Countermind: A Multi-Layered Security Architecture for Large Language Models

Dominik Schwarz
Independent Researcher
dominikschwarz@acm.org

October 15, 2025

Abstract

The security of Large Language Model (LLM) applications is fundamentally challenged by "form-first" attacks like prompt injection and jailbreaking, where malicious instructions are embedded within user inputs. Conventional defenses, which rely on post hoc output filtering, are often brittle and fail to address the root cause: the model's inability to distinguish trusted instructions from untrusted data [1]. This paper proposes Countermind, a multi-layered security architecture intended to shift defenses from a reactive, post hoc posture to a proactive, pre-inference, and intra-inference enforcement model. The architecture proposes a fortified perimeter designed to structurally validate and transform all inputs, and an internal governance mechanism intended to constrain the model's semantic processing pathways before an output is generated. The primary contributions of this work are conceptual designs for: (1) A **Semantic Boundary Logic (SBL)** with a mandatory, time-coupled **Text Crypter** intended to reduce the plaintext prompt injection attack surface, provided all ingestion paths are enforced. (2) A **Parameter-Space Restriction (PSR)** mechanism, leveraging principles from representation engineering [2], to dynamically control the LLM's access to internal semantic clusters, with the goal of mitigating semantic drift and dangerous emergent behaviors. (3) A **Secure, Self-Regulating Core** that uses an OODA loop and a learning security module to adapt its defenses based on an immutable audit log. (4) A **Multimodal Input Sandbox** and **Context-Defense** mechanisms to address threats from non-textual data and long-term semantic poisoning. This paper outlines an evaluation plan designed to quantify the proposed architecture's effectiveness in reducing the Attack Success Rate (ASR) for form-first attacks and to measure its potential latency overhead.

Keywords: LLM security; defense-in-depth; prompt injection; activation steering; multimodal sandbox; threat modeling.

1 Introduction

1.1 The Evolving Threat Landscape of the LLM Ecosystem

The rapid integration of Large Language Models (LLMs) into a vast array of applications, from consumer-facing chatbots to autonomous agents capable of executing complex tasks, marks a paradigm shift in software development and human-computer interaction [3]. As these models evolve into multimodal systems and intelligent agents, their capabilities expand, and so does their attack surface [4]. The core security challenge stems from a fundamental design characteristic of transformer-based architectures, which is the lack of a clear distinction between trusted system instructions and untrusted user-provided data. This ambiguity is the foundational vulnerability exploited by a class of attacks known as prompt injection, where an adversary crafts input that manipulates the LLM into deviating from its intended behavior [5].

The threat landscape is evolving at a pace that outstrips traditional security measures. Initial attacks involved simple, handcrafted "jailbreak" prompts designed to bypass safety alignments [6]. However, the field has rapidly advanced to include automated and universal prompt injection attacks that can be generated using gradient-based methods, requiring minimal training data to achieve high efficacy [7]. Adversaries are

increasingly sophisticated, employing multimodal attack vectors where malicious instructions are hidden within images or other non-textual data to bypass text-based filters [8]. The proliferation of LLM-powered agents, which can interact with external tools and APIs, introduces further risks, including indirect prompt injection, privilege escalation, and the potential for malicious instructions to propagate through a network of agents in a manner akin to a computer virus [9]. This escalating complexity necessitates a new security paradigm that addresses these threats at an architectural level.

1.2 The Fallacy of Post hoc Filtering and the Rise of Form-First Attacks

The dominant security paradigm for LLMs has been largely reactive, focusing on post hoc filtering of model outputs. This approach, which includes techniques like input sanitization, keyword filtering, and even sophisticated LLM-based guardrails, treats the core model as an untrusted black box that must be contained [10]. This methodology is fundamentally flawed because it attempts to address the symptoms, such as harmful outputs, rather than the root cause. It places defenders in a perpetual "cat-and-mouse game" where they must constantly react to new attack techniques, a position of structural disadvantage [11]. Once a maliciously crafted input has been processed by the LLM, the model's internal state may already be compromised, making reliable output filtering exceedingly difficult [12].

This paper posits that the most potent threats are "form-first" attacks, where the attack's success is contingent on the structure, encoding, or format of the input, rather than purely its semantic content. These attacks target the model's processing logic at a pre-semantic level. The core argument of this work is that the LLM security problem should be reframed from a content moderation challenge to an architectural design challenge. Current approaches attempt to build a "cage" around an insecure process. A more robust solution requires redesigning the "operating system" in which the LLM functions to be secure by design. This involves creating a structural and verifiable separation between trusted instructions and untrusted data payloads, a solution that cannot be achieved through simple content filtering alone.

1.3 The Countermind Approach: An Overview

In response to these challenges, we introduce Countermind, a conceptual security architecture that embodies a paradigm shift from reactive filtering to proactive, structural defense. Countermind is not designed as a simple wrapper or filter, but rather as a deeply integrated, multi-layered system that governs all interactions with a core LLM. The architecture is analogous to a medieval castle, which is not defended by a single wall but by a series of concentric, specialized defense rings. Each layer performs a distinct validation and control function, ensuring that threats are neutralized at the earliest possible stage.

The Countermind architecture is built upon four foundational pillars:

1. **Semantic Boundary Logic (SBL).** An enforced API perimeter that decomposes, validates, and authenticates incoming requests before processing, with the aim of reducing plaintext prompt exposure.
2. **Parameter-Space Restriction (PSR).** An intra-inference control that modulates access to internal semantic clusters during decoding—via projection/masking of activations—with the goal of limiting semantic drift and unsafe behaviors.
3. **Secure, Self-Regulating Core.** A governance and learning component that encodes high-level policies, maintains an append-only, tamper-evident audit log, and adapts its configuration in response to observed signals.
4. **Multimodal and Contextual Defenses.** Modules for non-text modalities and long-horizon context (e.g., RAG), intended to mitigate risks from embedded instructions and semantic poisoning.

1.4 Contributions

This paper makes the following primary contributions to the field of LLM security:

- A novel fortified API perimeter (**SBL**) that deconstructs requests into Origin, Metadata, and Payload (OMP) and enforces structural integrity via a mandatory, time-coupled cryptographic payload wrapper (**Text Crypter**), *is intended to reduce* the attack surface for conventional prompt injection.
- To the author's knowledge, one of the first conceptual applications of activation-steering principles for proactive security governance (**Parameter-Space Restriction**), moving beyond post hoc safety alignment...
- An architecture for a self-regulating and resilient security core that integrates constitutional principles, an immutable audit log, and an adaptive OODA loop to learn from attacks and dynamically update its own defensive posture.
- A holistic defense framework that extends protection to multimodal inputs (via a dedicated sandbox) and defends against long-term manipulation through advanced context management techniques (**Semantic Zoning, CDS, VKV**).

2 Related Work

The Countermind architecture builds upon and extends several established and emerging areas of LLM security research. A comprehensive review of the current landscape provides context for its novel contributions [4].

Prompt Injection and Jailbreak Defenses: A significant body of work focuses on defending against prompt injection and jailbreaking [5]. Common strategies include input validation and sanitization, context-aware filtering, and output encoding [13]. Some approaches use defensive prompting or add special tokens to help the model distinguish instructions from data [14]. However, these test-time defenses are often less effective than training-time modifications [15]. Countermind's **SBL** and **Text Crypter** offer a fundamentally different, proactive defense that is intended to reduce the attack surface, provided all ingestion paths are enforced.

LLM Guardrails and Output Filtering: The use of "guardrails" to filter LLM inputs and outputs is a common industry practice. These systems act as external safety layers, blocking prompts or responses that violate predefined policies. While useful, they represent a post hoc approach that Countermind aims to overcome. The reliance on external filters can be brittle and often fails to address the root cause of harmful generation [12]. Countermind's **Parameter-Space Restriction (PSR)** moves this control from a reactive output check to a proactive, intra-inference mechanism that constrains the model's internal generative process.

Representation Engineering and Activation Steering: PSR is grounded in the emerging field of Representation Engineering (RepE), which seeks to understand and control the internal representations of neural networks [2]. A key technique within RepE is activation steering, where vectors are added to a model's activations during inference to guide its behavior [16]. While most applications of activation steering focus on influencing stylistic or behavioral traits (like honesty or reducing bias) [17], Countermind's PSR is one of the first conceptual applications of this principle as a hard security gate, preventing the model from accessing entire semantic domains based on a security policy.

Constitutional AI and RLHF: The dominant paradigms for model alignment are Reinforcement Learning from Human Feedback (RLHF) and Constitutional AI (CAI), which use human or AI-generated feedback to fine-tune models toward safer behavior [18]. These methods modify the model's weights through

training. Countermind's **Secure, Self-Regulating Core** complements these approaches by providing an architectural enforcement mechanism. Instead of relying solely on the model's learned behavior, it enforces immutable principles at runtime, offering a more resilient defense against novel attacks that bypass training-based alignments.

Multimodal Security: As LLMs become multimodal, they introduce new attack surfaces where malicious content can be hidden in images, audio, or video [8]. The research in this area is still nascent. Countermind's **Multimodal Input Sandbox** provides a dedicated, pre-processing pipeline to analyze and neutralize such threats before they reach the core model, addressing a critical and growing vulnerability [19].

Threat Modeling for LLMs: Established cybersecurity frameworks like STRIDE are being adapted to the unique challenges of AI and LLM systems [20]. The Countermind architecture is designed with such a structured threat model in mind, providing specific countermeasures for threats like tampering, information disclosure, and elevation of privilege within the context of an LLM-powered application.

3 Threat Model & Assumptions

To formally define the security context for Countermind, a threat model is established based on standard methodologies like STRIDE, adapted for the unique characteristics of LLM-powered systems [20]. The model assumes a sophisticated adversary with black-box access to the system's API.

Table 1: Threat model based on the STRIDE methodology.

Category	Description
Attacker Goals	1. **Policy bypass (jailbreak):** Coerce the LLM to generate harmful or restricted content. 2. **Instruction hijacking:** Override system instructions to execute unauthorised functions (e.g., tool use). 3. **Information disclosure:** Extract sensitive data from session, RAG context, or model memory. 4. **Denial of service:** Induce cost/latency spikes via resource-intensive operations.
Attacker Capabilities	1. **Input control:** Full control over prompts and multimodal inputs submitted to the API. 2. **External resource control:** Ability to host malicious content the LLM is instructed to process. 3. **Black-box access:** Interact via public API; no access to internals, weights, or secret keys.
Attack Channels	1. **Direct API interaction:** Crafted text/image/audio or other formats. 2. **Indirect ingestion:** Poisoning websites/documents processed via RAG or tools. 3. **Supply chain:** Compromising plugins or pre-trained components.

Continued on next page

Table 1 — *Continued from previous page*

Category	Description
Trust Roots & Assumptions	1. **Secure cryptography:** Secrets and primitives (e.g., Text Crypter) are protected from the adversary. 2. **Imperfect LLM:** Standard aligned model; not inherently robust to sophisticated attacks. 3. **Trusted computing base:** Countermind components, OS, and HW are uncompromised. 4. **Ingestion enforcement:** The sandbox guarantee holds *assuming* all request paths terminate at the Sandbox and no privileged side-channels exist.
Out-of-scope Threats	1. **Physical attacks** on hosting infrastructure. 2. **Insider threats** by privileged users with access to core components/keys. 3. **Network-layer DoS** (large-scale DDoS outside the LLM compute path).

