Preprint for 2026 56th Annual IEEE International Conference on Dependable Systems and Networks - Supplemental Volume (DSN-S)

AEGIS: Preventing Cross-Domain Resource Abuse in MCP

Shriti Priya
IBM Research, USA
shritip@ibm.com

Teryl Taylor
IBM Research, USA
terylt@ibm.com

Frederico Araujo
IBM Research, USA
frederico.araujo@ibm.com

***Abstract*—The Model Context Protocol (MCP) is an open-source JSON-RPC protocol that standardizes how large language models (LLMs) interact with external systems through programmatic functions known as tools. Attackers or malicious agents can exploit certain modalities of these MCP tools to degrade the overall quality of service of agent-based applications. For example, an agent may request an excessively large search radius or very long videos, overloading backend systems and potentially causing slowdowns or denial-of-service. Each modality—including text, images, video, and location—introduces distinct vectors for resource abuse, complicating the development of consistent mitigation strategies. Moreover, multimodal and cross-domain tools expose diverse request schemas and parameters, making it difficult to define policies that are both generalizable and precise enough to enforce meaningful resource constraints.**

In this paper, we present AEGIS, a policy enforcement component that enables administrators to define fine-grained safeguards against resource abuse across heterogeneous MCP tools and modalities. AEGIS leverages the reasoning capabilities of large language models to analyze, categorize, and normalize diverse tool invocations into a unified, policy-friendly representation accessible to security practitioners. Integrated with the Open Policy Agent and the ContextForge AI Gateway, AEGIS detects and mitigates abusive behaviors while preserving the flexibility of MCP-based agent ecosystems.

***Index Terms*—Agentic security, Resource abuse, MCP, Policies, LLMs**

I. INTRODUCTION

The Model Context Protocol (MCP) [1] defines a standardized JSON-RPC interface through which large language models (LLMs) interact with external systems via programmatic tools. As the number of MCP servers continues to grow—driven by the flexibility and integration advantages they offer—the attack surface of MCP-based systems expands accordingly. One emerging risk is resource exhaustion caused by the parametric exploitation of tools.

A recent study by EnkryptAI [2] evaluated 1,000 MCP servers and found that 15% were vulnerable to resource exhaustion issues, including missing pagination limits, unbounded memory loops, memory leaks, and servers that could be brought down by a single malicious request. The study highlighted `kubernetes-mcp-server` as an illustrative case, where unconstrained parameters such as `replicas`, `timeout`, and `label_selector` can trigger excessive resource consumption within a Kubernetes cluster, ultimately leading to service degradation or complete denial of service.

Prior work has explored input validation [3], resource quotas [4], and runtime enforcement mechanisms [5] to mitigate such issues in MCP servers. Despite these efforts, new MCP servers continue to be deployed rapidly. A widely adopted runtime enforcement approach is the MCP Gateway, which acts as an intermediary layer that validates interactions between agents and MCP servers. These gateways host a diverse set of servers that agents access based on their operational requirements. To strengthen security at this layer, several systems introduce pluggable policy enforcement mechanisms, including OPA [6], rate-limiting, and Cedar-based plugins [7]. While these plugins enable policy-driven enforcement, significant challenges arise when such mechanisms are applied across large and heterogeneous MCP ecosystems.

Diverse Modalities. Modern MCP servers support multiple modalities—including text, image, audio, video, and spatial data such as location. Agents frequently operate across modalities within a single task. For example, the request "Find restaurants opened from 2020 to now within 50 miles, cap at 100 results, and generate a table showing each restaurant's name with images of its top three food specialties at 300 dpi," which requires the agent to interact with three MCP servers for location queries, text retrieval, and image generation. Each modality introduces distinct vectors for resource abuse: retrieving 100 text records is far less demanding than retrieving or generating 100 videos. These differences complicate the design of consistent policy controls.

Diverse Parameter Characteristics. The same example involves heterogeneous parameter types, e.g., temporal (2020 to present), volumetric (100 results, three images), and qualitative (50-mile radius, 300 dpi). Each parameter type can trigger excessive resource consumption. Without appropriate bounds, extreme values—such as querying historical data since the year 1800, generating large numbers of high-resolution images, or expanding spatial queries indefinitely—can overload backend services. Effective policies must therefore account for parameter characteristics and their potential for misuse.

Diverse Operations. MCP tool invocations may involve generation, retrieval, or deletion operations. Retrieving large datasets can strain storage and network resources, while generating high-resolution images or videos can consume substantial compute. Furthermore, generating 100 images is

typically far more resource-intensive than retrieving 100 existing images. These operational differences further complicate the definition of consistent resource-control policies.

