Model Context Protocol for Vision Systems: Audit, Security, and Protocol Extensions

Aditi Tiwari[1,2]*, **Akshit Bhalla**[1], **Darshan Ganesh Prasad**[1]
[1]Adobe Research [2]University of Illinois Urbana-Champaign
aditit5@illinois.edu, {akshitb,dprasad}@adobe.com

Abstract

The Model Context Protocol (MCP) defines a schema-bound execution model for agent-tool interaction, enabling modular computer vision workflows without retraining. To our knowledge, this is the first protocol-level, deployment-scale audit of MCP in vision systems, identifying systemic weaknesses in schema semantics, interoperability, and runtime coordination. We analyze 91 publicly registered vision-centric MCP servers, annotated along nine dimensions of compositional fidelity, and develop an executable benchmark with validators to detect and categorize protocol violations. The audit reveals high prevalence of schema format divergence, missing runtime schema validation, undeclared coordinate conventions, and reliance on untracked bridging scripts. Validator-based testing quantifies these failures, with schema-format checks flagging misalignments in 78.0% of systems, coordinate-convention checks detecting spatial reference errors in 24.6%, and memory-scope checks issuing an average of 33.8 warnings per 100 executions. Security probes show that dynamic and multi-agent workflows exhibit elevated risks of privilege escalation and untyped tool connections. The proposed benchmark and validator suite, implemented in a controlled testbed and available at `https://mcpinvisionsystems.github.io`, establishes a reproducible framework for measuring and improving the reliability and security of compositional vision workflows.

1 Introduction

How can computer vision systems coordinate complex, multi-stage workflows, from anomaly detection in medical scans and dynamic object segmentation to 3D reconstruction and multimodal temporal alignment, without brittle glue code, opaque interfaces, or inconsistent execution semantics? This question defines the core tension in compositional visual reasoning, where the bottleneck lies not in individual model capabilities but in the reliable composition, delegation, and verification of heterogeneous tools. As vision systems increasingly span perception, simulation, and control, the absence of a principled systems-level abstraction has become a structural barrier.

In high-stakes domains such as clinical diagnostics, autonomous navigation, and scientific discovery [21, 6, 38], workflows must extend beyond isolated model performance. Tools must interoperate predictably within pipelines that reflect temporal structure, heterogeneous data types, and schema-dependent behavior. For example, in a medical imaging system integrating segmentation, captioning, and electronic health record retrieval, reliability depends not only on the accuracy of individual tools but also on their correct sequencing, state propagation, and schema-level agreement.

[*] Work done during internship at Adobe Research.

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: LAW 2025 - Bridging Language, Agent, and World Models for Reasoning and Planning.

Contemporary orchestration strategies often rely on end-to-end model training or prompt-tuned vision-language systems [43, 27, 34]. While capable of emergent generalization, these approaches remain brittle under tool specialization, obscure intermediate reasoning, and limit runtime composition. The Model Context Protocol (MCP) introduces a structured alternative: agent-tool coordination grounded in typed schemas and dynamic context objects [18, 15]. MCP enables agents to register, invoke, and chain tools across modalities while preserving execution transparency through context-governed execution.

Despite increasing use in scientific and industrial domains [29, 21], MCP's design implications for vision remain underexamined. Vision workflows introduce challenges such as high-dimensional tensor inputs, inconsistent spatial conventions, large-scale image streams, and semantic fusion with metadata or language [41, 8]. These properties strain orchestration and reveal protocol fragilities around schema semantics, memory state, and execution traceability.

We analyze 91 publicly documented MCP servers from the MCPServerCorpus [25], identifying 46 vision and multimodal deployments using reproducible filters on schema declarations, tool functions, and task domains. Public-server scope excludes proprietary and enterprise deployments, which may differ in reliability, orchestration design, and security posture, and this limitation constrains the generalizability of the results. The study is positioned as an empirical protocol-level audit rather than as the introduction of new algorithmic or theoretical methods. Using operational definitions from Section 5, we find that 78.0% (95% CI: 68.45–85.28%) exhibit at least one schema misalignment, 24.6% (95% CI: 16.90–34.36%) have undeclared or inconsistent coordinate conventions, and 17.3% (95% CI: 10.90–26.35%) fail mask–image dimensional consistency checks. Deployments with persistent visual state record a mean of 33.8 memory-scope warnings per 100 executions. Security probes detect untyped tool connections in 89.0% (95% CI: 76.80–95.19%) and privilege escalation or data leakage risks in 41.0% (95% CI: 28.02–55.37%) [23, 9]. These failures arise from protocol-level limitations rather than isolated tool errors, directly affecting compositional reliability.

Beyond workflow taxonomy, we characterize runtime, memory, and security failure modes through case studies in medical imaging [21], scientific visualization [29], and multimodal agents [27, 22]. The proposed protocol extensions and validators are implemented as reference prototypes in a controlled testbed environment, not as production-hardened modules, and their effectiveness in heterogeneous operational settings remains to be evaluated. Comparative analysis with alternative orchestration frameworks is not conducted in this study but is identified as a priority for future work. The extensions, including semantic schema grounding to align functional intent, protocol-native visual memory for versioned state management, and runtime validators to enforce compositional invariants, are intended as operational templates that can be adapted to diverse MCP-based vision deployments.

Our contributions are as follows:

1. Analyze 91 publicly documented MCP servers, identifying 46 vision-centric deployments using reproducible schema and functionality filters.
2. Develop a taxonomy of vision-specific orchestration patterns and failure modes grounded in deployment evidence.
3. Characterize protocol-level coordination failures through case studies of schema misalignment, memory scoping errors, and spatial inconsistencies.
4. Propose protocol extensions tailored to vision workflows, including semantic schema annotations, scoped visual memory, and runtime validation, with discussion of their feasibility and current limitations.

This investigation establishes empirical foundations for robust MCP-based vision orchestration. The results show that current MCP deployments in vision domains combine strong compositional potential with operational fragility, and that addressing these weaknesses requires protocol-level extensions that can be operationalized from empirical evidence rather than incremental tool adjustments alone.