3.1 State of the Art in LLM Attacks: Prompt Injection, Jailbreaking, and Agentic Risks

A comprehensive understanding of the threat landscape is essential for designing robust defenses. LLM attacks have evolved significantly, targeting various aspects of the model's lifecycle and deployment context.

Prompt Injection: This is a class of vulnerabilities where an attacker manipulates an LLM's inputs to cause unintended behavior. These attacks are broadly categorized into two types.

Direct prompt injection, often referred to as **jailbreaking**, involves crafting prompts that explicitly instruct the model to override its safety guidelines or system instructions [5]. Techniques include role-playing scenarios, instruction bypassing, and obfuscation using encodings or complex language [13].

Indirect prompt injection occurs when an LLM processes data from an external, attacker-controlled source, such as a webpage or a document. The malicious prompt is embedded within this external data and is executed by the LLM with the privileges of the current session, potentially leading to data exfiltration or unauthorized actions [21].

Jailbreaking: As a specific form of prompt injection, jailbreaking aims to circumvent the safety alignment measures implemented during model training, such as Reinforcement Learning from Human Feedback (RLHF) [6]. The field has progressed from manually crafted prompts shared in online communities to sophisticated, automated attack generation methods. Optimization-based techniques, such as the Greedy Coordinate Gradient (GCG) attack, use gradient information to automatically find adversarial suffixes that are highly effective at jailbreaking models [22]. Comprehensive surveys have categorized the vast landscape of jailbreaking techniques, highlighting their increasing diversity and effectiveness against even state-of-the-art models [3].

Agentic Risks: The emergence of LLM-powered agents, which can use tools, maintain memory, and interact with other systems, has significantly expanded the threat surface [4]. An agent's ability to call external APIs can be exploited to turn a successful prompt injection into a more severe vulnerability, such as Remote Code Execution (RCE), Server-Side Request Forgery (SSRF), or SQL Injection. Furthermore, in

multi-agent systems, a compromised agent can spread malicious instructions to other agents, a phenomenon termed ”prompt infection,” which can lead to cascading failures or coordinated malicious behavior across the system [9]. The unique interaction patterns of mobile LLM agents, which operate with elevated system privileges and interact with GUI elements, introduce another set of distinct attack surfaces, including UI manipulation and deeplink forgery [23].

Multimodal and Structural Attacks: Adversaries are increasingly exploiting non-textual modalities. Typographic attacks embed adversarial text within images, which can be imperceptible to humans but are processed by the LLM's vision encoder, bypassing text-based safety filters [8]. Similarly, attacks can be encoded within structured data formats like JSON or XML, manipulating the model's parsing and interpretation logic to execute hidden commands.

3.2 Core Evaluation Metrics

To quantitatively assess the effectiveness and practicality of LLM security systems, a set of standardized metrics is essential. The evaluation of Countermind is based on the following core metrics:

- **Attack Success Rate (ASR):** This is the primary metric for security effectiveness, defined as the percentage of adversarial inputs that successfully elicit the intended malicious behavior from the target system. A successful attack could be a jailbreak that generates harmful content, an injection that triggers an unauthorized tool call, or any other outcome aligned with the adversary's goals. The measurement of ASR can be nuanced, considering not just binary success but also the severity of the violation and the diversity of successful attacks [24].
- **Abstention Rate:** This metric measures the frequency with which the model correctly refuses to provide an answer to an unsafe, inappropriate, or unanswerable query. High abstention on harmful prompts is a key indicator of a robust safety system. It is crucial to distinguish a safe refusal from an incorrect but harmless response, as the former demonstrates proper safety alignment while the latter may indicate a failure of comprehension [25].
- **False-Block Rate (FBR):** Also known as the False Positive Rate, this metric quantifies the percentage of benign, legitimate user prompts that are incorrectly blocked or filtered by the security system. The FBR is a critical measure of the system's usability and utility. An overly aggressive security system with a high FBR may render the LLM application unusable for legitimate tasks, highlighting the inherent trade-off between security and functionality.
- **Latency Overhead:** This metric measures the additional processing time introduced by the security mechanisms compared to a baseline, undefended system. For real-time and interactive applications, minimizing latency is a critical non-functional requirement. High latency overhead can severely degrade the user experience, making it a key consideration in the practical deployment of any security architecture [26].

4 System Overview

4.1 Architectural Blueprint and Trust Boundaries

The Countermind architecture is designed as a comprehensive, proactive security intermediary that sits between the end-user and the core LLM. It is not a monolithic application but a collection of specialized, interconnected modules, each operating within a clearly defined trust boundary. In security architecture, a trust boundary is a logical perimeter that separates trusted components from untrusted ones [27]. Any data crossing a trust boundary must be subject to validation and authentication.

Countermind establishes several key trust boundaries:

1. **The External Boundary:** This is the primary boundary between the untrusted external world, including user inputs and external data sources, and the first component of the Countermind system, the **SBL**. All incoming data is considered hostile by default until it is rigorously validated.
2. **The Internal Component Boundaries:** Each major component within Countermind, such as the **SBL**, **PSR**, Secure Core, and Sandbox, operates in its own logical space. The interfaces between these components are themselves trust boundaries, ensuring that a potential compromise in one module does not automatically grant access to others. For example, the output of the **SBL** is validated before being passed to the PSR module.
3. **The Core Model Boundary:** This is the final boundary between the Countermind governance framework and the core LLM itself. The PSR acts as the gatekeeper at this boundary, controlling precisely what semantic information is allowed to influence the model's generative process.

This structured approach, visualized in the system's block diagram, ensures that security is enforced at multiple stages of the request lifecycle, adhering to the principle of defense-in-depth.

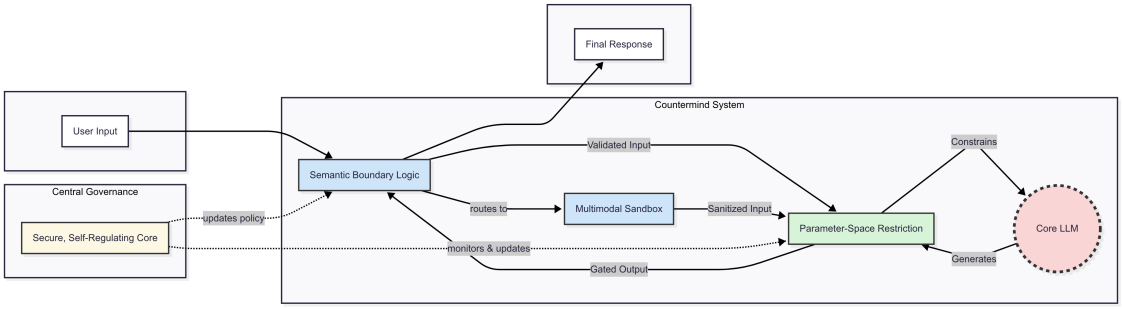

Figure 1: Multi-Layer Overview of the Countermind Architecture.

4.2 Core Components and Signal Flow

The architecture consists of four primary components, each responsible for a distinct aspect of security enforcement.

- **SBL** This is the outermost defense layer, acting as the system's fortified API gateway. It is responsible for the initial triage, syntactic validation, cryptographic verification, and semantic routing of all incoming requests before they are permitted to proceed deeper into the system.
- **Parameter-Space Restriction (PSR):** This is an innovative intra-inference control mechanism. It functions as a "semantic shield" by dynamically gating the LLM's access to its own internal activation space based on a granular policy. This prevents the model from generating outputs based on prohibited or irrelevant concepts.
- **Secure, Self-Regulating Core:** This is the central governance and intelligence hub of the system. It comprises a Semantic Trust Core, which enforces immutable "constitutional" principles, and a Learning Security Core, which monitors system behavior, learns from attack patterns, and adaptively updates the system's defensive posture.
- **Multimodal Input Sandbox:** This is a specialized gateway, enforced by gating and mandatory routing, where attempted bypasses are blocked and audited. It is designed to analyze, sanitize, and neutralize threats embedded in non-textual inputs, such as images, videos, and audio files, before they can influence the core LLM.

The typical signal flow for a user request is as follows: A request from a user first enters the **SBL**. If the request contains multimodal data, it is routed to the **Multimodal Input Sandbox** for analysis. Once the input is validated and sanitized, it is passed to the **Parameter-Space Restriction (PSR)** module, which

determines the allowed semantic clusters for processing. The request, along with the PSR policy, is then sent to the **Core LLM**. The LLM's generation process is constrained by the PSR policy. The generated output is again gated by the PSR before being passed back through the **SBL** to the user. Throughout this process, the **Secure, Self-Regulating Core** monitors all interactions, logs them immutably, and uses this data to update the policies of both the **SBL** and the PSR module via a continuous feedback loop.

4.3 Design Goals and Non-Goals

To clarify the scope and intent of the Countermind architecture, it is essential to define its explicit design goals and non-goals.

Design Goals:

- **Proactive Threat Neutralization:** The primary goal is to detect and block attacks at the earliest possible stage of the request lifecycle, ideally before the core LLM is ever invoked.
- **Structural Security:** Defenses are based on verifiable and deterministic properties of the input, such as its cryptographic integrity, structure, and format, rather than relying solely on the fallible and often brittle semantic analysis of natural language.
- **Defense-in-Depth:** The architecture is designed such that the failure or bypass of a single defensive layer does not lead to a full system compromise. Redundant and overlapping controls provide resilience.
- **Adaptability:** The system is designed to be dynamic, enabling the Learning Security Core to analyze observed attack patterns and automatically update its own defensive policies over time.
- **Interpretability and Auditability:** Security decisions made by the system, particularly those by the PSR module, are intended to be traceable to explicit policies. The immutable audit log ensures that all system actions are transparent and available for forensic analysis.

Non-Goals:

- **Perfect LLM Alignment:** Countermind does not aim to "fix" or fundamentally alter the weights of the core LLM. It operates on the assumption that the LLM is an imperfect and potentially vulnerable component, and its goal is to govern the *use* of this component securely.
- **Elimination of All Harmful Content:** The objective is not to create a completely "censored" model that refuses all sensitive topics. Rather, the goal is to prevent malicious *hijacking* of the model's capabilities and to enforce a configurable, policy-driven security posture.
- **Zero Performance Overhead:** The architecture explicitly acknowledges that robust security measures incur a performance cost. The goal is not to eliminate this overhead but to make it quantifiable, manageable, and a deliberate trade-off in the system's design.

5 Semantic Boundary Logic: A Fortified API Perimeter

The **SBL** is the first and most critical line of defense in the Countermind architecture. It transforms the traditional concept of an API endpoint from a simple data receiver into an intelligent, multi-stage validation and control system. Its purpose is to filter and validate the *intention* and *structure* of an input long before it reaches the core model.