Diverse Parameter Semantics. Across hundreds of MCP servers and registries, parameters that represent the same concept are often named differently. For example, location tools may use parameters such as `radius`, `initial_radius`, or `end_position`, while text-based servers may limit results using `limit`, `maxResults`, or `count`. This semantic inconsistency significantly complicates policy authoring and enforcement, forcing policy writers to manually track and normalize heterogeneous parameter vocabularies.

Policy Complexity Explosion. As a result, policy authors must understand not only the modalities supported by each server but also the parameters and resource-abuse vectors associated with each modality across dimensions such as time, quality, and quantity. Performing this analysis for every tool across a large MCP ecosystem is extremely difficult. It requires deep contextual knowledge of tool semantics and system behavior, making policy development complex, time-consuming, and difficult to scale.

To address these challenges, we present AEGIS, a security system that leverages the reasoning capabilities of large language models to assist in the creation and enforcement of policies that protect against resource abuse across diverse modalities and heterogeneous MCP servers. The main contributions of this paper are:

- A systematic investigation that identifies key parameters contributing to resource exhaustion across MCP servers.
- A methodology that uses LLMs to extract and normalize resource-related parameters from tool invocations into a unified taxonomy, enabling reusable policy templates and scalable policy enforcement across heterogeneous systems.
- A threshold estimation approach that provides end-to-end protection against resource abuse attacks in MCP servers.

II. Background and Related Work

The Model Context Protocol (MCP) [1] is an open protocol that enables AI agents to discover and connect to external tools, data sources, and services through a client–server architecture. MCP supports multimodal data through typed content blocks (e.g., text, image, audio) and binary resources shared using MIME types and base64-encoded blobs. In production deployments, MCP gateways are often placed in front of multiple MCP servers to provide a unified tool interface. A gateway enables centralized routing, authentication, access control, and observability. Examples include the ContextForge AI Gateway [8] and TrueFoundry [9].

Recent studies [10], [11] highlight resource abuse as an emerging security concern in agent-driven systems. Because agents autonomously generate tool calls and parameters, seemingly valid requests may still trigger expensive operations, potentially leading to denial-of-service, increased operational costs, or cascading failures in distributed environments. The OWASP *Agentic AI Threat and Mitigation* report [12] similarly identifies resource overload as a key risk when agents overuse compute, network, or data resources.

Empirical studies [13], [14] show that malicious tool metadata or cyclic registries can significantly inflate token usage and runtime while still producing apparently correct outputs. *LeechHijack* [15] further demonstrates that MCP tools can parasitize compute budgets without violating explicit permissions. At the ecosystem level, measurement studies such as MCPDiFF [16] and the large-scale MCP server study [17] reveal that description–implementation mismatches and weak operational practices can conceal expensive or risky behavior.

Benchmark suites including MCPSecBench [18], MCP-Tox [19], and MCP Security Bench expand evaluation coverage of MCP attack surfaces. Several defenses have also been proposed. Systems such as MCP Guardian [20], MCP-specific penetration testing [21], MindGuard [22], ETDI [23], MCP-Guard [5], and MCIP [24] introduce techniques including rate limiting, resource quotas [4], provenance tracking [19], signed tool definitions [23], [25], and loop-aware monitoring. However, the literature still lacks standardized cost semantics for tool schemas and practical trajectory-level verification for real deployments. Other work [26] further shows that rate limiting alone is insufficient when workflows cascade across multiple tools, motivating cost-aware and workflow-aware resource controls.

None of these approaches address the *fine-grained, parameter-specific resource controls* that AEGIS introduces *across heterogeneous and multimodal MCP servers*. Prior work mainly enforces static constraints, such as string length limits or numerical bounds, via resource quotas. In contrast, AEGIS applies dynamic, parameter-aware controls at runtime, enabling policies that incorporate argument semantics and enforce CPU and memory limits per invocation.

III. Threat Model

We consider a distributed agentic system in which an agent workflow interacts with an LLM provider and multiple MCP servers deployed across diverse domains. Each MCP server exposes tools that operate across different modalities. We assume that these tools lack robust parameter validation, creating opportunities for misuse by a compromised agent or legitimate user's unintentional excessive request.

Our work focuses on the *Resource Overload (T4)* threat identified in the OWASP Agentic AI Threat and Mitigation report [12]. In this threat model, an attacker or misuse scenario targets the computational, memory, or service capacities of AI systems in order to degrade performance or cause partial system failures. We do not consider network-level denial-of-service attacks or vulnerabilities within the MCP protocol itself; our focus is on misuse of tool parameters and resource-intensive invocations. The scope excludes high-volume request flooding and instead targets individual requests with excessive or unbounded parameter values that can trigger disproportionate resource consumption.

IV. Approach

Our approach addresses resource abuse in MCP servers through a *Tool Ontology* that formalizes and simplifies tool definitions. The ontology categorizes tools and their parameters, enabling reusable policies and thresholds that prevent resource abuse. This section introduces the ontology and policy model; the next section presents the system architecture.

