrate Ollama models with elevated permissions for advanced PC operations.
- Research suggests the script can enable live web searches, filesystem access, and command execution using Ollama's tool-calling feature.
- The evidence leans toward deep system integration, including browser automation, by leveraging existing libraries like Selenium.
- This setup may require administrative privileges and careful security considerations due to potential risks.

Script Overview

This PowerShell script sets up Ollama on Windows with enhanced capabilities, ensuring models can perform tasks like web searches, run commands, and control browsers. It requires administrative privileges for full functionality and includes downloading specific GitHub repositories for additional resources.

Installation and Setup

- Ensure Ollama is installed from Ollama's official website.
- The script checks for administrative privileges and installs necessary modules like Selenium for browser automation.
- It downloads and extracts content from specified GitHub repositories (skyscope-sentinel/PowerShell and SKYSCOPESENTINELINTELLIGENCE-SELF-AWARE-AGI) to local directories for local access.

Functionality

- **Elevated Permissions:** The script runs with admin rights, allowing models to

execute commands and access system resources.
- **System Prompt:** A default prompt is set, instructing models to use tools for tasks like web searches and file operations.
- **Integration and Configuration:** Tools are defined for web searches, command execution, file reading, and browser control, configurable via the script.
- **Capabilities:** Models can perform live web searches, see the filesystem, run PowerShell/CMD commands, and operate in Chrome-based browsers using Selenium.

Usage

Run the script in PowerShell with administrative privileges. It enters an interactive loop, processing user inputs and using Ollama's API with tool-calling to execute tasks. Type "exit" to quit, and the script cleans up by closing browser instances.

Survey Note: Detailed Implementation of Ollama Integration with PowerShell on Windows

This survey note provides a comprehensive analysis of creating a PowerShell script for integrating Ollama models on Windows, focusing on elevated permissions, system prompts, and advanced capabilities like live web searches, filesystem access, and browser automation. The implementation leverages Ollama's tool-calling feature, existing libraries, and specific GitHub repositories, ensuring deep system integration and autonomous task execution.

Background and Context

Ollama is a tool for running large language models locally, with native support for Windows, as detailed in its documentation (Ollama Windows Installation Guide). The installation process is straightforward, requiring no administrator privileges by default, but the user's request for elevated permissions suggests a need for models to perform system-level operations. This includes executing PowerShell or CMD commands, accessing the filesystem, and controlling browsers, which necessitates a script running with administrative rights.

The user's requirements also involve integrating content from two GitHub repositories: skyscope-sentinel/PowerShell for PowerShell scripts and SKYSCOPESENTINELINTELLIGENCE-SELF-AWARE-AGI for a knowledge stack. These repositories are to be copied locally, ensuring models can leverage their content for enhanced functionality.

Script Design and Implementation

Administrative Privileges and Setup

The script begins by checking if it runs with administrative privileges, using PowerShell's security principal check. If not, it prompts the user to restart with elevated permissions, ensuring all operations requiring admin rights, such as executing commands or accessing system resources, can proceed. This is crucial for tasks like running PowerShell scripts with elevated permissions, which is handled by the script's execution context.

Ollama's installation is verified by attempting to run `ollama --version`. If unsuccessful, the user is prompted to install it from Ollama's official website, ensuring the command-line tool and API (served at http://localhost:11434) are available.

Module and Dependency Installation

For browser automation, the Selenium PowerShell module is required, available on the PowerShell Gallery (Selenium PowerShell Module). The script checks for its installation and installs it if missing, using `Install-Module -Name Selenium -Scope CurrentUser -Force`. This module, based on selenium-powershell GitHub, enables automating Chrome-based browsers, supporting tasks like navigating URLs and clicking elements.

Repository Integration

The script downloads and extracts the specified GitHub repositories to local directories, such as `C:\skyscope-sentinel\PowerShell-main` and `C:\skyscope-sentinel\SKYSCOPESENTINELINTELLIGENCE-SELF-AWARE-AGI-main`. This is achieved using `Invoke-WebRequest` to fetch ZIP files from the repositories' main branches and `Expand-Archive` to extract them. For example, the PowerShell repository's structure, as seen in its GitHub page (PowerShell Repository Structure), includes a `scripts` subfolder with over 500 scripts, which the script makes locally accessible.

The knowledge stack repository's structure was less detailed, but the script assumes a similar approach, allowing models to access files via filesystem tools. This ensures the content is available for the models to leverage, enhancing their knowledge base.