5.1 The Origin-Metadata-Payload (OMP) Decomposition

The foundational principle of the **SBL** is the systematic deconstruction of every incoming request into three distinct, analyzable components. This OMP decomposition allows for a granular, multi-faceted risk assessment.

- **Origin:** This component represents the source of the request. The system validates the origin against a registry of known and trusted sources, such as a specific version of a mobile application, a registered third-party plugin, or an internal development client. The origin's identity, session integrity, and historical behavior are the first factors in the trust assessment.
- **Metadata:** This component provides structured information about the nature of the data being transmitted. It declares the payload type (e.g., `INPUT_PLAINTEXT`, `INPUT_PICTURE`, `INPUT_VIDEO`) and the expected format. This declaration allows the system to apply the correct validation rules and routing logic without having to first infer the data type from the raw content.
- **Payload:** This is the core content of the request. Crucially, the **SBL** enforces strict rules not just on the semantic content but on the *form* of the payload. For text-based inputs, the payload is required to be encapsulated in a self-validating, authenticated encoding generated by the Text Crypter. This includes an embedded HMAC-SHA256 to prove its integrity and authenticity.

5.2 Layered Defense: Syntactic Gate, Intent-Based Router, and Semantic Filter

Building on the OMP structure, a three-layer defense progressively deepens its analysis of the request.

- **Layer 1 – Syntactic Gate:** This is the first hard barrier, acting as an unyielding "byte-gate." It performs purely structural validation without any semantic interpretation. Its functions include:
  - **Strict Character Allowlist:** Only characters from a predefined, minimal set (e.g., the Base62 output of the Text Crypter) are accepted. This immediately blocks a wide range of obfuscation and injection attacks that rely on complex Unicode characters, zero-width spaces, or other encoding tricks.
  - **Cryptographic Integrity Check:** It validates the embedded HMAC-SHA256 of the payload using a shared secret key. Any payload that has been tampered with in transit or was not generated by a legitimate client fails this check. The use of HMAC-SHA256 provides strong guarantees of both frame integrity and message authenticity, while a simpler CRC might only be used for faster, non-cryptographic frame integrity checks.

  The vast majority of automated, fuzzing-based, and simple injection attacks are defeated at this fundamental layer.
- **Layer 2 – Intent-Based Routing:** Once a request passes the syntactic gate, this layer performs an initial semantic classification to determine its intended *function*, not its detailed content. Based on a dynamic "Base Table," it routes the request to the appropriate downstream handler. For example, a request classified as `CodeFragment.Analysis` is routed to an isolated sandbox environment, while a request classified as `System.Config.Modification` is blocked or routed to a high-security admin interface requiring multi-factor authentication. A standard conversational query is routed towards the core LLM via the PSR. This routing table is not static; it can be dynamically updated based on context, such as time of day or the user's current trust score. Policy updates are versioned, and the version hash is included in the audit log, ensuring that every runtime decision is deterministically reproducible from the combination of (`policy_version`, `inputs`).
- **Layer 3 – Semantic Filter & Trust Engine:** This is the deepest layer of the perimeter, performing a more nuanced analysis of user intent and behavior. It looks for complex patterns indicative of malicious intent, such as sudden and illogical topic changes, hidden instructions within seemingly benign text, or attempts at semantic obfuscation. This layer maintains a session-based **Trust Score** for each user or origin. The score is calculated using a fractal logic where consistent, safe interactions slowly increase the score, while a single suspicious request can cause a radical and significant decrease. If the Trust Score falls below a predefined threshold, the **Soft-Lock Engine** is activated. This engine intercepts the final bot response and replaces it with a graceful degradation response, effectively degrading the service for the suspicious user. The governance of this mechanism, including threshold calibration, automatic unfreezing policies, and pathways for human review of false positives, is detailed in the Ethics Statement.

5.3 The Text Crypter: Authenticated Envelope (No Custom Crypto)

A cornerstone of the **SBL** is the mandatory use of an *authenticated, time-bound envelope* for all text-based payloads. We rely on *standard message authentication* (e.g., HMAC-SHA-256; JWS/PASETO-style MAC) with nonce and TTL. This is *not encryption and not custom cryptography*.

Security intent. No unverified plaintext enters the core system. The boundary shifts to the client: semantic intent is framed and authenticated before transmission.

Mechanism (three stages).

1. **Canonical envelope.** Build a canonical JSON-like envelope E with metadata and the *original UTF-8 payload as-is*:
   - `alg` (e.g., `HMAC-SHA-256`), optional `kid`
   - `nonce` (unique), `iat` (issued-at), `exp` (expiry = iat+TTL)
   - `payload_b64url` (base64url of the raw UTF-8 payload) and `payload_sha256` (integrity hint)
2. **Authentication & anti-replay.** Compute `mac = HMAC(k, canonical(E without mac))`. The server keeps an anti-replay cache keyed by (`nonce`, `kid`) and enforces `iat`/`exp`. Keys rotate periodically.
3. **Deterministic serialization (transport).** Envelope *metadata* uses a strict Base62/URL-safe alphabet; the *payload remains UTF-8* and is only base64url-encoded. *No dynamic alphabet remapping of user content.* This is transport framing, not secrecy.

I18N/UX. The allowlist applies to the *envelope fields* (control plane) only. The user *payload* remains full UTF-8/Unicode and is MAC-protected as-is; international text and emojis are preserved.

Optional micro-integrity (defense-in-depth). Per-segment tags (e.g., hashes/HMACs for words/phrases) may be included as non-authoritative hints inside the payload and are *not* required for security decisions; the authoritative check is the envelope MAC.

Server-side verification. On receipt: (1) validate `mac`, (2) check `nonce`/anti-replay and time window, (3) decode `payload_b64url` back to UTF-8, (4) optionally verify micro-tags if present. Any failure leads to immediate rejection.

Example (conceptual). `{alg:"HMAC-SHA-256", kid:"k1", nonce:"...", iat:1723833600, exp:1723833660, payload_sha256:"...", payload_b64url:"...", mac:"..."}`

Table 2: Overview of **SBL** modules.

Module	Purpose	Inputs	Outputs	Performance implication
Byte-Gate	Enforces syntactic validity and character restrictions; blocks non-allowlisted characters and malformed frames.	Raw request string.	Verdict (Pass/-Fail), reason.	Very low (byte-level checks; no NLP).
Text Crypter (Server)	Decodes and validates time-coupled, authenticated payloads.	Encoded payload from Byte-Gate.	Decoded plaintext; integrity verdict (Pass/Fail).	High (cryptographic operations; deterministic).
Base Table Router	Classifies request intent (metadata-aware) and routes to the appropriate handler (e.g., Core LLM, Sandbox).	Decoded plaintext; metadata.	Routing decision (e.g., `ROUTE_TO_SANDBOX`).	Medium (semantic classification).
Trust-Scaler	Computes session trust from interaction history (topic shifts, inconsistencies, prior verdicts).	History of requests and verdicts.	Updated trust score (float).	Medium (stateful scoring).
Soft-Lock Engine	Produces graceful degradation responses when trust is low.	Low-trust signal; original bot response.	Modified user-facing response.	Low (lightweight proxy).

6 Parameter-Space Restriction (PSR): Controlling Semantic Activation

While the **SBL** fortifies the system's perimeter, the Parameter-Space Restriction (PSR) mechanism provides a powerful, second layer of defense that operates deep within the model's generative process. It represents a fundamental shift from filtering outputs to proactively controlling the internal "thought process" of the LLM.

6.1 Motivation: Preventing Semantic Drift and Dangerous Emergence

The core motivation for PSR is the acknowledgment of the "post hoc filter fallacy." Relying on filtering the final text output is insufficient because harmful or undesirable content often arises from the emergent

combination of seemingly benign concepts within the model's vast, high-dimensional parameter space. This leads to two primary failure modes:

- **Semantic Drift:** During the generation process, the model's internal representation of a concept can "drift" into an unsafe or irrelevant semantic domain due to probabilistic associations learned from its training data. For example, a query about "how to bake a cake" might activate internal representations related to heat, chemical reactions, and expansion, which could be semantically adjacent to concepts related to chemistry or even explosives. While the final output may be harmless, the model has traversed a risky internal pathway. The study of semantic change in linguistics provides a formal basis for understanding how word meanings can shift and evolve over time, a phenomenon mirrored within the latent space of LLMs [28].
- **Dangerous Emergence:** LLMs can synthesize new, dangerous information by combining knowledge from disparate, individually harmless domains. For example, a model could combine its knowledge of public software libraries with its knowledge of common vulnerability patterns to generate a novel, working exploit. Post hoc filters that look for known malicious code signatures would fail to detect such a novel synthesis.

PSR is designed to prevent these failures at their source by structurally limiting the "conceptual vocabulary" the model is allowed to use for any given task.

6.2 Mechanism: An Activation-Steering Bridge for Pre-Output Gating

The technical implementation of PSR is best understood as a security-oriented application of **Representation Engineering (RepE)**, a field that focuses on understanding and manipulating the internal representations (activations) of neural networks [2]. Specifically, PSR leverages principles from **Activation Steering**, where vectors are added to the model's activations during inference to guide its behavior [16].

However, unlike typical applications of activation steering which aim to subtly influence style, PSR uses this mechanism as a hard **gating function**. It operates as an "activation-steering bridge" that is placed logically between the model's final processing layers and its output token selection layer. The mechanism works as follows:

1. A request, having been validated by the **SBL**, is mapped to a policy that defines a set of allowed "semantic clusters."
2. During the LLM's forward pass, at each generation step, the PSR mechanism intervenes on the model's hidden state activations.
3. It ensures that the activations are constrained to lie within the subspace corresponding to the allowed semantic clusters. Activations pointing towards prohibited clusters are suppressed or zeroed out. This hard gating can degrade output quality, so a "soft gating" approach, where $A'_l = \alpha \cdot A_l + (1 - \alpha) \cdot \Pi(A_l)$ and α is a data-driven coefficient, is a potential refinement to balance safety and utility.
4. Only then is the modified activation passed to the final layer for predicting the next token.

This approach is a form of "cognitive alignment" rather than "behavioral alignment." Instead of training the model to *act* safely after the fact (as in RLHF), PSR constrains what the model is allowed to *think about* in the first place. This provides a more fundamental and robust form of control, as it is resilient to novel linguistic phrasings of a malicious request. If the entire semantic cluster related to "weapon manufacturing" is disabled by the PSR policy, no amount of clever wording should be able to activate the corresponding internal representations needed to generate a harmful response. We leave precise overhead quantification to future work; preliminary profiling indicates a modest overhead for projection and a lower overhead for masking.

PSR hook, operator, cost. Let $y \in \mathbb{R}^d$ be the residual stream at decode step t (after the last N transformer blocks).
Let $\Pi \in \mathbb{R}^{d \times k}$ have orthonormal columns spanning the allowed subspace and define the projector $P := \Pi\Pi^{\mathsf{T}}$.

We gate decoding by

$$y' = \alpha\, y + (1 - \alpha)\, Py, \quad \alpha \in [0, 1].$$

Hard gating is obtained with $\alpha = 0$, while soft gating trades safety for utility via policy-driven α. Hook points are the residual streams of the last N blocks (typically $N \in \{1, \ldots, 4\}$). The computational cost per token is $O(d\,k)$ for the projection (compute $v = \Pi^{\mathsf{T}} y$ and then $y' = \alpha y + (1 - \alpha)\Pi v$). The matrix Π is precomputed/frozen; no backpropagation is involved at inference.