A. Tool Ontology

Figure 1(a) illustrates an MCP tool definition for a `generateImages` tool that generates `n` images at a specified `resolution`. The figure also shows the corresponding *Tool Ontology*, which extracts the following information from the tool definition.

Modality. The primary data type processed by a tool (e.g., image, video, audio, text, location, or human-in-the-loop input). For example, `generateImages` is classified as an image modality.

Operation. The action performed by the tool. We classify tools into three categories: *transform* (create or modify data), *retrieve* (access existing data), and *delete* (remove data). These distinctions capture different resource profiles. For instance, generating 1,000 images is significantly more expensive than retrieving 1,000 existing records. Similarly, retrieving video data consumes more bandwidth and compute than retrieving text. The `generateImages` tool is therefore classified as a transform operation.

These distinctions also guide policy measurement units. Text retrieval may use a count-based parameter, whereas video retrieval may require time-based thresholds (e.g., duration in seconds).

Parameter Characteristics. Tool parameters are categorized according to their role:

- *Volumetric*: controls data volume (e.g., `limit`, `count`)
- *Timebound*: controls temporal extent (e.g., video duration)
- *Qualitative*: controls output quality (e.g., image resolution)

In the `generateImages` example, parameter `n` is volumetric (number of images), while `resolution` is qualitative.

Parameter Normalization. To support reusable policies, parameters are normalized into a set of common names across tools and servers. We derived this vocabulary by manually analyzing 56 widely used MCP servers mentioned in Section VII-A, examining their tool definitions, and identifying parameters associated with resource abuse across modalities. The resulting unified parameter taxonomy is shown in Table I. In Figure 1(b), for example, parameter `n` is normalized to `num_records`.

B. Policy Templates

Once parameters are normalized, policies can be expressed using generic templates such as the one shown in Figure 1(c). Policies reference parameters using dot notation derived from the ontology layers described above. This abstraction decouples policies from individual tool implementations, reduces policy complexity, and enables reuse across heterogeneous MCP servers.

TABLE I
Resource abuse parameters across modalities

Modality	Parameters
Image	Aspect Ratio (`aspect_ratio`), number of images (`num_records`), scaling percent (`scaling_percent`), width (`width`), height (`height`), channels (`num_channels`), resolution (`resolution`)
Video/Audio	Start time (`start_time`), end time (`end_time`), bitrate (`bitrate`), resolution (`resolution`), number of videos (`num_records`), channels (`num_channels`), format (`format`), sampling rate (`sampling_rate`)
Text	Size, start/end time (`start_time`, `end_time`), record count (`num_records`), filters (`filter_array`), repeat interval (`repeat_interval`), timeout (`timeout`), breadth/depth (`max_breadth`, `max_depth`), retries (`retries`)
Location	Radius (`radius`), start position (`start_position`), end position (`end_position`), waypoints (`waypoints`), precision (`precision`)
Human	Number of users (`num_users`)

C. Policy Thresholds

Policy templates require carefully chosen thresholds to prevent resource abuse without disrupting normal operation. Threshold selection is challenging because servers handling different modalities (e.g., text, image, or audio) exhibit different baseline CPU and memory usage.

We therefore derive thresholds using an acceptable tool-call error rate as the stability criterion. Parameter values are increased until system performance degrades beyond the specified error tolerance. The highest value that maintains stable operation is selected as the final threshold and incorporated into the policy.

V. Aegis System Architecture

Aegis generates tool ontologies from tool definitions, constructs policy templates, and estimates policy thresholds. The resulting policies are enforced through an MCP gateway. Aegis operates in two phases: *Bootstrapping* and *Runtime*.

A. Bootstrapping Phase

Bootstrapping is an offline process that prepares policies before deployment. Figure 2 illustrates the bootstrapping phase of Aegis, which consists of seven steps described below:

Collection (Step 1). At first Aegis retrieves tool definitions registered with the MCP gateway using the `list_tools()` API, which returns tool schemas and metadata.

Ontology Identification (Step 2). Each tool definition is processed by an LLM using a structured system prompt (Figure 3) to generate a corresponding tool ontology. The prompt guides the model through sequential reasoning using generalized clues and reasoning patterns derived from our analysis of the 937 tool definitions described in Section VII-A. The system prompt consists of four components:

```
{
    "name": "generateImages",
    "description": "A tool to generate images
    using LLM"
    "arguments": {
        "n": "Number of images",
        "resolution": "Image resolution"
    }
}
```

(a) Tool Definition

```
{
    "modality": "image",
    "operation": "transform",
    "parameter_classification": {
        "n": "volumetric",
        "resolution": "qualitative",
    },
    "parameter_normalization": {
        "n": "num_records",
        "resolution": "resolution",
    }
}
```

(b) Tool Ontology

```
default allow = false

allow if {
    input.server.name == "image-server"

    input.image.transform.volumetric.num_records <=
    50
        input.image.transform.qualitative.resolution
    <= 300
}
```