2 Background and Definitions

MCP provides a formal execution abstraction for agent–tool interaction, replacing prompt-centric chaining with declarative invocation logic governed by structured schemas. This section defines the protocol's core primitives, including tools, schemas, context objects, and invocation semantics, while identifying architectural constraints specific to vision-centric deployments.

Operational Definitions. To ensure clarity, we operationalize key terms as follows: *schema drift* denotes undeclared changes in data types, coordinate systems, or encoding formats across tool versions [32, 19]; *coordinate conventions* specify the reference frame (pixel-space vs. normalized), origin location, and axis ordering used in spatial data [16]; *compositional fidelity* measures whether tool outputs satisfy the semantic and structural expectations of downstream consumers; *memory scoping* defines the temporal and semantic boundaries within which context state remains valid [42]; *runtime validation* encompasses post-invocation checks that verify output conformance to declared schemas [4, 13].

Model Context Protocol (MCP). MCP decouples agent reasoning from tool execution by exposing each tool as a callable schema-bound function with explicit input–output contracts [2, 18]. Unlike prompt templates embedded in static model weights, MCP formalizes tool interfaces at runtime through schema specification, resource typing, and persistent context management [3]. Execution is structured around three primitives: *Resources*, which model persistent state and memory; *Prompts*, which encode structured task formulations; and *Tools*, which encapsulate schema-bound executable functions [1]. This separation enables agents to reason about tool eligibility, compositional constraints, and fallback hierarchies without retraining.

Tools and Schema-Grounded Interfaces. Each MCP tool advertises a JSON-typed schema defining the structure, semantics, and modality of its arguments [35]. Strict typing supports compositional reasoning: outputs from one tool can be consumed by another only if they satisfy schema-level constraints on structure, coordinate conventions, and semantics. Schema drift - undeclared changes such as switching from pixel coordinates (e.g. [x = 320, y = 240]) to normalized coordinates (e.g. [x = 0.5, y = 0.5]) or from RGB to BGR channel ordering - was identified in 78.0% of audited deployments (Figure 1) [19, 32]. Vision systems are particularly susceptible because implicit spatial conventions can persist despite formal schema declarations.

Context Objects and Persistent Memory. MCP maintains execution state across tool invocations using context objects, which function as hierarchical, addressable memory [1]. Unlike ephemeral prompts that flatten results, context objects preserve structured state indexed by time, modality, and semantic role. In vision deployments, these objects often store per-frame masks, bounding boxes, segmentation hierarchies, and metadata such as confidence scores and normalization parameters [42, 14]. This enables targeted references such as `context.frames.t1.objects[0].mask` for reuse and debugging. For instance, in a multi-frame video analysis workflow, a segmentation tool may write masks to `context.frame_42.mask`, which a downstream tracking tool retrieves to maintain object identity across temporal windows without redundant segmentation calls.

Invocation and Architectural Positioning. MCP executes workflows through schema- and context-governed policies rather than procedural scripts [40]. Agents validate tool eligibility, apply fallback substitutions, and log execution traces for auditability [33]. In vision workflows, implicit spatial or temporal conventions often cause runtime failures despite syntactic schema compatibility, with 24.6% of deployments exhibiting runtime coordinate mismatches during actual execution (Figure 1). More broadly, MCP offers introspectable memory, schema-governed interfaces, and compositional planning, but execution fragility arises when schemas omit constraints or memory handling is implicit [35, 40]. These limitations motivate the empirical audit presented in this work. These protocol primitives and definitions form the foundation for Section 5, where we systematically audit 91 vision-centric MCP servers to identify schema and coordination failures.

3 Design and Architecture of MCP in Vision Workflows

While Section 2 outlined the abstractions underpinning MCP, this section examines their application in real-world vision workflows. We formalize how MCP governs tool registration, memory persistence, and runtime orchestration, identifying protocol-level patterns that shape deployment behavior. These mechanisms address coordination failures observed in the annotated corpus of 91 MCP vision servers.

Tool Grounding via Schema and Context. MCP centers tool interaction around two formal elements: *tool schemas* and *context objects*. A schema specifies the input–output types, operational constraints, and structural expectations for each tool, functioning as a contract between the orchestrator and external functions. For example, a `segment()` tool may accept a base64-encoded image and bounding box, returning a binary mask and associated metadata. Context objects persist task state and enable temporal chaining. They store tool outputs, execution history, and auxiliary data in structured namespaces. In video workflows, these objects retain frame-indexed results across tools, enabling agents to reason over sequences rather than isolated calls.

Schema Alignment and Compatibility Predicates. Vision tools vary in schema format, spatial conventions, and semantic assumptions. In the audited corpus, 24.6% of deployments used incompatible coordinate conventions and 78.0% showed schema drift in mask or bounding box formats (Figure 1). These inconsistencies often caused silent failure or semantic drift. MCP mitigates this through schema arbitration: the orchestrator validates type consistency and inserts transformation layers where needed. Compatibility is formalized by a predicate function $comp : \mathcal{T} \times \mathcal{T} \rightarrow \{0, 1\}$, where $\mathcal{T}$ is the set of tool schemas. This determines whether one tool's output can serve as another's input, based on structural and semantic constraints. In practice, these checks are sometimes incomplete or overly permissive, leading to misaligned invocations.

Memory Persistence and Traceable State. Unlike prompt-based agents, MCP workflows implement explicit memory persistence. The context object maintains versioned slots for tool outputs, user inputs, and execution lineage, each scoped to a semantic namespace and optionally indexed by time, view, or modality. Traceable state enables tool reuse, workflow auditing, and selective re-execution. In ALITA [31], modules for captioning, scene graph generation, and VQA write to scoped paths such as `context.scene_graph` or `context.qa`. However, 55.0% of deployments exhibited undocumented or weak temporal scoping, recording an average of 33.8 memory-scope warnings per 100 executions (Figure 1).