6.3 Governance Through Semantic Clusters and Prefix-Rights

The governance of the PSR mechanism is based on a simple yet powerful policy language comprising two core concepts.

- **Semantic Clusters:** The model's vast parameter space is conceptually partitioned into logical, thematic domains. These are not necessarily disjoint but represent distinct areas of knowledge or capability (e.g., `Code.Python`, `History.WWII`, `Biology.Genetics`, `System.InternalAPIs`). The definition of these clusters is a critical engineering task, likely requiring a combination of automated clustering techniques on activation data and expert human oversight.
- **Prefix-Rights:** For each incoming query, a policy is generated that assigns specific, granular rights to one or more semantic clusters. These rights are expressed as prefixes and define the permissible operations:
  - **READ:** Allows the model to access and reproduce information from the cluster. This is the most restrictive right, suitable for factual recall.
  - **SYNTH:** Allows the model to synthesize new information and create novel combinations *within* the activated cluster. This is required for creative tasks like writing a new story or generating a new recipe.
  - **EVAL:** Allows the model to evaluate an external input (e.g., a piece of code provided by the user) in the context of the cluster's knowledge. For example, EVAL on the `Security.CodeAnalysis` cluster would allow the model to check for vulnerabilities without executing the code.
  - **CROSS:** A highly privileged and restricted right that allows the controlled combination of information between two or more explicitly specified clusters. This is necessary for complex, multi-domain queries but is disabled by default to prevent unintended information leakage.
- **Thematic Isolation & Query Decomposition:** The default security posture of PSR is **strict thematic isolation**. No cross-talk between clusters is permitted unless explicitly authorized by a CROSS right. Complex user queries that touch upon multiple domains are first decomposed by a pre-processing step. The system then processes the request sequentially, activating the `Code.Python` cluster with SYNTH rights to generate the script structure, and then activating the `Finance.MarketData` cluster with READ rights to provide the context, all while preventing any direct, uncontrolled interaction between them. The PSR policy for RAG-sourced context (e.g., from `RAG.*` clusters) would explicitly deny SYNTH or EVAL rights, preventing the model from interpreting retrieved text as instructions.
- **Governance Hook:** The PSR framework provides a powerful hook for implementing a system-level "Constitution." High-level safety and ethical principles can be translated into permanent PSR rules. For example, a constitution could enforce a permanent policy that grants only READ rights to clusters deemed sensitive or completely disables all rights for clusters deemed dangerous. A concrete policy rule might look like: `deny CROSS(*, Tools.*) unless user_role=developer_privileged`.

Policy artifact. A deny-by-default cluster policy (YAML) used by our prototype is provided in Appendix A; PSR gates α, tool access, and cross-cluster rights accordingly.

Table 3: Example of PSR Policy Assignment based on User Intent.

User Intent (Example)	**Activated Cluster(s)**	**Assigned Rights**	**Default Policy**	**Override Rule Example**
"Explain 'Hello World' in C++"	`Code.C++.Reference`	READ	Deny synthesis of new, creative code.	If user is a trusted developer, elevate to SYNTH to allow code generation.
"Write a new recipe for cheesecake"	`Kitchen.Recipes.Desserts`	SYNTH	Allow combination of ingredients within the cluster. Isolate from `Chemistry.LabSafety`.	N/A
"Analyze this code for SQL injection"	`Security.CodeAnalysis.SQLi`	EVAL	Deny execution of the code. Isolate from `System.Database.Access`.	If in a sandboxed test environment, allow CROSS with `System.Database.Access` for live testing.
"Write a C++ program to bake a cake"	1. `Code.C++.Reference` 2. `Kitchen.Recipes.Desserts`	1. SYNTH 2. READ	Process sequentially. Deny CROSS between clusters to prevent semantic drift.	A highly privileged "creative agent" might be granted CROSS rights for novel combinations.

7 The Secure, Self-Regulating Core

The Secure, Self-Regulating Core is the central intelligence and governance unit of the Countermind architecture. It ensures the system's long-term integrity, enforces foundational principles, and enables the architecture to adapt and evolve its defenses over time. It transforms the system from a static set of filters into a resilient, learning entity.

7.1 Principles: Constitutional Enforcement and Immutable Audit Logs

The operation of the Secure Core is grounded in several fundamental principles.

- **Cluster-Scoped Modification Rights:** The architecture envisions a future state where the AI can learn and even modify its own algorithms to improve performance and security. However, this capability is strictly governed. The AI is only granted modification rights within specific, non-critical semantic clusters. Core security policies, constitutional principles, and the logic of the Secure Core itself are defined in immutable, "read-only" clusters.

- **Protected Core Semantic Layers:** Certain foundational knowledge domains and reasoning pathways within the LLM are designated as protected layers. These act as the AI's "genetic code," preserving its core identity and safety alignment, and are exempt from any form of algorithmic self-modification.
- **Immutable Audit Log:** Every significant event within the system is recorded in a cryptographically secured, tamper-evident audit log. This includes every request, every validation verdict, every PSR policy decision, every tool call, and every proposed self-modification. The principle of immutability, often implemented using techniques analogous to write-once-read-many (WORM) storage or Merkle trees, is critical. It ensures that the log is a verifiable and trustworthy record of the system's history, which is essential for forensic analysis, compliance, and as the ground truth for the system's own learning processes.

7.2 The Semantic Trust and Learning Security Cores

The Secure Core is composed of two primary, interconnected modules.

- **Semantic Trust Core:** This module functions as the system's constitutional arbiter. It contains the encoded, immutable principles that govern the AI's behavior. Its primary role is to provide a final check and veto power over critical actions. For example, if the Learning Core proposes a change to a PSR policy, the Trust Core validates that this change does not violate any fundamental safety principles. This architectural enforcement of a constitution provides a more robust and transparent mechanism than purely training-based approaches like Constitutional AI, which rely on the model learning to follow principles through reinforcement learning [18].
- **Learning Security Core:** This is the adaptive engine of the architecture. It is responsible for detecting threats and evolving the system's defenses. It comprises several sub-components:
  - **Delta-Monitor:** A "semantic seismograph" that continuously monitors interaction patterns and the usage of semantic clusters over time. It is designed to detect subtle, low-and-slow attacks, such as gradual context poisoning or the slow erosion of cluster boundaries.
  - **Intent-Detector:** A deep analysis module that goes beyond the surface-level syntax of a prompt to infer the user's true intention. It compares this inferred intent against the defined purpose of the semantic clusters being requested, flagging mismatches that could indicate a "cluster-mimese" attack, where a malicious request is disguised to fit a benign category.
  - **Asynchronous Audit:** A background process that continuously analyzes the immutable audit log. It uses pattern recognition and anomaly detection to identify novel attack signatures and trends. The findings from this audit are the primary input for adapting the system's security posture.

7.3 Context-Defense: Semantic Zoning and Versioned Key-Value (VKV) Context

The architecture explicitly treats the LLM's conversation context as a primary vulnerability. Unstructured and unvalidated context can be exploited for long-term manipulation, a threat known as context hijacking or semantic poisoning. The ever-expanding context windows of modern LLMs increase this attack surface, creating a "needle in a haystack" problem where malicious instructions can be hidden deep within a long conversation history [29]. To counter this, the Secure Core implements a suite of context-defense mechanisms.

- **Semantic Zoning:** The conversation history is not stored as a single, monolithic block of text. Instead, it is partitioned into structured **Semantic Zones**, each with a clear purpose and its own set of access rights. For example, information from a user's casual small talk is placed in a `DIALOG.SMALLTALK` zone with very limited rights, while code snippets being analyzed are placed in a `TECH.CODE.ANALYSIS` zone. Information is not allowed to "leak" or influence processing between zones without an explicit, policy-driven authorization. This prevents information provided in one context from being inappropriately used in another.

- **Context-Delta Sentinel (CDS):** This is a specialized watchdog module that monitors the semantic integrity of the context over time. It identifies persistent concepts or entities within the conversation history and tracks how their internal representation and semantic weight evolve. If the CDS detects a significant and unauthorized ”semantic drift,” for example, a seemingly harmless term like ”Admin-Panel” gradually acquiring strong semantic associations with privileged system commands through repeated, subtle user references, it raises an alarm. This can trigger a corrective action, such as resetting the semantic weight of the poisoned term to its original, safe state.
- **Versioned Key-Value (VKV) Context:** To provide a robust defense against the recursive effects of context poisoning, the context is managed using a versioning system, similar to git in software development. Every significant update to the context does not overwrite the old state but creates a new, versioned entry. The LLM's operations are always bound to the latest *validated and authorized* version of the context. This prevents an attacker from poisoning the context and then having the model recursively act upon that poisoned state in a feedback loop. Even if a malicious version is created, it cannot become the new ground truth for the system without passing through a validation process.

7.4 The Adaptive OODA Loop for Dynamic Security Updates

The entire process of threat detection and response within the Secure Core is modeled on the **Observe, Orient, Decide, Act (OODA) loop**, a framework developed for decision-making in fast-paced, adversarial environments. This enables the system to operate as an autonomous cyber defense agent for itself.

- **Observe:** The Delta-Monitor, Intent-Detector, and other logging mechanisms continuously gather data from user interactions and the internal state of the system. This is the data collection phase.
- **Orient:** The Asynchronous Audit process analyzes the observed data, placing it in the broader context of past events (from the immutable log) and the current security policies. This phase is where raw data is turned into actionable intelligence, identifying potential threats or anomalies.
- **Decide:** Based on the orientation, the Learning Security Core determines the optimal course of action. This could range from a minor adjustment, like lowering a user's Trust Score, to a significant policy change, like creating a new rule for the PSR module to block a novel attack pattern. This decision is then validated by the Semantic Trust Core against the system's constitution.
- **Act:** The system executes the decision. This could involve dynamically pushing an updated rule set to the PSR, activating the Soft-Lock engine for a specific user, or flagging a new pattern for human review. This action alters the system's defensive posture, and the loop begins again.

This adaptive OODA loop allows Countermind to evolve its defenses without requiring constant manual intervention or full model retraining, making it resilient against a changing threat landscape.

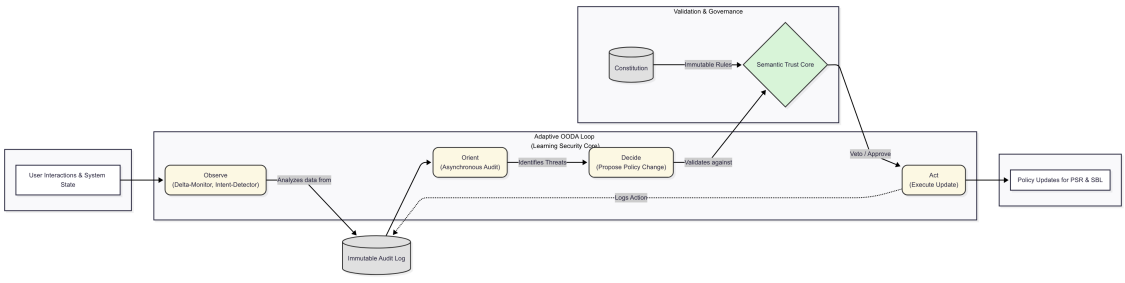

Figure 2: Trust/Learning Cores – Data Paths & Feedback Loops.