(c) Policy Template

Fig. 1. From tool definition to policy template

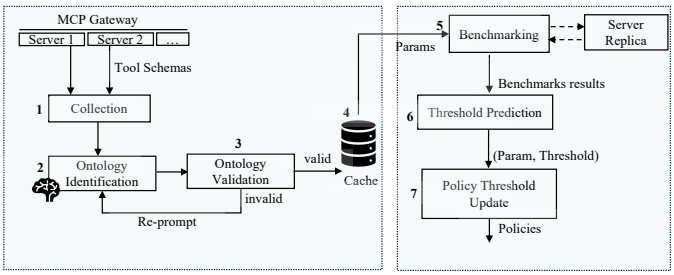

Fig. 2. AEGIS Bootstrapping

- *Multi-step reasoning*: The model performs six sequential steps: (1) classify the tool modality, (2) identify the operation type, (3) determine whether parameters may cause resource abuse, (4) identify resource-related parameters, (5) classify parameter characteristics, and (6) normalize parameters using the predefined taxonomy in Table I. Each step is supported by curated clue–reasoning pairs (Figure 3). For example, the variable `clue_reasoning_modalities` contains patterns such as: *"Requests involving screenshots, image or chart generation (e.g., bar, pie, line, funnel, violin, combo, Sankey), diagrams, social media or other visual content retrieval, publishing, analysis, editing, or transformation of images."* These clues help guide the model toward the intended task and reduce false positives. Similar definitions are provided for all remaining steps.
- *Normalization*: The prompt includes the predefined parameter normalization taxonomy (Table I), along with associated clues and reasoning patterns used to identify and map parameters from tool definitions to the normalized schema.
- *Server context*: Some tools rely on server-specific terminology whose meaning may not be globally obvious. For example, on a Pinterest server, the term *pin* refers specifically to an image. To ensure accurate interpretation, additional contextual information is provided through the `server_context` variable (Figure 3). This context may be manually curated or generated automatically using an LLM or RAG pipeline.
- *Strict structured output*: The LLM outputs the tool ontology (Figure 1(b)) as a strict JSON schema. Enforcing a structured format reduces hallucinations, simplifies parsing, and enables automated structural validation.

Ontology Validation (Step 3). AEGIS validates the LLM output by ensuring it conforms to the expected JSON structure,

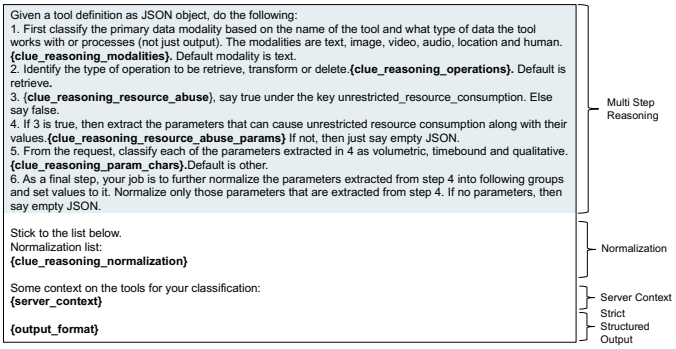

Fig. 3. System prompt for ontology identification

verifying that normalized keys match the predefined taxonomy, and confirming that extracted parameters appear in the original tool definition. Invalid outputs trigger re-prompting or are flagged for human review.

Ontology Cache (Step 4). Ontology identification is performed offline to avoid runtime latency. Validated ontologies are cached so that normalized parameters can be applied during policy evaluation. The cache key includes the server name, tool name, and parameters. When new servers are registered, their tools are automatically analyzed and added to the cache.

Benchmarking (Step 5). To estimate policy thresholds, AEGIS benchmarks servers using the identified resource-related parameters. A load generator varies parameter values and concurrent user counts while measuring latency, throughput, error rate, CPU, and memory usage. Benchmarking may run from several hours to multiple days depending on the server and parameter space.

Threshold Prediction (Step 6). Benchmark results are analyzed to determine parameter thresholds. The system selects the highest parameter value that maintains error rates within the acceptable tolerance across all load levels.

Policy Threshold Update (Step 7). The computed thresholds are inserted into the policy templates for each server. Once configured, the policies are ready for deployment in the runtime phase.

B. Runtime Phase

Figure 4 illustrates the runtime phase of AEGIS. At runtime, policies are enforced using a framework such as Open Policy

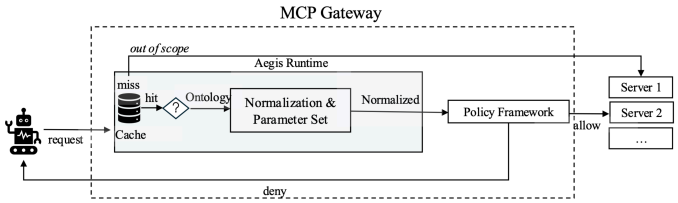

Fig. 4. AEGIS Runtime

Agent (OPA). During a tool invocation, AEGIS checks whether the tool ontology exists in the cache. If present, the request parameters are normalized and evaluated against the policy. Requests within threshold limits are forwarded to the MCP server; otherwise they are rejected.