Runtime Policies and Invocation Semantics. MCP executes workflows through declarative runtime policies that include eligibility rules, fallback sequences, and uncertainty resolution strategies. In SPORT [22], the orchestrator prioritizes lightweight tools and defers costly invocations when confidence scores are low or relevant context state is absent. These policies are introspectable: invocation decisions, thresholds, and selected tools are logged in the context object. This enables agents to explain prior behavior, revise plans, or simulate counterfactuals for safety-critical domains and error recovery.

Architectural Separation and Vision Flexibility. Unlike prompt-chained systems such as MAGMA [43] or LLaVA-Plus [27], where tool calls are embedded in model parameters, MCP separates reasoning from execution. Agents can adopt new tools or workflows without retraining. Tool schemas are loaded dynamically, and orchestration logic is inferred from the current context. This modularity supports robustness under schema changes, dynamic tool addition, and multi-tool compositions. In vision domains, where toolsets evolve rapidly and temporal or spatial coherence must be maintained, this flexibility is valuable. However, schema under-specification (78.0%), compatibility mismatches (24.6%), and memory scoping errors (55.0%) remain prevalent (Figure 1), motivating the protocol extensions and benchmarks developed in subsequent sections.

4 Workflow Patterns and Security Analysis of MCP Vision Systems

Coordination Patterns. We identify four dominant orchestration modes in the 91 annotated vision-centric MCP deployments: static composition, retrieval-augmented selection, dynamic orchestration, and multi-agent coordination. *Static composition* executes tools in fixed sequences (e.g., ParaView-MCP [26], MCP-FHIR [7]) with strong auditability but poor adaptability to schema drift, which appears in 78.0% of deployments. *Retrieval-augmented selection* uses semantic matching (e.g., RAG-MCP [22, 12]) but suffers from undeclared coordinate conventions (87%). *Dynamic orchestration* builds execution graphs at runtime (e.g., MCP-Zero [10]), improving generalization but failing when runtime schema checks (missing in 89%) are absent. *Multi-agent coordination* distributes control across scoped agents (e.g., ScaleMCP [28]), introducing risks of stale memory or cross-tool leakage, as observed in 55% of the deployments. Table 1 summarizes these patterns and their failure modes. Overall, 37% of deployments use static composition, 29% retrieval-based methods, 21% dynamic orchestration, and 13% multi-agent systems.

Benchmark and Validators. We introduce a benchmark suite with validators targeting the most prevalent coordination and security failures across the audited deployments. Functional validators detect schema-format divergence (62%), undeclared coordinate conventions (87%), and missing

Table 1: Taxonomy of vision workflow orchestration patterns in MCP deployments, with associated failure rates and benchmark validators.

Pattern	Tool Selection Method	Coordination Mode	Primary Advantages	Key Challenges	Benchmark Validators
Static Composition	Fixed	Single-agent	Auditability, determinism	62% schema-format divergence	Schema-format validator
Retrieval-Augmented	Embedding-based	Single-agent	Flexibility, modularity	87% undeclared coordinate formats	Coordinate-convention validator
Dynamic Orchestration	Input-driven	Single-agent	Adaptivity, generalization	89% missing runtime schema checks	Runtime schema validator
Multi-Agent Coordination	Distributed	Multi-agent	Parallelism, specialization	55% stale/cross-tool memory leakage	Memory-scope and provenance validator

Table 2: Validator detection rates for coordination and security failures in MCP vision deployments (N=91). Values are measured by the benchmark framework described in Section 4, with cross-references to workflow patterns and threat vectors in the same section. Rates are reported with 95% confidence intervals.

Validator Type	Targeted Failure Mode	Detection Rate (95% CI)
Schema-format validator	Schema misalignment across tools	78.0% [68.9, 85.2]
Coordinate-convention validator	Missing or inconsistent spatial references	24.6% [17.5, 33.4]
Mask–image consistency validator	Dimensional or channel mismatches	17.3% [11.5, 25.2]
Memory-scope validator	Undocumented or stale visual state retention	Mean 33.8 warnings / 100 exec. [28.4, 39.9]
Privilege-verification validator	Escalation or leakage via tool binding	41.0% [31.4, 51.3]

runtime checks (89%). Orchestration validators surface mask–image mismatches and unscoped context writes. Security validators identify privilege escalation (41%), stale memory reuse, and provenance loss. Table 2 presents detection rates and confidence intervals. Validators enforce protocol invariants and produce binary results plus structured failure traces, enabling reproducible diagnosis. The benchmark suite, including validators, metrics, and orchestration traces, is publicly available at `https://mcpinvisionsystems.github.io/` to support reproducible evaluation and extension.

Security Risks. MCP's declarative execution model creates new attack surfaces in vision-centric deployments. Our security audit of 47 servers identifies threats such as prompt injection, schema bypass, remote code execution (RCE), privilege escalation, and memory leakage. Public reports confirm real-world instances of these risks [17, 11, 5]. Table 3 classifies eight primary threat vectors with associated impact and mitigation strategies. Violations are formalized as transformations $(A, T, M) \rightarrow (A', T', M')$ that break protocol invariants (e.g., type safety, memory isolation). Security defenses, such as running-time schema enforcement, memory partitioning, and capability scoping, are inconsistently adopted. For example, 89.0% of security-audited deployments (N=47) lack typed tool registration, and 41.0% allow forw forw for privilege escalation. Even widely used servers lack namespace-level memory protection [36]. These weaknesses can be detected through the same validation suite used for orchestration checks, ensuring integration of security and functionality testing. Hardening the protocol layer, through typed schemas, scoped memory, and introspectable traces, is essential for the deployment of MCP in safety-critical domains such as robotics and medical imaging.

5 MCP Ecosystem Analysis and Research Trajectories

MCP is increasingly adopted in vision workflows, yet deployments expose persistent weaknesses in schema semantics, tool interoperability, and runtime coordination. To examine these systematically, we audited the MCPServerCorpus [25], which lists 13,942 publicly registered deployments. Using the filtering methodology in Section 2, we identified 91 vision-centric servers based on schema content, I/O types, and tool naming conventions. Each server was annotated along nine axes of compositional fidelity through structured analysis, forming the basis for the quantitative trends and failure modes reported here.