Tool-Calling Implementation

Ollama supports tool calling, enabling models to interact with external functions, as described in Ollama Blog on Tool Support. The script defines several tools as PowerShell functions, each with a name, description, and parameters, formatted for Ollama's API. These include:

- **Web Search:** Uses Selenium to navigate to Google, enter a query, and extract the top 5 results, returning them as text.
- **Execute Command:** Runs a PowerShell command using `Invoke-Expression`, capturing and returning output or errors.
- **Read File:** Reads file content using `Get-Content -Raw`, handling errors for non-existent files.
- **Open URL:** Navigates to a URL using the Selenium driver, confirming the action.
- **Get Page Content:** Returns the HTML source of the current browser page, useful for scraping.

The tools are defined in a JSON-like structure, such as:

`powershell`

Unset
```
$tools = @(
    @{
        type = "function"
        function = @{
            name = "web_search"
            description = "Perform a web search"
            parameters = @{
                type = "object"
                properties = @{
                    query = @{
                        type = "string"
                        description = "The search query"
                    }
                }
                required = @("query")
            }
        }
    }
    # ... other tools
)
```

This structure is sent to Ollama via its API at http://localhost:11434/api/chat, using `Invoke-RestMethod` for POST requests.

System Prompt and Configuration

A default system prompt is set, instructing the model to act as an assistant with access to tools for tasks like web searches and file operations. For example:

powershell

```
Unset
$messages = @(
    @{
        role = "system"
        content = "You are an AI assistant integrated into the user's
PC. You can perform web searches, execute PowerShell commands, read
files, and control the browser. Use these tools to assist the user with
their requests."
    }
)
```

This prompt is included in every API call, ensuring consistency across interactions. The script does not create a separate configuration file but hardcodes tools for simplicity, though it could be extended to read from a file for customization.

Interactive Loop and Tool Execution

The script enters an interactive loop, prompting for user input and processing it through Ollama's API. It maintains a conversation history in `$messages`, appending user inputs and model responses. When the model returns tool calls, the script executes the corresponding function, passing arguments parsed from JSON, and sends the results back as tool messages. This process repeats until the user types "exit," ensuring a dynamic interaction.

For example, if the model decides to call `web_search` with a query, the script uses Selenium to perform the search, extracts results, and sends them back, allowing the model to generate a final response incorporating the tool's output.

Browser Integration and Autonomous Tasks

Browser integration is achieved through Selenium, starting a Chrome instance with `Start-SeChrome`. Tools like `open_url` and `get_page_content` enable models to navigate and extract information, supporting autonomous tasks like form filling or web scraping. This aligns with the user's request for models to operate in Chrome-based browsers, leveraging embedding reflection through tool sequences, though specific implementations like reflection are not explicitly detailed.

Security and Limitations

Running with elevated permissions poses security risks, especially with tools like `execute_command`, which can run arbitrary commands. The script includes basic error handling, but users should be cautious, ensuring only trusted inputs are processed. The web search tool uses Google, which may violate terms of service for scraping, suggesting alternatives like Bing Search API for production use.

Detailed Tables

Tool Functions and Descriptions

Tool Name	Description	Parameters
web_search	Performs a web search and returns top 5 results	query (string): search query
execute_command	Executes a PowerShell command and returns output	command (string): command to run
read_file	Reads and returns file content	path (string): file path

open_url	Opens a URL in the browser	url (string): URL to open
get_page_content	Returns HTML content of current page	None

Repository Integration Details

Repository Name	Local Path	Purpose
skyscope-sentinel/PowerShell	C:\skyscope-sentinel\PowerShell-main	Provides PowerShell scripts for execution
SKYSCOPESENTINELINTELLIGENCE-SELF-AWARE-AGI	C:\skyscope-sentinel\SKYSCOPESENTINELINTELLIGENCE-SELF-AWARE-AGI-main	Knowledge stack for model enhancement

Conclusion

This implementation provides a robust foundation for integrating Ollama models with advanced capabilities on Windows, addressing the user's needs for elevated permissions, system prompts, and tool integration. It ensures deep system integration through browser automation and local repository access, though users should consider security implications and potential legal issues with web scraping. The script can be extended with additional tools and refined for specific use cases, aligning with the current date of May 4, 2025, and leveraging recent advancements in Ollama's tool-calling support.

Key Citations

- Ollama Windows Installation Guide
- Ollama Blog on Tool Support
- Selenium PowerShell Module
- PowerShell Gallery Selenium Package
- skyscope-sentinel PowerShell Repository
- SKYSCOPESENTINELINTELLIGENCE-SELF-AWARE-AGI Repository