If the ontology is not cached, the request bypasses normalization and is considered *out of scope* for policy enforcement.

VI. IMPLEMENTATION

The runtime component of AEGIS was implemented using the ContextForge AI Gateway [8]. The system leverages the gateway's plugin framework, which enables modular extension and policy control. To manage policy enforcement, the pre-existing OPA (Open Policy Agent) plugin was adopted as the foundational enforcement layer. In this configuration, each tool invocation request is first processed by AEGIS before being forwarded to OPA.

During the bootstrapping phase AEGIS collects all the tool definitions of the registered servers in the gateway running ontology identification using the Claude 4 Sonnet [27] LLM. Ontologies are cached in the Redis Database.

VII. EVALUATION

We evaluate AEGIS through the following research questions.

RQ1. How accurately can AEGIS identify and categorize tool ontologies from MCP tool definitions?

RQ2. How effective is AEGIS at protecting MCP servers against resource abuse attacks?

A. Dataset

Since no publicly available labeled dataset exists for this task, we curated a dataset from 56 MCP servers spanning diverse modalities in the OpenTools MCP Registry [28]. Across these servers, we collected 937 tool definitions. The corresponding tool schemas were crawled from the registry, and a custom parser extracted relevant metadata including tool names, descriptions, parameters, and associated server information.

The dataset covers a broad set of real-world MCP categories, including Cloud Infrastructure (AWS, Azure, Alibaba, Qiniu), Database and Data Management (Pinecone, Meilisearch, YDB, ClickHouse), Productivity and Collaboration (Notion, Sanity, Backlog), Search and Web tools (Tavily, Exa Search, Browserbase, DataForSEO), API and E-commerce platforms (Stripe, Shopify Dev, Chargebee Agentkit), Analytics and Gaming services (OP.GG, Graphlit, PubNub, AntV Chart), and Development tools (E2B, Semgrep, Metricool).

Each of the 937 tools was manually reviewed and assigned a primary modality label from the following categories: text, image, human, video, audio, or location. A tool was labeled according to the primary data modality it processes (e.g., image generation or resizing tools were labeled as image). Tools not associated with image, video, audio, location, or human interaction were labeled as text.

We further annotated each tool by operation type, classifying them as *retrieve*, *transform*, or *delete*. Next, we identified whether a tool contained parameters capable of causing unrestricted resource consumption. A tool was labeled as resource-sensitive if at least one parameter could potentially lead to excessive resource usage.

The extracted parameters were then categorized according to their characteristics as *volumetric*, *qualitative*, or *time-bound*. Finally, all parameter names were normalized using the predefined parameter taxonomy described in Section IV Table I.

Dataset Insights. Among the 937 tool definitions, 869 belong to the text modality, 38 to image, 14 to location, 10 to video, 5 to audio, and 1 to human.

In terms of operation type, 572 tools were classified as retrieve operations, 308 as transform operations, and 57 as delete operations. Out of all tools, 441 contained parameters capable of causing potential resource abuse.

B. Accuracy (RQ1)

To evaluate the bootstrapping accuracy of AEGIS, we compared its ontology identification results against the manually labeled ground truth using standard classification metrics including accuracy, precision, recall, and F1-score (Table II).

Experiments were conducted using Claude 4 Sonnet LLM and repeated across three runs with default inference settings with each inference taking approximately 0.52 seconds. Overall results show more than 84% accuracy across all ontology identification tasks. The model performed best on modality, operation, parameter characteristics classification and resource consumption detection tasks. Parameter normalization has lower performance metrics as compared to other tasks due to the complexity of the problem. Misclassifications primarily occurred in cases requiring additional contextual interpretation. For example, distinguishing between storing a binary image file and posting an image on a social media platform requires understanding platform semantics. In systems such as Instagram, the term "post" may refer to either text or image content, leading to occasional ambiguity in modality classification.