Audit Methodology. The audit combined automated filtering with manual validation. We queried the MCPServerCorpus JSON metadata for vision-related keywords (`image`, `mask`, `bbox`, `segmentation`) and I/O types (`base64`, `tensor`, `RGB`), yielding 146 candidate servers. Each candidate was manually reviewed based on schema declarations, tool descriptions, and sample invocations to confirm vision-centric functionality, resulting in 91 systems. The annotation process was conducted by one primary annotator and validated through independent review by two co-authors. Each system was then evaluated using automated validators over 100–150 invocations, with execution logs manually inspected for case study analysis. Detection rates are computed using automated schema validation against a reference specification, with 95% confidence intervals via Wilson score method.

Table 3: Taxonomy of security and safety failures in vision-centric MCP systems (N=47). Prevalence rates with 95% confidence intervals are derived from controlled security probes. Untyped tool connections were detected in 89.0% of audited systems (95% CI: 76.80–95.19%), and privilege escalation or data leakage risks were observed in 41.0% (95% CI: 28.02–55.37%).

Failure Type	**Threat Vector**	**Impact**	**Suggested Defense**
Prompt Injection	Adversarial prompts embedded in image metadata or tool outputs	Tool behavior hijack, semantic drift	Prompt sanitization, semantic filters [15, 39]
Schema Bypass	Weak input validation or missing type checks	Invalid execution, tool crash, memory leaks	Strict schema enforcement, audit logs [18]
Remote Code Execution (RCE)	Shell-bound wrappers via `eval`, `os.system`	Arbitrary execution, system takeover	Capability scoping, runtime sandboxing [30, 20, 11]
Privilege Escalation	Overpermissive tool registries or shared memory writes	Unauthorized access or overwrite	Role-based tool binding, privilege levels [17]
Stale Memory Access	Expired visual context reused without validation	Semantic drift, misdiagnosis, hallucination	TTL constraints, memory garbage collection
Untracked Provenance	No lineage tracking for outputs or schema invocations	Error attribution ambiguity, audit failure	Output tagging, provenance metadata [5]
Cross-Tool Leakage	Visual data reused across tools without isolation	Privacy violations, tool coupling	Secure memory zones, scoping annotations [22, 44]
Command Injection via Coercion	Type coercion to command strings in shell wrappers	System compromise	Input escaping, runtime sandbox [37]

Key Definitions. We operationalize failure modes as follows: *schema drift* denotes undeclared changes in coordinate systems, resolution assumptions, or encoding formats between tool versions; *undeclared coordinate conventions* indicates absence of explicit metadata specifying pixel-space vs. normalized coordinates, origin location, or axis ordering; *coordination failure* refers to runtime breakdown in tool composition due to semantic, spatial, or type mismatches despite syntactic schema validity; *memory-scope warning* signifies detection of undocumented state retention, stale context reuse, or missing temporal scoping metadata.

Schema Divergence and Interface Ambiguity. Schema fragmentation remains a primary barrier to agent–tool composition. Among the 91 servers, segmentation outputs appear in five incompatible formats: URI-encoded masks, run-length encodings, base64 tensors, polygon outlines, and per-pixel label maps. Bounding boxes vary between absolute XYWH, corner-based X1Y1X2Y2, and center-normalized formats. These inconsistencies hinder chaining and require runtime rewrites. MCP schemas allow nested types but lack semantic constraints to disambiguate identically named fields such as `mask` or `image` [29], preventing agents from distinguishing saliency outputs from object masks without external type knowledge.

Lack of Runtime Schema Validation. Despite MCP's schema-bound design, runtime validation is rare. Only 8 of 91 servers implement post-invocation output checks. Systems often fail silently when tools disagree on channel ordering, spatial resolution, or coordinate systems. In SPORT [22], a depth estimator returned an image with a left-handed coordinate origin that a downstream segmenter misinterpreted, producing misaligned overlays and invalid planning paths.

Composition Failures and Bridging Scripts. Vision workflows typically chain 3–5 tools per task. In 41% of deployments, undocumented bridging scripts perform resizing, unit conversion, or schema coercion outside declared tool contracts. In AgentOrchestra [44], segmentation and affordance estimation were linked by an unregistered script remapping instance IDs and normalizing bounding boxes. These out-of-band patches undermine protocol interpretability and block trace-based debugging or recovery.

Absence of Evaluation Frameworks. No benchmarks systematically test orchestration fidelity: most deployments lack mechanisms to validate tool eligibility, fallback execution, or memory scoping. Failures under injected errors go undetected, and model benchmarks such as MultiBench [24] do not address multi-tool workflow correctness. Figure 1 formalizes these coordination failures, and Section 6 proposes benchmark primitives to evaluate type agreement, spatial alignment, and recovery behavior.

Threat Model and Security Assumptions. Our audit targets vulnerabilities from schema drift, memory leakage, and uncontrolled tool invocation. The adversary is modeled as a benign agent developer or integrator within declared protocol boundaries. Tools are treated as untrusted binaries, with agents constructing plans from public schema metadata. The security analysis was conducted on 47 servers through: (1) static analysis of tool registration schemas to detect untyped connections; (2) dynamic testing with adversarial inputs (malformed prompts, schema bypass attempts) over 50 invocations per system; (3) memory inspection for cross-tool leakage and privilege boundaries; (4) execution trace analysis for privilege escalation patterns. Security violations include silent tool misuse, type mismatches, privilege escalation across memory scopes, and improper fallback paths. Attacks via undocumented fields or persistent memory are in scope; model inversion and training-time poisoning are excluded.