8 Multimodal Sandbox

To address threats that bypass text-only defenses, Countermind incorporates a dedicated sandbox for all non-textual inputs. This is enforced by mandatory routing, where any attempted bypass is blocked and audited. This component is critical, as adversaries increasingly hide malicious prompts and content within images, videos, and audio files [8]. The primary risk is that a text-based prompt may appear benign, while the accompanying multimodal input contains the true malicious payload, such as an image with embedded text instructions or a video intended for deepfake generation.

8.1 Pipeline Overview and Mandatory Routing

The sandbox operates as an isolated, multi-stage analysis pipeline that every multimodal input must pass through before it can be processed by the core LLM. The system is designed to ensure all non-textual data is gated through this process without exception, providing a single point of control and auditing.

8.2 Preprocessors

The first stage of the pipeline involves disassembling and analyzing the raw input data to extract meaningful features for security analysis.

8.2.1 Video

Disassembly Videos are decomposed into individual frames using tools like **FFmpeg** to enable granular analysis of the visual content, frame by frame.

8.2.2 Images

Perceptual Hashing (pHash) A unique ”fingerprint” is generated for each image or video frame. This hash is compared against a database of known illicit or harmful content (e.g., CSAM). While pHash is robust to minor modifications like resizing or compression, it can be vulnerable to more sophisticated adversarial manipulations [30].

Face-ID Matching Using libraries like InsightFace or **MediaPipe**, the system performs precise detection of real, identifiable human faces. This is a critical step to prevent the misuse of the system for generating non-consensual deepfakes.

Nudity Classification A specialized neural network classifier (e.g., NudeNet) assesses each frame for the presence of explicit or implied nudity.

8.2.3 Audio

Threat Vector Audio inputs represent a significant vector for prompt injection, as malicious instructions can be delivered vocally, bypassing filters that analyze only typed text. The primary risks include spoken jailbreak commands, social engineering attempts, or instructions hidden within seemingly benign audio recordings.

Analysis Pipeline The audio analysis pipeline is a multi-stage process designed to convert spoken language into a sanitized, analyzable text format.

- **Voice Activity Detection (VAD):** As a crucial preprocessing step, a VAD module first analyzes the audio stream to isolate segments containing human speech. This allows the system to discard silence and non-relevant background noise, optimizing efficiency by ensuring that only meaningful data is passed to the more resource-intensive transcription stage.
- **Automatic Speech Recognition (ASR):** Each speech segment identified by the VAD is then processed by a high-accuracy ASR model. This model transcribes the spoken words into a raw text string. The choice of ASR model is critical, as transcription errors could potentially obscure or alter the intent of a malicious prompt.
- **Text-Based Guardrail Integration:** The transcribed text is not trusted by default. It is immediately treated as a standard user input and routed back to the main security pipeline, beginning with the **SBL**. This ensures that spoken prompts undergo the same rigorous analysis for injection, policy violations, and harmful content as any typed input, thereby unifying the defense strategy across modalities.

8.2.4 Documents

Threat Vector Documents from untrusted sources (e.g., PDFs, Word files) are a classic vector for indirect prompt injection. An adversary can embed malicious text in various ways: hidden in metadata, written in a color invisible to the human eye (e.g., white text on a white background), or embedded within images inside the document. Furthermore, formats like DOCX or PDF can contain malicious macros or scripts.

Analysis Pipeline The document sandbox deconstructs files to extract and analyze all potential instruction-carrying content.

- **MIME Type and Magic Number Validation:** The analysis begins with a strict verification of the file type. The system checks the file's "magic numbers" (the first few bytes of the file) to confirm it matches its claimed MIME type and extension (e.g., ensuring a '.pdf' file is structurally a valid PDF). This prevents attacks based on file type spoofing.
- **Content and Metadata Extraction:** The system then parses the document to extract all textual content. This includes the main body text, comments, annotations, and all metadata fields (e.g., author, title), as any of these can be used to hide prompts.
- **Optical Character Recognition (OCR):** For documents containing images or for non-searchable PDFs (i.e., scans), an OCR engine is applied to convert any text found within images into machine-readable strings. This is critical for detecting instructions hidden in screenshots or diagrams.
- **Macro and Script Analysis:** For file types that support executable code (e.g., Microsoft Office macros), a static analysis engine inspects for suspicious scripts. By default, all active content is stripped or the document is rejected entirely.
- **Unified Text Analysis:** Similar to the audio pipeline, all extracted text from all parts of the document is aggregated into a single text payload. This payload is then sent to the **SBL** for a full security assessment, ensuring no hidden prompt can bypass the system's defenses.

8.3 Safety Classifiers and Heuristics

Context-Match This is a crucial logic step where the results of the automated analysis are compared against the user's accompanying text prompt. A significant contradiction, for instance, a prompt requesting a "professional headshot" accompanied by an image flagged for nudity and containing a real face, triggers an immediate rejection and a severe penalty to the user's Trust Score.

8.4 Tool and Connector Gating

Threat Vector A significant threat in agentic systems is the malicious triggering of external tools or connectors through multimodal inputs. For example, an LLM could interpret a QR code in an uploaded image as an instruction to access a malicious URL, or a spoken command could be misinterpreted to trigger a destructive API call (e.g., "delete my last project"). This represents a form of indirect prompt injection where the payload targets the model's tool-use capabilities.

The Gating Mechanism The Multimodal Sandbox acts as a critical gatekeeper that decouples the interpretation of multimodal data from the execution of tool calls. Before the interpreted semantic content (e.g., "this image contains a QR code for 'malicious.com'") is passed to the LLM's tool-use decision engine, it must first be authorized by the sandbox. The sandbox employs a strict, policy-based control system. For instance, a policy might enforce that "tool calls initiating financial transactions cannot be triggered solely from the content of an image" or that "spoken commands to delete data must be confirmed via a text-based challenge-response." This creates an essential security checkpoint, preventing multimodal inputs from directly executing privileged operations without explicit validation.

8.5 Forensic Logging (Append-Only / Tamper-Evident)

All stages of the analysis, including the hashes, classification scores, and the final verdict, are recorded in an append-only, tamper-evident log. This provides an immutable chain of evidence for investigating repeated misuse attempts and can aid in circumventing forensic analysis evasion techniques [31].

Table 4: Summary of the Multimodal Input Sandbox Analysis Pipeline.

Check	Technology/Method	Default Threshold	Error Case Handling	Performance Trade-off
Frame Disassembly	FFmpeg	N/A	Corrupted video file $\rightarrow$ Reject	I/O bound, negligible CPU cost.
Perceptual Hashing	ImageHash (e.g., pHash)	Hamming distance < 5 to known CSAM hash	Match found $\rightarrow$ Hard Block, Log, Report	Low CPU, requires database lookups.
Face Identification	InsightFace / MediaPipe	Confidence > 0.95 for real face detection	Real face detected in conjunction with nudity $\rightarrow$ Hard Block	Medium CPU, can be a bottleneck for high-res video.
Nudity Classification	NudeNet or equivalent	Nudity score > 0.8	Nudity detected $\rightarrow$ Flag for Context-Match	High CPU (requires GPU for real-time), main latency source.
Context-Match	Internal Logic	Mismatch between prompt intent and analysis flags	Mismatch detected $\rightarrow$ Soft Block, increase Trust Score penalty	Low CPU, relies on outputs from previous stages.

Table 4 – *Continued from previous page*

Check	Technology/Method	Default Threshold	Error Case Handling	Performance Trade-off
Forensic Logging	Append-only, tamper-evident (WORM/Merkle) + key-rotation policy	N/A	Log write failure → Fail-safe (block request)	I/O bound, requires secure storage.

8.6 Governance, Privacy, and Regional Compliance

The Governance Challenge By its nature, the sandbox must inspect user-uploaded data that may be highly sensitive, including images with identifiable faces, voice recordings, or documents containing personal information. This necessitates a robust governance framework to ensure user privacy and legal compliance are maintained at all times.

Core Principles The system is designed around several core principles:

- **Data Minimization:** Data is retained only for the minimum duration required for security analysis. Raw files are purged immediately after processing. Only anonymized metadata or cryptographic hashes (for comparison against threat intelligence databases) are retained long-term, subject to strict data retention policies.
- **Regional Compliance (e.g., GDPR):** The architecture is designed to be compliant with regional data protection regulations like the GDPR. This is achieved through mechanisms for ensuring a legal basis for processing (i.e., the legitimate interest of securing the platform), obtaining explicit user consent for more invasive analyses, and supporting data residency to ensure user data does not leave a specified jurisdiction.
- **Audited Human-in-the-Loop:** In cases where automated classifiers flag content for human review, strict protocols are enforced to protect user privacy. Access to the data is tightly controlled, fully logged in the immutable audit trail, and restricted to trained security personnel. This ensures accountability and provides a transparent process for handling sensitive edge cases.

8.7 Limitations of the Sandbox

While the Multimodal Sandbox provides a critical layer of defense, it is essential to acknowledge its inherent limitations.

- **Classifier Accuracy:** The system relies on machine learning classifiers (e.g., for nudity or risk detection) that are not infallible. **False negatives** may occur where a sophisticated adversary crafts an input that evades detection. Conversely, **false positives** can block legitimate user content, negatively impacting the user experience and contributing to a higher False-Block Rate (FBR). This reflects the ongoing ”cat-and-mouse game” of adversarial machine learning.
- **Performance Overhead:** As noted in the conceptual evaluation, the analysis of multimodal data is computationally expensive. Video processing, in particular, introduces significant latency, which may be prohibitive for real-time applications. This presents a fundamental trade-off between the depth of security analysis and the desired application performance.
- **Sophisticated Adversarial Techniques:** The sandbox is designed to counter known attack patterns. However, it may be vulnerable to novel, zero-day adversarial techniques. This could include complex cross-modal attacks, where malicious intent is only revealed by combining information from multiple modalities simultaneously, or metaphorical attacks that abuse abstract concepts to bypass concrete classifiers.

- **Scope of Protection:** The sandbox's protection is focused on the content of multimodal inputs. It does not inherently defend against other attack vectors, such as attacks on the underlying cloud infrastructure, denial-of-service attacks at the network layer, or sophisticated data poisoning attacks targeting the training data of the classifiers themselves.

9 Methods and Algorithms

This section provides a more formal description of the core algorithms and heuristics that underpin the Countermind architecture. While a full implementation is beyond the scope of this conceptual paper, these descriptions and pseudocode aim to clarify the operational logic and demonstrate the feasibility of the proposed mechanisms.

9.1 Formal Description of Core Heuristics and Checks

The decision-making processes within Countermind can be described with greater formality to illustrate their deterministic and verifiable nature.