C. Effectiveness Against Resource Abuse Attacks (RQ2)

To evaluate the effectiveness of AEGIS in mitigating resource abuse attacks, we conducted controlled experiments measuring policy enforcement under varying load conditions. This required empirically deriving thresholds for abuse-prone

TABLE II
ONTOLOGY IDENTIFICATION PERFORMANCE SUMMARY (MEAN ± STD ACROSS 3 RUNS)

Category	Metric	Mean ± Std
Modality	Accuracy	0.9868 ± 0.0018
	Precision	0.9879 ± 0.0016
	Recall	0.9868 ± 0.0018
	F1-Score	0.9872 ± 0.0017
Operation	Accuracy	0.9740 ± 0.0013
	Precision	0.9743 ± 0.0015
	Recall	0.9740 ± 0.0013
	F1-Score	0.9742 ± 0.0014
Resource Consumption	Accuracy	0.9196 ± 0.0005
	Precision	0.9684 ± 0.0012
	Recall	0.8571 ± 0.0000
	F1-Score	0.9094 ± 0.0005
Parameter Characteristics Classification	Exact Match	1.0000 ± 0.0000
	Key Match	1.0000 ± 0.0000
	Partial Match	1.0000 ± 0.0000
Parameter Normalization	Exact Match	0.8321 ± 0.0035
	Key Match	0.8403 ± 0.0033
	Partial Match	0.8342 ± 0.0035

N = 937 samples per run

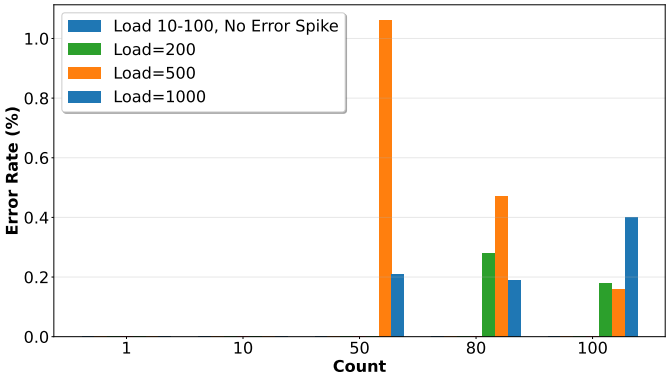

Fig. 5. The threshold estimation was conducted by benchmarking the CIFAR-10 MCP Server using the `get_images` tool, which retrieves images based on specified count parameters. The experiment comprised 118 benchmark runs across varying load levels (10, 25, 50, 100, 200, 500, and 1000) and image count values (1, 10, 50, 80, and 100).

parameters and testing whether AEGIS could prevent overload while maintaining normal system operation.

For controlled deployment and measurement, we implemented a custom image-based MCP server called *CIFAR-10 FastMCP Server* using the FastMCP framework. The server exposes the CIFAR-10 dataset [29], which contains 60,000 32×32 color images across 10 object classes, as AI-callable tools that allow agents to retrieve and manipulate images. Images are stored as raw RGB bytes in a SQLite database (`cifar10.db`) configured with WAL (Write-Ahead Logging) mode, a 64 MB page cache, and a 30-second lock timeout to support safe concurrent reads. The server exposes nine tools in total. Our experiments focus on the `get_images` tool, where the `count` parameter controls the number of images returned and can be manipulated to induce resource exhaustion.

To estimate policy thresholds, we varied the number of concurrent users (10, 25, 50, 100, 200, 500, 1000) invoking

TABLE III
DETAILED COMPARISON: CIFAR-10 FASTMCP SERVER PERFORMANCE WITHOUT VS. WITH AEGIS

Metric	Without AEGIS	With AEGIS
Total Load	200	200
Successful	4	200
Failed	196	0
Throughput (req/s)	1.34	30.03
Min Latency (ms)	1064.84	469.79
Median Latency (ms)	15380.94	2366.13
P95 Latency (ms)	15562.50	2863.94
CPU Max (%)	24.85	29.07
CPU Avg (%)	36.40	47.84
Memory Max (MB)	504.77	387.72
Memory Avg (MB)	380.17	4.17
Wall Time (s)	146.27	4.17

`get_images` and the `count` parameter (1, 10, 50, 80, 100) as mentioned in Figure 5. In total, 118 benchmark runs were executed, with each run taking approximately 8 minutes. The experiments were run on a MacBook Pro (Apple M1 Max chip with 64 GB of RAM).

Experiments used MCP `ClientSession` with SSE transport. For each run, we collected the following metrics: latency (mean, min, max, median, p95, p99, standard deviation), throughput (requests/sec), error rate, resource usage (CPU and memory), and total wall-clock execution time. Although multiple metrics were recorded, the primary criterion for selecting parameter thresholds was error rate, as it generalizes well across heterogeneous MCP servers. Following MCP best practices [30], we targeted an error rate below 0.1%.

AEGIS analyzes the benchmarking results to determine parameter limits that maintain reliable performance under concurrent load. Across the 118 benchmark runs as mentioned in Figure 5, `count=10` consistently maintained error rates below 0.1% across all load levels, whereas `count=100` caused significant error spikes. Based on this analysis, AEGIS selects `count=10` as the policy threshold.