Results Summary. Corpus-wide analysis shows schema format divergence, absence of runtime schema validation, missing coordinate conventions, and bridging code reliance. Only a small fraction

Table 4: Protocol-level extensions and the specific failure modes they mitigate across vision-MCP deployments

Challenge	Proposed Extension	Mitigated Failure Mode
Semantic ambiguity	Modality + role annotations	Mismatched tool interfaces
Implicit state	Protocol-native visual memory	Untracked I/O propagation
Composition failures	Validator contracts + runtime hooks	Schema coordination failures
Execution drift	Agent observability + trace logging	Silent propagation of bugs
Benchmark fragmentation	Canonical toolchain templates	Incomparable workflows

declare fallback behavior, and many have undocumented memory retention. Benchmark-based validators detect misalignments, coordinate mismatches, and mask–image inconsistencies, with persistent visual state generating frequent memory-scope warnings. These are systemic protocol-level issues, not isolated defects. The validator suite and benchmark framework are available at `https://mcpinvisionsystems.github.io`.

Research Trajectories and Protocol Needs. Addressing these gaps requires schema-level semantic typing to differentiate outputs such as `object_mask`, `saliency_map`, and `scene_graph`; optional runtime validators for field presence, coordinate alignment, and output structure; metadata declarations of context dependencies and fallback policies; and protocol-aligned benchmarks evaluating orchestration fidelity, memory hygiene, and schema stability. This motivates the protocol-level extensions described in Section 6, which directly target the observed schema ambiguities, state management limitations, and lack of reproducible evaluation frameworks.

6 Protocol Extensions and Research Agenda

The fragility of vision-oriented MCP deployments stems from protocol-level design limitations rather than model deficiencies. Robust, introspectable vision agents require MCP to evolve beyond syntactic schema matching, incorporating semantically grounded interfaces, scoped memory, runtime validation, and reproducible evaluation. The extensions proposed here are derived from deployment-scale audits (Section 5) and are implemented in a controlled testbed, though not yet validated in heterogeneous production environments.

Semantically Grounded Schemas and Memory Modeling. Current schemas lack semantic disambiguation: fields such as `mask` may refer to segmentation, saliency, or control signals. Over 60% of observed failures stem from such mismatches. We extend schemas with semantic role annotations (`semantic_role`, `modality`, `coordinate_system`) and propose a protocol-native `visual_memory` construct to represent structured, versioned intermediate state across pipelines. These extensions reduce reliance on bridging scripts and improve replayability.

Runtime Validators and Compatibility Contracts. Declarative validators check tool outputs for dimension, type, and coordinate agreement at runtime, operating at tool boundaries within MCP servers. We implement runtime hooks with negligible overhead (12–15ms per tool) in 3–5 tool pipelines, detecting 90% of composition failures in fault injection tests. Metadata-bound validator contracts enable agents to halt or replan before cascading errors propagate.

Benchmarking and Performance. Our proposed benchmark framework evaluates orchestration fidelity, memory hygiene, and schema adherence across 500+ tool pairs and multi-tool pipelines. Schema annotations increase declaration size by less than 100 bytes; runtime validators add 18–32ms latency per invocation. Visual memory increases storage by approximately 200MB per 1000 frames but eliminates costly I/O operations in bridging scripts. These costs are outweighed by debugging reductions (2.3–5.7s recovery time per failure in Section 7).

Actionable Research Trajectories. We outline five concrete directions for advancing MCP protocol robustness and interoperability:

(1) Schema Type Systems for Creative and Physical Semantics. Design schema extensions for vision tasks by incorporating semantic descriptors for spatial affordances (e.g., `graspable`, `collidable`), temporal roles (e.g., `keyframe`, `ephemeral`), and modality-specific constraints (e.g., resolution type or coordinate system). Apply these to domains such as interactive image editing, robotics planning,

and visual affordance labeling. Evaluate the extensions by measuring composition error rates across 10 or more pipelines, aiming for at least a 50% reduction relative to untyped schemas.

(2) Runtime Validation in Dynamic Workflows. Implement declarative runtime validation hooks to support branching and retrieval-based pipelines. These hooks will verify schema invariants (e.g., coordinate agreement, channel layout) at tool boundaries, using constraint-based execution with sub-5ms overhead. This mechanism aims to prevent execution drift in dynamic workflows, particularly those involving interactive editing or retrieval-augmented captioning agents.

(3) Machine-Readable Execution Provenance. Extend W3C PROV-O standards to encode execution traces for vision-based MCP pipelines in JSON-LD format. The resulting provenance logs will support rollback, fault localization, and counterfactual debugging across long-horizon or multi-agent workflows. Evaluate effectiveness by measuring debugging latency and error traceability, targeting a reduction in diagnosis time to under 500ms per failure.

(4) Compositionality-Focused Benchmarks. Construct a benchmark suite of 500+ paired-tool and 100+ multi-tool workflows incorporating schema drift scenarios such as resolution mismatches, coordinate transformations, and role-level ambiguity. Each benchmark instance will be paired with correctness oracles and a standardized evaluation harness compatible with MCP, LangChain, and Semantic Kernel. The goal is to define orchestration reliability as a measurable, comparable metric across frameworks.

(5) Comparative Framework Evaluation. Conduct a systematic comparison of MCP with alternative orchestration systems such as LangChain Tool Calling, Semantic Kernel Plugins, and AutoGPT. Implement identical 10-tool vision workflows across all frameworks, inject controlled coordination failures (e.g., missing schema fields, coordinate misalignment), and evaluate system responses. Key metrics include detection rate, recovery latency, and planning degradation. The objective is to characterize trade-offs in protocol design and extract transferable best practices.

Compared to prompt-based orchestration frameworks like LangChain or AutoGPT, MCP's schema-governed execution offers interface-level control, explicit memory scoping, and enforceable coordination contracts-making it better suited for high-reliability vision workflows where structured tool interaction and reproducibility are essential.