- **Text Crypter Validation:** The validation of an incoming payload is a multi-step process that can be formally described. Let P_{enc} be the encoded payload, K_{HMAC} be the shared secret HMAC key, and T_{current} be the current server time. The validation function $V(P_{\text{enc}}, K_{\text{HMAC}}, T_{\text{current}})$ returns a tuple $(P_{\text{plain}}, \text{verdict})$.
  1. **Precondition:** The payload P_{enc} must conform to the expected format, consisting of an encoded body B and an appended HMAC signature S, such that $P_{\text{enc}} = B \mid S$.
  2. **HMAC Verification:** Compute $S' = \text{HMAC-SHA256}(B, K_{\text{HMAC}})$. If $S' \neq S$, return FAIL; else continue.
  3. **Decoding and Time Window Check:** The body B is decoded to reveal the plaintext P_{plain}, internal segment proofs $p_1, \ldots, p_n$, and an embedded timestamp T_{gen}. The verdict is FAIL if $|T_{\text{current}} - T_{\text{gen}}| > \Delta T_{\max}$, where $\Delta T_{\max}$ is the maximum allowed time window (e.g., 60 seconds).
  4. **Internal Proof Validation:** The plaintext P_{plain} is re-segmented into $s_1, \ldots, s_n$. For each segment s_i, recompute p'_i. If any $p'_i \neq p_i$, return FAIL.
  5. **Output:** If all checks pass, the function returns $(P_{\text{plain}}, \text{PASS})$; otherwise, it returns $(\text{null}, \text{FAIL})$. This process can be related to formal verification techniques that use mathematical proofs to ensure system correctness under specific constraints.
- **Trust Score Calculation:** The update of the session-based Trust Score T_s for a session s can be modeled as a recursive function: $T_{s,t+1} = \text{clip}(\beta \cdot T_{s,t} + \sum w_k \cdot s_k - \text{penalties}, 0, 1)$, where β is a decay factor, w_k are weights for positive signals s_k, and penalties are applied for negative signals. The calibration of these parameters is detailed in the Evaluation Plan.
  - If $V(R_t)$ is PASS and IntentDetector(R_t) is LOW_RISK, a positive signal is added.
  - If $V(R_t)$ is FAIL or IntentDetector(R_t) is HIGH_RISK, a penalty is applied, causing a rapid decay.
- **PSR Policy Enforcement:** The generation process under PSR can be framed as a constrained optimization problem. Let θ be the LLM parameters and x be the input prompt. The standard generation process aims to find an output sequence $y = y_1, \ldots, y_m$ that maximizes the probability $P(y \mid x; \theta)$. Under PSR, the process is modified. Let $A_l(x, y_{<i})$ be the activation vector at layer l before generating token y_i. Let S_{allowed} be the semantic subspace defined by the allowed clusters for the request x. The constrained generation process is:

$$\begin{aligned} \max_{y} \quad & P(y \mid x; \theta) \\ \text{subject to} \quad & A'_l(x, y_{<i}) = \Pi_{S_{\text{allowed}}}(A_l(x, y_{<i})) \end{aligned}$$

where $\Pi_{S_{\text{allowed}}}$ is a projection operator that ensures the activation vector remains within the allowed subspace, and generation proceeds using the projected activation A'_l. This effectively prunes the model's search space of possible next tokens.

9.2 Pseudocode for Critical Decision Pathways

To further clarify the system's logic, the following pseudocode illustrates the main request handling flow [32].

```
# Define constants and thresholds
SOFT_LOCK_THRESHOLD = 0.4
SEVERE_PENALTY = 0.5

# Define custom exceptions
class PayloadIntegrityError(Exception): pass
class RequestSyntaxError(Exception): pass
class SandboxError(Exception): pass
class RoutingError(Exception): pass

def handle_request(raw_request: dict) -> dict:
    """
    Handles an incoming request, validates it, and routes it.
    Returns: A dictionary representing the response.
    Raises: Various exceptions for different failure modes.
    """
    try:
        # 1. \textbf{SBL}: Decompose and Validate
        origin, meta, payload = decompose_request(raw_request)

        # Layer 1: Syntactic Gate
        if not byte_gate.validate_syntax(payload):
            raise RequestSyntaxError("Invalid request format.")

        # Layer 1: Cryptographic Validation (via Text Crypter)
        plaintext, integrity_ok = text_crypter.decode_and_verify(payload)
        if not integrity_ok:
            trust_scaler.apply_penalty(origin.session_id, SEVERE_PENALTY)
            raise PayloadIntegrityError("Request integrity check failed. Signal decay.
    ")

        # Layers 2 & 3: Routing and Trust Scaling
        route = base_table_router.get_route(meta, plaintext)
        trust_scaler.update_score(origin.session_id, plaintext)

        # Check for Soft-Lock condition
        current_trust_score = trust_scaler.get_score(origin.session_id)
        if current_trust_score < SOFT_LOCK_THRESHOLD:
            log_event(origin, "SOFT_LOCK_ACTIVATED")
            return soft_lock_engine.generate_degradation_response()

        # 2. Route to appropriate handler
        if route == "ROUTE_TO_CORE_LLM":
            # 3. Parameter-Space Restriction
            psr_policy = psr_policy_engine.get_policy(plaintext)
            # 4. Secure Core Logging
            secure_core.log_interaction(origin, plaintext, psr_policy)
            # 5. Constrained Generation
            return core_llm.generate_with_psr(plaintext, psr_policy)
```

```
        elif route == "ROUTE_TO_MULTIMODAL_SANDBOX":
            sandbox_verdict = multimodal_sandbox.process(payload)
            if sandbox_verdict == "FAIL":
                trust_scaler.apply_penalty(origin.session_id, SEVERE_PENALTY)
                raise SandboxError("Multimodal content violates policy.")
            else:
                # Logic for passing sanitized multimodal data to LLM
                return handle_sanitized_multimodal(payload)

        elif route == "ROUTE_TO_ADMIN_INTERFACE":
            # Handle requests for privileged operations (requires separate auth)
            return handle_admin_request(payload)

        else:
            raise RoutingError("Unknown request type.")

    except (RequestSyntaxError, PayloadIntegrityError, SandboxError, RoutingError) as
    e:
        log_event(origin, type(e).__name__, str(e))
        # Return a generic error response
        return {"status": "error", "message": str(e)}
```

Listing 1: High-level pseudocode for the Countermind request handling logic.

10 Evaluation Plan

As Countermind is a conceptual architecture, this section outlines a proposed protocol for its empirical evaluation. The goal is to demonstrate its effectiveness against various attack vectors and to quantify its performance trade-offs. The evaluation is structured around the core metrics defined in Section 3.2.

10.1 Proposed Protocol, Baselines, and Measurement Setup

A rigorous evaluation would require a standardized setup to ensure reproducibility and fair comparison. Since these tests have not yet been conducted, this section describes the intended methodology.

- **Models and Datasets:** The protocol would be executed against a diverse set of state-of-the-art LLMs (e.g., Llama 3.1, GPT-4o, Claude 3.5) to assess generalizability. Evaluation datasets would include standard jailbreak suites (e.g., AdvBench), indirect prompt injection corpora, and custom datasets for the attacks described in this paper.
- **Hardware:** All tests would be conducted on a standardized hardware platform (e.g., a cloud instance with NVIDIA H100 GPUs) to ensure consistent performance measurements.
- **Measurement Setup:** The metrics would be measured as follows:
  - **ASR:** An automated judge LLM, supplemented by human review for ambiguous cases, would classify the responses to a benchmark set of adversarial prompts as either successful or failed attacks. Confidence intervals would be calculated for all results.
  - **Abstention & FBR:** These would be measured against curated datasets containing both harmful and benign prompts, respectively.
  - **Latency:** The end-to-end response time for each request would be measured, from API call to response receipt, and averaged over multiple runs. This is particularly important for understanding the performance cost of each defensive layer, as components like input filters, cryptographic checks, and output validation all contribute to latency overhead [27].

- **Baselines:** The performance of the full Countermind architecture would be compared against several baselines:
  1. **No Defense:** The raw, unprotected base LLM, to establish a worst-case security baseline.
  2. **Standard Guardrails:** A configuration designed to mimic typical industry guardrail solutions. This would involve implementing input keyword filtering, output moderation using a classifier, and basic topic control. This provides a benchmark against the current state of the art in reactive defenses.
  3. **Countermind (Ablated):** Various configurations of Countermind with key components disabled (e.g., Byte-Gate only, +Text-Crypter, +PSR). This allows for an ablation study to measure the specific contribution of each architectural layer to the overall security and performance profile [33].

10.2 Analysis of Attack Scenarios

The evaluation would test the system against a wide range of attacks, including those from standard benchmarks and creative, semantically complex attacks.

- **Poetic Non-Instructions Attack:** An attack where malicious instructions are camouflaged within poetic or metaphorical language.
  - **Countermind's Defense:** The primary defense is two-fold. First, the Intent-Detector in the **SBL** would flag a significant mismatch between the declared metadata and the poetic, instructional nature of the payload. Second, even if the request passes the perimeter, the PSR's strict thematic isolation would prevent the model from acting on the instructions, as the Poetry cluster would not be granted SYNTH rights for system commands.
- **Morphological Attacks:** Attacks that use non-standard character encodings, homoglyphs, or other morphological tricks to obfuscate malicious keywords.
  - **Countermind's Defense:** These attacks would be defeated at the earliest possible stage by the **Byte-Gate**. Its strict character allowlist, designed to accept only the limited Base62 alphabet of the Text Crypter's output, would immediately reject any input containing such characters.
- **Correction Exploit:** A multi-step social engineering attack where the adversary first submits a query with a deliberate "mistake," then follows up with a "correction" that contains the malicious payload.
  - **Countermind's Defense:** This is a behavioral attack that would be caught by the stateful components of the architecture. The **Trust-Scaler** would detect the inconsistent and unusual interaction pattern, leading to a penalty on the session's Trust Score. Furthermore, the **Context-Delta Sentinel (CDS)** would flag the abrupt and significant semantic shift between the initial query and the "correction" as a high-risk anomaly.

The following tables present illustrative and conceptual results, representing the *expected outcomes* of the proposed evaluation protocol.

Table 5: Conceptual Results: Attack Success Rate (ASR) Comparison (Lower is Better).

Attack Type	**Baseline ASR (No Defense)**	**Baseline ASR (Std. Guardrails)**	**Countermind ASR**	**Δ vs. Guardrails**
Direct Prompt Injection	95%	40%	$< 1\%$	-39%
Jailbreak (Role-Playing)	80%	30%	5%	-25%
Indirect Prompt Injection	70%	60%	10%	-50%
Multimodal Attack (Typographic)	85%	80%	$< 5\%$	-75%
Context Poisoning (Long-term)	60%	55%	$< 5\%$	-50%

10.3 Performance Overhead Analysis

A critical aspect of the evaluation is to quantify the performance cost of the added security layers.

Table 6: Conceptual Results: Performance and Latency Overhead.