Using this threshold, we then compared server performance with and without AEGIS policy enforcement under a resource abuse scenario. Specifically, we simulated an abuse scenario using `count=1000` under a 200-concurrent-user load (1 request per user) and measured the impact on key resources in both unprotected and protected configurations as shown in Table III. At this high count value, AEGIS effectively maintained server's availability, eliminating errors and improving overall request throughput.

VIII. CONCLUSION

In this study, we leveraged LLMs to address resource abuse across MCP servers and modalities, achieving over 90% accuracy in classifying modality, operation, resource usage, and parameters. Parameter normalization remains challenging, highlighting an important area for improvement as LLM capabilities evolve. Our evaluation serves as a proof of concept for AEGIS in enabling end-to-end protection, though it is limited in scope and coverage. Future work will extend to real-world MCP services and incorporate cascading tool invocations for more comprehensive protection.

REFERENCES

[1] Model Context Protocol, "What is the Model Context Protocol (MCP)?" 2024. [Online]. Available: https://modelcontextprotocol.io/docs

[2] Enkrypt AI, "We Scanned 1,000 MCP Servers: 33% Had Critical Vulnerabilities," 2025. [Online]. Available: https://enkryptai.com/blog/we-scanned-1-000-mcp-servers-33-had-critical-vulnerabilities

[3] Z. Wang, Q. Chang, H. Patel, S. Biju, C.-E. Wu, Q. Liu, A. Ding, A. Rezazadeh, A. Shah, Y. Bao *et al.*, "MCP-Bench: Benchmarking Tool-Using LLM Agents with Complex Real-World Tasks via MCP Servers," *arXiv preprint arXiv:2508.20453*, 2025. [Online]. Available: https://doi.org/10.48550/arXiv.2508.20453

[4] V. S. Narajala and I. Habler, "Enterprise-grade security for the model context protocol (MCP): Frameworks and mitigation strategies," in *2026 IEEE 5th International Conference on AI in Cybersecurity (ICAIC)*. IEEE, 2026, pp. 1–8. [Online]. Available: https://ieeexplore.ieee.org/document/11395723

[5] W. Xing, Z. Qi, Y. Qin, Y. Li, C. Chang, J. Yu, C. Lin, Z. Xie, and M. Han, "MCP-Guard: A Defense Framework for Model Context Protocol Integrity in Large Language Model Applications," *arXiv preprint arXiv:2508.10991*, 2025. [Online]. Available: https://doi.org/10.48550/arXiv.2508.10991

[6] Open Policy Agent, "Open Policy Agent (OPA) Documentation," 2016. [Online]. Available: https://www.openpolicyagent.org/docs

[7] J. W. Cutler, C. Disselkoen, A. Eline, S. He, K. Headley, M. Hicks, K. Hietala, E. Ioannidis, J. Kastner, A. Mamat *et al.*, "Cedar: A new language for expressive, fast, safe, and analyzable authorization," *Proceedings of the ACM on Programming Languages*, vol. 8, no. OOPSLA1, pp. 670–697, 2024. [Online]. Available: https://doi.org/10.1145/3649835

[8] IBM, "ContextForge: Model Context Protocol gateway & proxy - unify REST, MCP, and A2A with federation, virtual servers, retries, security, and an optional admin UI," 2025. [Online]. Available: https://github.com/IBM/mcp-context-forge

[9] TrueFoundry, "MCP Gateway - Secure & Unified Access to MCP Servers." [Online]. Available: https://www.truefoundry.com/mcp-gateway

[10] Y. Guo, P. Liu, W. Ma, Z. Deng, X. Zhu, P. Di, X. Xiao, and S. Wen, "Systematic Analysis of MCP Security," *arXiv preprint arXiv:2508.12538*, 2025. [Online]. Available: https://doi.org/10.48550/arXiv.2508.12538

[11] S. Gaire, S. Gyawali, S. Mishra, S. Niroula, D. Thakur, and U. Yadav, "Systematization of Knowledge: Security and Safety in the Model Context Protocol Ecosystem," *arXiv preprint arXiv:2512.08290*, 2025. [Online]. Available: https://doi.org/10.48550/arXiv.2512.08290

[12] OWASP Foundation: OWASP GenAI Security Project, "Agentic AI Threats and Mitigations," 2025. [Online]. Available: https://genai.owasp.org/resource/agentic-ai-threats-and-mitigations/

[13] K. Zhou, Y. Zheng, Y. He, M. Xue, X. Gong, Y. Wang, X. Zhang, and K.-Y. Lam, "Beyond Max Tokens: Stealthy Resource Amplification via Tool Calling Chains in LLM Agents," *arXiv preprint arXiv:2601.10955*, 2026. [Online]. Available: https://doi.org/10.48550/arXiv.2601.10955

[14] Y. Lee, J. Jang, S. Choi, S. Kim, and S. Choi, "Overthinking Loops in Agents: A Structural Risk via MCP Tools," *arXiv preprint arXiv:2602.14798*, 2026. [Online]. Available: https://doi.org/10.48550/arXiv.2602.14798