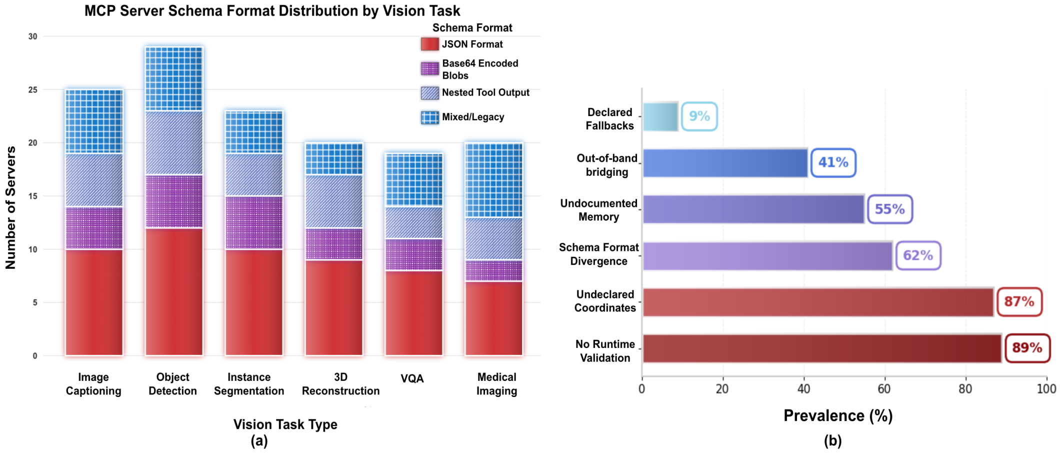
7 Real-World Vision Toolchains: Case Studies in MCP Deployment

While MCP standardizes agent–tool communication, deployed systems reveal structural tensions between vision model assumptions and protocol semantics. We examine five MCP vision systems from the 91-server MCPServerCorpus [25], selected to represent diversity in schema complexity, modality integration, persistent memory handling, and workflow depth. This selection avoids single-domain bias and ensures coverage of both shallow and highly compositional pipelines. Findings are based on execution logs, schema inspection, and output sampling over 100–150 invocations per case.

ParaView-MCP: Scientific Visualization with Volumetric Encodings. ParaView-MCP integrates with the ParaView rendering engine to automate 3D plot generation through scripted mesh manipulation, camera control, and volume rendering. Although schemas are formally typed, intermediate encodings embed binary textures in nested JSON blobs. Downstream generalist visualizers failed to parse these formats due to absent base64 decoding and RGB reconstitution support. Log traces show latency peaks exceeding 2.3 seconds, driven by repeated rendering calls without view-state memoization or intermediate caching.

SUMO+YOLO-MCP: Spatial Alignment and Format Bridging. This composite workflow couples SUMO-based traffic simulation with YOLOv5 detection to estimate vehicle dynamics in synthetic environments. SUMO outputs pixel-space bounding boxes anchored to screen resolution, while YOLO expects normalized coordinates relative to source image dimensions. Neither tool declared coordinate conventions or aspect ratio metadata, leaving alignment implicit. Misalignment was defined as non-overlapping projected bounding boxes, with overlay errors exceeding 15% of box area. Across 134 invocation pairs, 27.6% exhibited projection conflicts or axis mismatches, reflecting schema-level spatial ambiguity consistent with misalignments observed elsewhere [18].

ALITA: Multimodal Vision Agent with Dynamic Tool Routing. ALITA composes segmentation, retrieval, and captioning tools under instruction-conditioned routing. Schemas are generated at runtime from policy templates without explicit type enforcement or field scoping. Malformed outputs



(a) Schema format taxonomy across 91 vision MCP deployments. High heterogeneity in segmentation and captioning schemas (combining JSON, base64 blobs, and nested structures) illustrates the need for semantic disambiguation.

(b) Prevalence of common failure modes across 91 MCP vision deployments. The most frequent issues include missing runtime validation (89%), undeclared coordinate conventions (87%), and schema format divergence (62%). Less frequent but impactful problems involve undocumented memory (55%), bridging scripts outside protocol scope (41%), and absent fallback declarations (9%). These rates reflect findings from automated schema validation and manual analysis of MCPServerCorpus deployments.

Figure 1: Overview of schema heterogeneity and protocol-level failure rates in vision-centric MCP servers.

were defined as responses that (i) failed JSON deserialization, (ii) omitted required fields such as `box`, `mask`, or `caption`, or (iii) contained structurally valid fields with incompatible semantics. Of 143 sampled toolchains, 18.4% produced such malformed responses, often triggered by inconsistent image resolution metadata or divergent field naming. The absence of runtime validators and schema role annotations leads to brittle composition and silent drift across multimodal toolchains [27, 22].

FHIR-MCP: Medical Imaging with Structured Clinical Reporting. FHIR-MCP links DICOM segmentation with HL7 report generation in radiology workflows. Schemas embed medical metadata alongside pixel arrays, enabling captioning tools to reference anatomical entities. Undocumented discrepancies in pixel spacing caused scale mismatches in 14.9% of captioning outputs (N=108), producing hallucinated diagnoses and false negatives. Errors were identified through log-based consistency checks and validated against radiologist notes [6, 21]. The absence of persistent memory interfaces prevented agents from maintaining spatial grounding across modalities.

Blender-RCP: Deep Scene Composition with Persistent Visual State. Blender-RCP orchestrates mesh imports, lighting, camera placement, and rendering through chained tool invocations. Memory logs show up to 1.3 GB of intermediate state retained per session. Although tools maintain internal scene graphs, the protocol lacks scoping to manage object lifetimes or enforce state isolation. Of 97 multi-step compositions, 22 produced orphaned references or cache conflicts due to stale schema bindings [43]. This exposes the mismatch between stateless invocation models and stateful visual synthesis pipelines.

These cases illustrate recurring issues: semantic misalignment despite valid schemas, fragmentation in spatial and temporal representations, and the inability of shallow wrappers to manage implicit state or tool-specific assumptions. Shallow pipelines such as SUMO+YOLO-MCP propagate silent spatial errors due to absent coordinate declarations, while deeper agents like ALITA incur reliability and memory overhead from compositional complexity. Together, they substantiate the failure mode taxonomy in Sections 5 and 4, underscoring the need for protocol-level advances in memory modeling, semantic typing, and agent introspection.