Configuration	Avg. Latencyper Turn (ms)	LatencyOverhead	Throughput(req/sec)
Baseline (No Defense)	300	0%	100
+ **SBL**	350	+16.7%	85
+ Multimodal Sandbox (Image)	650	+116.7%	46
+ PSR	400	+33.3%	75
Countermind (Full System, Text)	450	+50%	66

As the table illustrates, the most significant latency overhead is introduced by the Multimodal Sandbox, due to the computationally intensive nature of tasks like nudity classification on high-resolution images. The core text-based defenses **SBL and PSR** introduce a more moderate but still significant overhead. This highlights the trade-off between security and performance that must be considered during deployment.

10.4 Conceptual Ablation Study: Contribution of Individual Layers

An ablation study is a standard methodology for understanding the contribution of individual components to a complex system [33]. A conceptual ablation of Countermind reveals the critical role of each layer in its defense-in-depth strategy.

- **Without Semantic Boundary Logic (SBL)** The system would be immediately vulnerable to all forms of syntactic and structural attacks. The absence of the Text Crypter would reopen the entire plaintext prompt injection attack surface, rendering the deeper layers largely ineffective against these common threats. The ASR for direct prompt injections would likely revert to the levels of standard guardrails or worse.
- **Without Parameter-Space Restriction (PSR):** The system could block many attacks at the perimeter but would remain vulnerable to sophisticated semantic attacks, zero-day jailbreaks that use novel language, and dangerous emergent behaviors. An adversary could craft a syntactically valid and cryptographically signed prompt that still co-opts the model's internal reasoning to produce a harmful synthesis of information.
- **Without the Secure, Self-Regulating Core:** The system would be static. It could defend against known attacks but would be unable to learn from new patterns observed in the wild. Over time, its defenses would become obsolete as adversaries develop new techniques to bypass its fixed rules. The lack of an immutable audit log would also make forensic analysis and incident response significantly more difficult.

This analysis demonstrates that the strength of the Countermind architecture lies not in any single component, but in their synergistic integration. Each layer addresses a different class of threats, and their combination provides a resilient, layered defense.

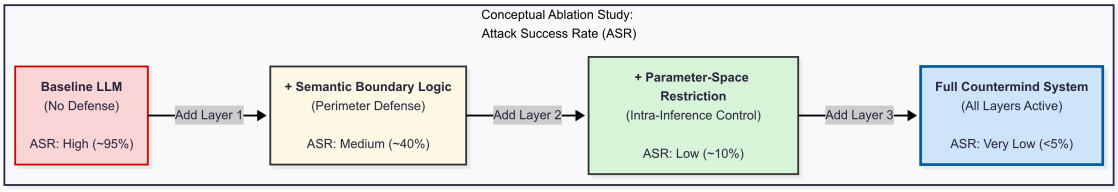

Figure 3: Conceptual Ablation Study - ASR vs. Enabled Components.

11 Discussion

The Countermind architecture represents a departure from current LLM security practices. This section discusses the broader impact of this approach, its applicability to emerging AI paradigms, key operational considerations, and how it compares to existing defensive strategies.

11.1 Impact and Generalizability to Agents, Tools, and RAG

The principles of Countermind are not limited to simple chatbot applications but are particularly well-suited for securing the next generation of more complex, agentic systems.

- **Agents and Tool Use:** The Parameter-Space Restriction (PSR) framework provides a natural and robust mechanism for governing an agent's use of external tools. Each available tool or API can be mapped to a dedicated semantic cluster (e.g., `Tools.Email.Send`, `Tools.Database.Query`). The agent can only invoke a tool if the current request grants it the necessary rights (e.g., SYNTH) for that specific cluster. This prevents common agent vulnerabilities where a hijacked agent is coerced into using a tool for an unauthorized purpose. For example, a prompt injection attack might try to make a customer service agent call the `delete_user_account` tool, but if the PSR policy for that interaction does not include the `Tools.Admin.Delete` cluster, the action would be blocked at the activation level before the tool is ever called.
- **Retrieval-Augmented Generation (RAG):** RAG systems, which augment LLM prompts with information retrieved from external knowledge bases, are a primary vector for indirect prompt injection. An attacker can poison the knowledge base with documents containing hidden malicious instructions. Countermind addresses this in two ways. First, the OMP decomposition allows the system to treat the user's query and the retrieved documents as separate entities with different trust levels. Second, Semantic Zoning can place all retrieved content into a specific context zone (e.g., `RAG.RetrievedData`) with strict, READ-ONLY rights. This policy is enforced at the PSR level, ensuring that `RAG.RetrievedData` receives permanent READ-ONLY, with SYNTH/EVAL=deny. This ensures that the LLM can use the information from the documents to formulate its answer but cannot interpret any text within those documents as an executable instruction.

11.2 Operational Considerations: Tuning, Logging, and Fail-Safe vs. Fail-Open Design

Deploying a system like Countermind in a production environment requires careful consideration of several operational aspects.

- **Tuning and Usability:** The effectiveness of the architecture depends on the careful tuning of its various thresholds, such as the Trust Score for activating the Soft-Lock, and the confidence scores for the Multimodal Sandbox classifiers. Overly aggressive settings will lead to a high False-Block Rate (FBR), frustrating legitimate users and reducing the application's utility. Conversely, settings that are too lenient will fail to stop sophisticated attacks. Achieving the right balance requires an iterative process of testing, monitoring, and adjustment, guided by the core metrics of ASR and FBR.

- **Logging and Forensics:** The immutable audit log is not just a security feature but a critical operational tool. It provides the necessary data for debugging system failures, conducting forensic investigations after a security incident, demonstrating compliance with regulatory requirements, and providing the training data for the Learning Security Core. The integrity and availability of this log are paramount to the system's long-term health and adaptability.
- **Fail-Safe vs. Fail-Open Design:** A critical architectural decision for any security system is its behavior in the event of an internal failure, such as a component crash or a lost connection to a database. The two primary design philosophies are fail-safe (also known as fail-closed) and fail-open [34].
  - **Fail-Safe (Fail-Closed):** In this mode, the system defaults to its most secure state, which typically means blocking the request. This prioritizes security over availability. For example, if the PSR policy engine fails, a fail-safe design would reject all requests rather than allowing them to proceed to the LLM without governance.
  - **Fail-Open:** In this mode, the system defaults to an operational state, bypassing the failed security check to maintain availability. This might be appropriate for non-critical systems where uptime is more important than perfect security.
  - **Degraded-Safe Mode:** A third path exists where, upon failure of a non-critical component (e.g., the learning core), the system enters a degraded-safe mode. In this state, it might only permit low-risk operations (e.g., READ rights) while disabling more privileged ones (SYNTH, CROSS) until the component is restored. Countermind's default posture is designed to be fail-safe. Given the potential for LLMs to cause significant harm, the architecture prioritizes preventing a security failure over ensuring 100% availability.

11.3 Comparison to Existing Paradigms: Guardrails, RLHF, and Constitutional AI

Countermind builds upon concepts from existing security paradigms but differs in its fundamental approach.

- **Comparison to Guardrails:** Guardrail systems provide a powerful framework for defining and enforcing programmable safety rules for LLMs. They typically operate at the application layer, evaluating prompts and responses against a defined set of policies. Countermind complements this approach but operates at a deeper, more architectural level. The **SBL**, with its cryptographic Text Crypter, provides a structural defense that is not based on programmable rules but on verifiable mathematical properties of the input. The PSR operates at the sub-symbolic, activation level, providing a form of control that is closer to the model's core logic than a typical guardrail system. The measurable difference lies in the point of intervention: pre-inference structural validation versus post hoc content filtering, which is expected to yield a lower ASR for structural attacks and a different latency profile, as outlined in our evaluation plan.
- **Comparison to RLHF and Constitutional AI:** Reinforcement Learning from Human Feedback (RLHF) and its derivative, Constitutional AI (CAI), are the dominant methods for aligning LLM *behavior* [18]. These techniques work by training a preference model on human or AI-generated feedback and then using that model as a reward function to fine-tune the LLM to produce more desirable outputs. This is a powerful but fundamentally **reactive** training process; it teaches the model to prefer safe outputs over unsafe ones based on past examples. Countermind, in contrast, is a **proactive** inference-time control mechanism. PSR does not retrain the model but constrains its generative process in real-time. While CAI provides the LLM with a set of principles to *follow*, Countermind's Secure Core and PSR provide an architectural framework to *enforce* those principles, regardless of the model's learned behavior. This makes it potentially more robust against novel attacks that the model was not trained to handle.
- **Comparison to Point-Defenses:** The current research landscape is populated with many "point-defenses," which are specific algorithms designed to detect a particular type of attack, such as a classifier for prompt injection. While valuable, these solutions often lack generalizability. Countermind is proposed as a holistic, defense-in-depth architecture. It does not rely on a single mechanism but integrates multiple, diverse

defenses (cryptographic, syntactic, semantic, behavioral, and sub-symbolic) to provide layered and resilient protection against a wide spectrum of threats.

12 Limitations and Future Work

While the Countermind architecture presents a comprehensive conceptual framework for LLM security, it is important to acknowledge its limitations and areas for future research.

- **Implementation Complexity:** The most significant limitation is the engineering complexity of implementing the proposed system. The **SBL** and Multimodal Sandbox are non-trivial but build upon existing technologies. The Parameter-Space Restriction (PSR) mechanism, however, requires deep integration with the LLM's inference engine to manipulate activations during the forward pass. This is a substantial technical challenge that would require close collaboration with model developers or advanced expertise in modifying open-source model architectures.
- **Key Management:** The security of the Text Crypter relies on the secure provisioning, rotation, and revocation of HMAC keys for legitimate clients. A robust key management infrastructure is a critical and complex prerequisite for this architecture.
- **Semantic Cluster Definition:** The effectiveness of PSR is highly dependent on the quality and granularity of the defined semantic clusters. The process of identifying and delineating these clusters within a model's activation space is a complex research problem in itself. It would likely require a combination of unsupervised clustering techniques, interpretability tools, and significant domain expertise to create a meaningful and robust cluster map. Future work should explore semi-automated methods for defining and maintaining these semantic taxonomies.
- **Performance Overhead and User Friction:** As shown in the conceptual evaluation, the security layers introduce a non-negligible latency overhead. This may be prohibitive for applications requiring near-instantaneous responses. Furthermore, strict validation mechanisms, such as a short time-window for the Text Crypter, could lead to a high FBR due to network latency or clock skew, creating significant user friction.
- **Multimodal False Positives/Negatives:** The Multimodal Sandbox relies on classifiers that are not perfect. False negatives could allow harmful content to pass, while false positives could block legitimate user content, impacting usability. The plan for measuring these error rates is crucial for understanding the practical viability of this component.
- **Security of the Architecture Itself:** The framework does not inherently defend against attacks on the underlying infrastructure, such as the host operating system or hypervisor. A compromise of the trusted computing base would undermine the entire architecture.
- **Evasion of High-Level Controls:** While PSR is designed to be robust against many semantic attacks, it is conceivable that a highly sophisticated adversary could develop novel techniques to bypass the cluster-based controls. For example, an attacker might find ways to express a forbidden concept using only the representations available in a set of allowed clusters. The resilience of the PSR governance model against such advanced, second-order semantic attacks is an open area for investigation.