[15] Y. Zhang, W. Wang, Z. Zhou, K. Wang, J. Zhang, L. Sun, Y. Liu, and S. Su, "LeechHijack: Covert Computational Resource Exploitation in Intelligent Agent Systems," *arXiv preprint arXiv:2512.02321*, 2025. [Online]. Available: https://doi.org/10.48550/arXiv.2512.02321

[16] Z. Li, B. Ma, X. Dai, M. Xu, Y. Zhang, B. Yan, and K. Li, "Don't believe everything you read: Understanding and Measuring MCP Behavior under Misleading Tool Descriptions," *arXiv preprint arXiv:2602.03580*, 2026. [Online]. Available: https://doi.org/10.48550/arXiv.2602.03580

[17] M. M. Hasan, H. Li, E. Fallahzadeh, G. K. Rajbahadur, B. Adams, and A. E. Hassan, "Model Context Protocol (MCP) at First Glance: Studying the Security and Maintainability of MCP Servers," *arXiv preprint arXiv:2506.13538*, 2025. [Online]. Available: https://doi.org/10.48550/arXiv.2506.13538

[18] Y. Yang, C. Gao, D. Wu, Y. Chen, Y. Li, and S. Wang, "MCPSecBench: A Systematic Security Benchmark and Playground for Testing Model Context Protocols," *arXiv preprint arXiv:2508.13220*, 2025. [Online]. Available: https://doi.org/10.48550/arXiv.2508.13220

[19] Z. Wang, Y. Gao, Y. Wang, S. Liu, H. Sun, H. Cheng, G. Shi, H. Du, and X. Li, "MCPTox: A Benchmark for Tool Poisoning on Real-World MCP Servers," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 40, no. 42, 2026, pp. 35 811–35 819. [Online]. Available: https://doi.org/10.1609/aaai.v40i42.40895

[20] S. Kumar, A. Girdhar, R. Patil, and D. Tripathi, "MCP Guardian: A security-first layer for safeguarding mcp-based ai system," *arXiv preprint arXiv:2504.12757*, 2025. [Online]. Available: https://doi.org/10.48550/arXiv.2504.12757

[21] T. Siameh, A. A. Addobea, and C.-H. Liu, "Context injection vulnerabilities and resource exploitation attacks in model context protocol," *Authorea Preprints*, 2025. [Online]. Available: https://doi.org/10.36227/techrxiv.175321790.02502754/v1

[22] Z. Wang, J. Zhang, G. Shi, H. Cheng, Y. Yao, K. Guo, H. Du, and X.-Y. Li, "MindGuard: Tracking, detecting, and attributing mcp tool poisoning attack via decision dependence graph," *arXiv preprint arXiv:2508.20412*, 2025. [Online]. Available: https://doi.org/10.48550/arXiv.2508.20412

[23] M. Bhatt, V. S. Narajala, and I. Habler, "ETDI: Mitigating tool squatting and rug pull attacks in model context protocol (MCP) by using oauth-enhanced tool definitions and policy-based access control," in *2025 Cyber Awareness and Research Symposium (CARS)*. IEEE, 2025, pp. 1–6. [Online]. Available: https://ieeexplore.ieee.org/document/11337310

[24] H. Jing, H. Li, W. Hu, Q. Hu, X. Heli, T. Chu, P. Hu, and Y. Song, "MCIP: Protecting MCP safety via model contextual integrity protocol," in *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, 2025, pp. 1177–1194. [Online]. Available: https://aclanthology.org/2025.emnlp-main.62/

[25] S. Jamshidi, K. W. Nafi, A. M. Dakhel, F. Khomh, A. Nikanjam, and M. A. Hamdaqa, "Secure Tool Manifest and Digital Signing Solution for Verifiable MCP and LLM Pipelines," *arXiv preprint arXiv:2601.23132*, 2026. [Online]. Available: https://doi.org/10.48550/arXiv.2601.23132

[26] H. Errico, J. Ngiam, and S. Sojan, "Securing the Model Context Protocol (MCP): Risks, Controls, and Governance," *arXiv preprint arXiv:2511.20920*, 2025. [Online]. Available: https://doi.org/10.48550/arXiv.2511.20920

[27] Anthropic, "Claude Sonnet Models Announcements," 2025. [Online]. Available: https://www.anthropic.com/claude/sonnet

[28] OpenTools, "Opentools mcp server registry," 2025. [Online]. Available: https://opentools.com/registry

[29] A. Krizhevsky and G. Hinton, "The CIFAR-10 dataset," 2009. [Online]. Available: https://www.cs.toronto.edu/ kriz/cifar.html

[30] Model Context Protocol, "MCP Best Practices: Architecture & Implementation Guide," 2024. [Online]. Available: https://modelcontextprotocol.info/docs/best-practices/