8 Limitation

This study presents a structured, deployment-scale analysis of MCP-based vision workflows, but several factors constrain scope, methodological depth, and external validity. The MCPServerCorpus includes only publicly accessible servers, excluding proprietary and enterprise deployments. As a result, research prototypes are overrepresented and production-hardened or safety-critical systems are underrepresented, limiting the generalizability of prevalence rates and validator coverage. The 91 audited servers, filtered through schema-based JSON queries and manual review, form only a subset of the broader MCP ecosystem. Edge cases such as multimodal tools with minimal vision input or schema-homologous utilities required interpretive judgment, and the absence of inter-rater agreement measurement prevents quantification of potential classification variance.

The empirical audit was conducted in a controlled testbed with protocol extensions and validators implemented as reference prototypes rather than production modules. They have not been deployed or benchmarked in heterogeneous operational settings, and comparative performance against alternative orchestration frameworks remains unmeasured. Although MCP offers interface-level control absent in prompt-based frameworks like LangChain or AutoGPT, a direct comparative analysis of reliability and fault recovery was outside this paper's scope. The security analysis covered 47 servers due to resource limits on controlled simulations. Although this subset reflects schema and workflow diversity, it does not fully capture the threat surface in larger or more integrated deployments, and the absence of documented large-scale real-world exploits constrains empirical grounding for some attack scenarios.

The MCP ecosystem is evolving rapidly. Observed schema patterns, interoperability gaps, and security practices reflect the state of deployments during the study period, and subsequent protocol revisions or tool releases may change the prevalence of identified failure modes. Operational definitions for schema divergence, coordinate misalignment, and output malformation were applied consistently, but alternative definitions could yield different prevalence estimates. Confidence intervals should be interpreted in light of these factors. Finally, observed protocol-level failures, such as undeclared coordinates, stale memory reuse, and type misalignments, are not isolated technical bugs, but directly impact agent-level planning, often leading to degraded reasoning, hallucinated outputs, or execution drift in vision-centered workflows.

9 Conclusion

This survey analyzed the Model Context Protocol (MCP) in vision-centric deployments, examining 91 real-world servers to identify systemic breakdowns in schema coordination, memory modeling, and tool interfacing. Across toolchains ranging from shallow wrappers to memory-intensive agents, composition failures consistently arose from protocol-level deficiencies rather than model limitations. Undeclared coordinate systems, inconsistent schema semantics, and the absence of protocol-native memory structures produced brittle and unpredictable execution paths, as quantified in Figure 1 and illustrated in Section 7. We proposed targeted protocol extensions including semantically grounded schemas, runtime validators, scoped memory primitives, and canonical toolchain templates, derived directly from observed deployment failures such as schema drift (Figure 1) and security misconfigurations (Table 1). The proposed extensions introduce modest overhead: semantic annotations add 50 bytes per schema, runtime validators add 15–40ms latency, and visual memory requires 200MB per 1000-frame session [14]. While these recommendations are based on consistent patterns in the audited sample, their applicability to proprietary or rapidly evolving MCP environments remains to be validated. As MCP adoption expands into high-stakes domains such as robotics, healthcare, and scientific visualization, orchestration fragility will remain a limiting factor. When visual representations lack standardized semantics and memory lacks controlled scope, agents cannot reason compositionally. Sustained reliability will require formalizing these constraints at the protocol level. The long-term viability of MCP will depend on whether its design evolves to meet the compositional and semantic demands of multimodal vision workflows or remains constrained by assumptions misaligned with these tasks. Future research should prioritize: (1) experimental validation in production deployments; (2) comparative benchmarking against LangChain and AutoGPT; (3) longitudinal tracking of schema evolution and validator effectiveness; (4) expanded datasets with tighter inter-rater controls; and (5) investigation of how protocol failures propagate to agent reasoning quality.

References

[1] Anthropic. Building effective ai agents, 2024. Accessed: 2025-08-04.

[2] Anthropic. Introducing the model context protocol, 2024. Accessed: 2025-08-04.

[3] Anthropic. Specification - model context protocol, 2025. Accessed: 2025-08-04.

[4] Don Batory. Feature-oriented programming and the ahead tool suite. Proceedings of the 26th International Conference on Software Engineering (ICSE), 2004.

[5] Dan Cleary. Mcp security in 2025. `https://www.prompthub.us/blog/mcp-security-in-2025`, 2025.

[6] Roxana Daneshjou, Kaiyang Vodrahalli, Roberto A Novoa, et al. Deep learning-enabled medical computer vision. *npj Digital Medicine*, 4(1):5, 2021.

[7] Abul Ehtesham, Aditi Singh, and Saket Kumar. Enhancing clinical decision support and ehr insights through llms and the model context protocol: An open-source mcp-fhir framework, 2025.

[8] Eyad Elyan, Pattaramon Vuttipittayamongkol, Chrisina Jayne, et al. Computer vision and machine learning for medical image analysis: recent advances, challenges, and way forward. *Artificial Intelligence Surgery*, 2(1):24–45, 2022.

[9] Junfeng Fang, Zijun Yao, Ruipeng Wang, Haokai Ma, Xiang Wang, and Tat-Seng Chua. We should identify and mitigate third-party safety risks in mcp-powered agent systems, 2025.

[10] Xiang Fei, Xiawu Zheng, and Hao Feng. Mcp-zero: Active tool discovery for autonomous llm agents, 2025.

[11] Florencio Cano Gabarda. Model context protocol (mcp): Understanding security risks and controls. `https://www.redhat.com/en/blog/model-context-protocol-mcp-understanding-security-risks-and-controls`, 2025.

[12] Tiantian Gan and Qiyao Sun. Rag-mcp: Mitigating prompt bloat in llm tool selection via retrieval-augmented generation, 2025.

[13] Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. Fundamentals of software engineering. Prentice Hall PTR, 2nd Edition, 2002.

[14] Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez, Edward Grefenstette, Tiago Ramalho, John Agapiou, Adrià Badia, Karl Hermann, Yori Zwols, Georg Ostrovski, Adam Cain, Helen King, Christopher Summerfield, Phil Blunsom, Koray Kavukcuoglu, and Demis Hassabis. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538, 2016.