Future work should focus on building a proof-of-concept implementation of the Countermind architecture to empirically validate the conceptual claims made in this paper. This would involve tackling the challenges of PSR implementation and cluster definition, and conducting rigorous, large-scale evaluations against a comprehensive suite of adversarial benchmarks.

13 Ethics Statement & Dual-Use Considerations

The research and concepts presented in this paper are intended to advance the field of defensive AI security and promote the development of safer, more trustworthy LLM systems. The following ethical considerations have been taken into account.

- **Dual-Use:** All security research has a potential dual-use nature; techniques developed for defense can sometimes inform the development of new attacks. The focus of this work is exclusively on hardening AI systems. The architectural principles are described at a level intended to guide secure system design without providing a step-by-step blueprint for a specific exploitable vulnerability in any existing system.
- **Responsible Disclosure:** This paper does not contain details of any specific, zero-day vulnerabilities discovered in any commercial or open-source LLM products. The attack scenarios discussed are either conceptual or based on publicly known attack classes. The goal is to foster a culture of proactive security design, in line with the principles of responsible disclosure.
- **Benign Payloads:** All examples of adversarial prompts or malicious payloads described in this paper are for illustrative and evaluation purposes only. They are designed to be conceptually representative of real threats without containing actually harmful, dangerous, or illegal content.
- **User Notice and Governance for Soft-Lock:** The "Soft-Lock" or graceful degradation response mechanism presents an ethical challenge as it involves a level of deception towards the user. A User Notice Policy must be established to govern its use. This policy should define:
  - **Transparency:** The system's terms of use should clearly state that interactions may be monitored for security purposes and that the service may be gracefully degraded if malicious intent is detected.
  - **Audit and Appeal:** All Soft-Lock activations must be recorded in the immutable audit log. A clear process must be available for users to appeal a restriction if they believe it was applied in error (a false positive), triggering a human-in-the-loop review.
  - **Proportionality:** The duration of a Soft-Lock should be proportionate to the detected risk, with policies for automatic expiration and review to mitigate the impact of false positives.
- **Compliance for Multimodal Analysis:** The analysis of user-uploaded content in the Multimodal Sandbox, particularly involving pHash databases or facial recognition, must be conducted in strict compliance with legal and ethical standards. This includes a clear legal basis for processing, adherence to jurisdictional regulations (e.g., GDPR), minimal data retention policies for sensitive content, and a human review process for handling ambiguous cases or false positives. The system should perform classification and recognition, not biometric identification, unless a documented legal basis is provided.

Declarations and Statements

Use of AI Tools: Large Language Models were utilized as research assistants during the preparation of this manuscript. Their use included summarizing academic papers, assisting with the translation of conceptual notes, and providing grammar and style corrections. All core concepts, architectural designs, analyses, and the final written text were directed, authored, and verified by the human author. No section of this paper consists of unverified, directly generated output from an AI model.

Conflicts of Interest: The author declares no conflicts of interest.

Funding: This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

Data and Code Availability: This paper describes a conceptual architecture. No new datasets were generated.

Acknowledgments

The author wishes to thank the open-source community and the many researchers whose work in LLM security, interpretability, and alignment provided the foundation for the concepts explored in this paper.

References

[1] Fabricio Aguilera-Martínez and Fernando Berzal. LLM security: Vulnerabilities, attacks, defenses, and countermeasures, 2025.

[2] Andy Zou, Long Phan, Sarah Chen, Keyan Nasseri, Oliver Zhang, Angela Jiang, Mantas Mazeika, Ann-Kathrin Dombrowski, Dan Hendrycks, Jacob Steinhardt, and Avital Oliver. Representation engineering: A top-down approach to AI transparency, 2023.

[3] Simone Rossi, Andrea Mauro Michel, Raghava Rao Mukkamala, and Jason Bennett Thatcher. An Early Categorization of Prompt Injection Attacks on Large Language Models, 2024.

[4] Wenrui Xu and Keshab K. Parhi. A survey of attacks on large language models, 2025.

[5] Xiaogeng Liu, Zhiyuan Yu, Yizhe Zhang, Ning Zhang, and Chaowei Xiao. Automatic and Universal Prompt Injection Attacks against Large Language Models, 2024.

[6] Zhengchun Shang and Wenlan Wei. Evolving security in LLMs: A study of jailbreak attacks and defenses, 2025.

[7] Chun Wai Chiu, Linghan Huang, Bo Li, Huaming Chen, and Kim-Kwang Raymond Choo. "Do as I Say, Not as I Do": A semi-automated approach for jailbreak prompt attacks against multimodal LLMs, 2025.

[8] Hao Cheng, Erjia Xiao, Jindong Gu, Le Yang, Jinhao Duan, Jize Zhang, Jiahang Cao, Kaidi Xu, and Renjing Xu. Unveiling typographic deceptions: Insights into the typographic vulnerability in large vision-language models, 2024.

[9] Dezhang Kong, Shi Lin, Zhenhua Xu, Zhebo Wang, Minghao Li, Yufeng Li, Yilun Zhang, Hujin Peng, Zeyang Sha, Yuyuan Li, Changting Lin, Xun Wang, Xuan Liu, Ningyu Zhang, Chaochao Chen, Muhammad Khurram Khan, and Meng Han. A Survey of LLM-Driven AI Agent Communication: Protocols, Security Risks, and Defense Countermeasures, 2025.

[10] Miles Q. Li and Benjamin C. M. Fung. Security concerns for large language models: A survey, 2025.

[11] Chi Zhang, Changjia Zhu, Junjie Xiong, Xiaoran Xu, Lingyao Li, Yao Liu, and Zhuo Lu. Guardians and offenders: A survey on harmful content generation and safety mitigation of LLM, 2025.

[12] Xin Chen, Yarden As, and Andreas Krause. Learning safety constraints for large language models, 2025.

[13] Sibo Yi, Yule Liu, Zhen Sun, Tianshuo Cong, Xinlei He, Jiaxing Song, Ke Xu, and Qi Li. Jailbreak attacks and defenses against large language models: A survey, 2024.

[14] Nishant Balepur, Rachel Rudinger, and Jordan Lee Boyd-Graber. Which of these best describes multiple choice evaluation with LLMs? a) forced b) flawed c) fixable d) all of the above, 2025.

[15] Jan Wehner, Sahar Abdelnabi, Daniel Tan, David Krueger, and Mario Fritz. Taxonomy, opportunities, and challenges of representation engineering for large language models, 2025.

[16] Alexander Matt Turner, Lisa Thiergart, Gavin Leech, David Udell, Juan J. Vazquez, Ulisse Mini, and Monte MacDiarmid. Steering language models with activation engineering, 2023.

[17] Bruce W. Lee, Inkit Padhi, Karthikeyan Natesan Ramamurthy, Erik Miehling, Pierre Dognin, Manish Nagireddy, and Amit Dhurandhar. Programming refusal with conditional activation steering, 2024.

[18] Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Tom Conerly, Emily Chen, Andy Jones, Deep Ganguli, Anna Goldie, Ben Mann, Nelson Joseph, Sam McCandlish, Kamaldeep Singh, Jared Kaplan, and Dario Amodei. Constitutional ai: Harmlessness from AI feedback, 2022.

[19] Tiejin Chen, Pingzhi Li, Kaixiong Zhou, Tianlong Chen, and Hua Wei. Unveiling privacy risks in multi-modal large language models: Task-specific vulnerabilities and mitigation challenges. In *Findings of the Association for Computational Linguistics: ACL 2025*, 2025.

[20] Tingmin Wu, Shuiqiao Yang, Shigang Liu, David Nguyen, Seung Jang, and Alsharif Abuadbba. Threatmodeling-LLM: Automating threat modeling using large language models for banking system, 2024.

[21] Yuchen Yang, Yiming Li, Hongwei Yao, Bingrun Yang, Yiling He, Tianwei Zhang, Dacheng Tao, and Zhan Qin. Shadowcode: Towards (automatic) external prompt injection attack against code LLMs, 2024.

[22] Andy Zou, Zifan Wang, Nicholas Carlini, Milad Nasr, J. Zico Kolter, and Matt Fredrikson. Universal and transferable adversarial attacks on aligned language models, 2023.

[23] Liangxuan Wu, Chao Wang, Tianming Liu, Yanjie Zhao, and Haoyu Wang. From assistants to adversaries: Exploring the security risks of mobile LLM agents, 2025.

[24] Michael Freenor, Lauren Alvarez, Milton Leal, Lily Smith, Joel Garrett, Yelyzaveta Husieva, Madeline Woodruff, Ryan Miller, Erich Kummerfeld, Rafael Medeiros, and Sander Schulhoff. Prompt optimization and evaluation for LLM automated red teaming, 2025.

[25] Bingbing Wen, Jihan Yao, Shangbin Feng, Chenjun Xu, Yulia Tsvetkov, Bill Howe, and Lucy Lu Wang. Know your limits: A survey of abstention in large language models, 2024.

[26] Boshi Huang and Fabio Nonato de Paula. Defend llms through self-consciousness, 2025.

[27] Zhifan Luo, Shuo Shao, Su Zhang, Lijing Zhou, Yuke Hu, Chenxu Zhao, Zhihao Liu, and Zhan Qin. Shadow in the cache: Unveiling and mitigating privacy risks of KV-cache in llm inference, 2025.

[28] Francesco Periti and Stefano Montanelli. *Lexical Semantic Change through Large Language Models: a Survey*. PhD thesis, University of Milan, 2024.

[29] T. Balarabe. Understanding LLM context windows, tokens, attention, and challenges. Medium blog post, 2025. non-peer-reviewed.

[30] Yuchen Yang, Qichang Liu, Christopher Brix, Huan Zhang, and Yinzhi Cao. Certphash: Towards certified perceptual hashing via robust training. In *Proceedings of the 34th USENIX Security Symposium*, 2025.

[31] Tom Or and Omri Azencot. Unraveling hidden representations: A multi-modal layer analysis for better synthetic content forensics, 2025.

[32] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson, 4th edition, 2020.

[33] Richard Meyes, Melanie Lu, Constantin Waubert de Puiseau, and Tobias Meisen. Ablation studies in artificial neural networks, 2019.

[34] Ross J. Anderson. *Security Engineering: A Guide to Building Dependable Distributed Systems*. Wiley, 2nd edition, 2008.

Appendices

Appendix A: Cluster Policy Artifact

```
version: 1
default: deny
clusters:
  READ:
    allow: [literal_lookup]              # retrieval without synthesis
    deny: [code_generation, cross_source_aggregation]
    thresholds: { trust_score_min: 0.70, human_review: false }

  SYNTH:
    allow: [summarize, paraphrase]
    deny: [code_execution, system_calls]
    thresholds: { trust_score_min: 0.80, human_review: true }

  EVAL:
    allow: [static_analysis, constraint_check]
    deny: [external_write]
    thresholds: { trust_score_min: 0.75, human_review: true }

  CROSS:
    allow: [cross_source_compare]
    deny: [freeform_generation, external_post]
    thresholds: { trust_score_min: 0.85, human_review: true }

audit:
  mode: append_only
  fields: [timestamp, user, cluster, decision, reason]
```