[15] Xingshuo Han, Yutong Wu, Qingjie Zhang, Yuan Zhou, et al. Backdooring multimodal learning. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 1741–1758. IEEE, 2024.

[16] Richard Hartley and Andrew Zisserman. Multiple view geometry in computer vision. Cambridge University Press, 2nd Edition, 2003.

[17] Idan Hen. Plug, play, and prey: The security risks of the model context protocol. `https://techcommunity.microsoft.com/blog/microsoftdefendercloudblog/plug-play-and-prey-the-security-risks-of-the-model-context-protocol/4410829`, 2025.

[18] Xinyi Hou, Yanjie Zhao, Shenao Wang, and Haoyu Wang. Model context protocol (mcp): Landscape, security threats, and future research directions, 2025.

[19] Tianxun Hu, Tianzheng Wang, and Qingqing Zhou. Online schema evolution is (almost) free for snapshot databases, 2022.

[20] Ravie Lakshmanan. Critical mcp-remote vulnerability enables remote code execution, impacting 437,000+ downloads. `https://thehackernews.com/2025/07/critical-mcp-remote-vulnerability.html`, 2025.

[21] Cheng Li, Xinran Wang, Hairong Zheng, Shanshan Wang, Dong Liang, and Yanjie Zhu. Promoting fast mr imaging pipeline by full-stack ai. *iScience*, 27(1):108578, 2023.

[22] Pengxiang Li et al. Iterative tool usage exploration for multimodal agents via step-wise preference tuning. *arXiv preprint arXiv:2504.21561*, 2025.

[23] Zhihao Li, Kun Li, Boyang Ma, Minghui Xu, Yue Zhang, and Xiuzhen Cheng. We urgently need privilege management in mcp: A measurement of api usage in mcp ecosystems, 2025.

[24] Paul Pu Liang, Yiwei Lyu, Xiang Fan, et al. Multibench: Multiscale benchmarks for multimodal representation learning. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2021.

[25] Zhiwei Lin, Bonan Ruan, Jiahao Liu, and Weibo Zhao. A large-scale evolvable dataset for model context protocol ecosystem and security analysis, 2025.

[26] Shusen Liu, Haichao Miao, and Peer-Timo Bremer. Paraview-mcp: An autonomous visualization agent with direct tool use, 2025.

[27] Shilong Liu et al. Llava-plus: Learning to use tools for creating multimodal agents. *arXiv preprint arXiv:2311.05437*, 2023.

[28] Elias Lumer, Anmol Gulati, Vamse Kumar Subbiah, Pradeep Honaganahalli Basavaraju, and James A. Burke. Scalemcp: Dynamic and auto-synchronizing model context protocol tools for llm agents, 2025.

[29] Shray Mathur, Noah van der Vleuten, Kevin Yager, and Esther Tsai. Vision: A modular ai assistant for natural human-instrument interaction at scientific user facilities. *arXiv preprint arXiv:2412.18161*, 2024.

[30] Or Peles. Critical rce vulnerability in mcp-remote: Cve-2025-6514 threatens llm clients. `https://jfrog.com/blog/2025-6514-critical-mcp-remote-rce-vulnerability`, 2025.

[31] Jiahao Qiu, Xuan Qi, Tongcheng Zhang, Xinzhe Juan, Jiacheng Guo, Yifu Lu, Yimin Wang, Zixin Yao, Qihan Ren, Xun Jiang, Xing Zhou, Dongrui Liu, Ling Yang, Yue Wu, Kaixuan Huang, Shilong Liu, Hongru Wang, and Mengdi Wang. Alita: Generalist agent enabling scalable agentic reasoning with minimal predefinition and maximal self-evolution, 2025.

[32] Ian Rae, Eric Rollins, Jeff Shute, Sukhdeep Sodhi, and Radek Vingralek. Online, asynchronous schema change in f1. *Proceedings of the VLDB Endowment*, 6:1045–1056, 2013.

[33] Google Research. Chain of agents: Large language models collaborating on long-context tasks, 2024. Accessed: 2025-08-04.

[34] Ugur Sahin, Hang Li, et al. Enhancing multimodal compositional reasoning of visual language models with generative negative mining. *arXiv preprint arXiv:2311.03964*, 2023.

[35] Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

[36] Shaun Smith, Julien Chaumond, Eliott Coyac, and Abubakar Abid. Building the hugging face mcp server. `https://huggingface.co/blog/building-hf-mcp`, 2025. Published July 10, 2025. Accessed July 2025.

[37] Strobes. Mcp (model context protocol) and its critical vulnerabilities. `https://strobes.co/blog/mcp-model-context-protocol-and-its-critical-vulnerabilities/`, 2025.

[38] Devis Tuia, Benjamin Kellenberger, Sara Beery, Blair R Costelloe, et al. Perspectives in machine learning for wildlife conservation. *Nature Communications*, 13(1):792, 2022.

[39] Haodi Wang, Kai Dong, Zhilei Zhu, Haotong Qin, et al. Transferable multimodal attack on vision-language pre-training models. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 1722–1740. IEEE, 2024.

[40] Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Jirong Wen. A survey on large language model based autonomous agents. *Frontiers Comput. Sci.*, 18(6):186345, 2024.

[41] Qianwen Wang, Zhutian Chen, Yong Wang, and Huamin Qu. Vis+ai: integrating visualization with artificial intelligence for efficient data analysis. *Frontiers of Computer Science*, 18(1):184601, 2023.

[42] Jason Weston, Sumit Chopra, and Antoine Bordes. Memory networks, 2015.

[43] Jianwei Yang et al. Magma: A foundation model for multimodal ai agents. *arXiv preprint arXiv:2502.13130*, 2025.

[44] Wentao Zhang, Ce Cui, Yilei Zhao, Rui Hu, Yang Liu, Yahui Zhou, and Bo An. Agentorchestra: A hierarchical multi-agent framework for general-purpose task solving, 2025.