

MCP-MedImg: Enabling Standardized PACS-AI Integration through the Model Context Protocol

xiong shi

bailianfa@gmail.com

Wuhan University <https://orcid.org/0009-0006-2745-491X>

Research Article

Keywords: Model Context Protocol, Medical Imaging, PACS Integration, AI-Assisted Diagnosis, DICOM, Total Product Life Cycle, Regulatory Compliance, Service-Oriented Architecture, Conditional Verification, Interoperability

Posted Date: October 14th, 2026

DOI: <https://doi.org/10.21203/rs.3.rs-11357800/v1>

License:   This work is licensed under a Creative Commons Attribution 4.0 International License.

[Read Full License](#)

Additional Declarations: The authors declare no competing interests.

MCP-MedImg: Enabling Standardized PACS-AI Integration through the Model Context Protocol

Xiong Shi

Independent Researcher

ORCID: 0009-0006-2745-491X

Code: github.com/Xiong-Shi/MCP-MedImg

Corresponding author: bailianfa@gmail.com

October 11, 2026

Abstract

Background: The proliferation of AI-assisted diagnostic models in medical imaging has created a fragmented integration landscape with PACS systems, leading to redundant development, inconsistent security compliance, and disconnected clinical workflows. Most current AI diagnostic products exist as standalone Software as a Medical Device (SaMD), limiting their scalability, auditability, and clinical adoption. Existing integration solutions—Direct Plugin, DICOM Router, VNA—are point-to-point adaptations without unified semantic layers. While the Model Context Protocol (MCP) provides tool invocation frameworks for AI agents, it lacks dedicated extensions for the medical imaging domain. Critically, global regulators—FDA, NMPA, and the EU—are converging on total product life cycle (TPLC) approaches that demand continuous post-market surveillance, yet no standardized protocol exists to support such surveillance at scale.

Objective: We propose MCP-MedImg, an open reference architecture and protocol specification designed to enable interoperable integration and governance of AI services within medical imaging ecosystems. The objective is to propose a standardized communication and service abstraction layer that allows AI capabilities to be discovered, invoked, monitored, and governed across heterogeneous healthcare environments—without requiring replacement of existing PACS infrastructures.

Methods: MCP-MedImg extends MCP with a four-layer architecture: Transport Layer (DICOMweb-compatible, with dicom-mcp upward compatibility), Security and Compliance Layer (PHI de-identification + 14-column audit trail), AI Orchestration Layer (model routing and version management), and Application Semantics Layer (inspection-driven routing + lesion encoding + conditional human-in-the-loop verification). The protocol specification defines six core information

model entities (Imaging Study, AI Service, Inference Task, AI Result, Verification Policy, Audit Event) and a tiered conformance model with five mandatory core requirements and seven optional extensions. A reference implementation based on open-source Orthanc PACS and nnU-Net validates protocol consistency, end-to-end workflow, multi-model routing, standardized DICOM SR/SEG generation, and complete audit trail functionality.

Results: The reference implementation successfully demonstrates end-to-end interoperability between medical imaging infrastructure and AI services. MCP-MedImg enabled standardized AI service registration, workflow execution, structured result integration (DICOM SR Comprehensive 3D, SOP Class 1.2.840.10008.5.1.4.1.1.88.34), and auditable tracking of AI lifecycle events. Real GPU inference benchmarks on an NVIDIA RTX 4090 D validate end-to-end latency under realistic clinical conditions: LIDC-IDRI lung nodule CT (200 volumes, mean 5.4s, P99 22.6s)¹ and MSD Heart cardiac MRI (10 test volumes, mean 18.2s, P50 19.1s, P99 27.6s, 5-fold ensemble Dice 0.9324). The validation results, comprising 132 passing tests across 10 verification scenarios (Table 16)², indicate that the proposed protocol provides a feasible foundation for managing heterogeneous medical imaging AI services.

Conclusions: MCP-MedImg aims to transform AI-assisted diagnosis from standalone software into auditable, service-oriented capabilities with conditional human-in-the-loop verification—directly addressing the regulatory surveillance gap and establishing foundations for governable public infrastructure in healthcare AI. By defining a tiered conformance model, the protocol enables incremental adoption: from minimal compliance (Level 1) through full production deployment (Level 3). The service-oriented delivery model, with built-in audit trails, configurable verification, and standardized output generation, significantly enhances clinical trust, lowers deployment barriers, and accelerates the adoption of AI-assisted diagnosis in routine clinical practice.

Keywords: Model Context Protocol; Medical Imaging; PACS Integration; AI-Assisted Diagnosis; DICOM; Total Product Life Cycle; Regulatory Compliance; Service-Oriented Architecture; Conditional Verification; Interoperability

Abbreviations

AI: Artificial Intelligence

AUC: Area Under the Curve

¹The LIDC benchmark used a randomly initialized 3D U-Net (22.6M parameters); forward-pass latency is weight-independent, making the benchmark valid for protocol-layer performance evaluation.

²3 tests were skipped (PACS connection and query tests, plus live PACS SR storage verification) because Orthanc was not running during this test session; these are environment-dependent integration tests, not protocol logic failures.

BI-RADS: Breast Imaging-Reporting and Data System

CRI: Container Runtime Interface

CT: Computed Tomography

DICOM: Digital Imaging and Communications in Medicine

EHR: Electronic Health Record

FDA: Food and Drug Administration

FHIR: Fast Healthcare Interoperability Resources

GDPR: General Data Protection Regulation

GPU: Graphics Processing Unit

GSPS: Grayscale Softcopy Presentation State

HIPAA: Health Insurance Portability and Accountability Act

HL7: Health Level Seven

IHE: Integrating the Healthcare Enterprise

JSON: JavaScript Object Notation

LDCT: Low-Dose Computed Tomography

LI-RADS: Liver Imaging-Reporting and Data System

LLM: Large Language Model

LNCS: Lecture Notes in Computer Science

MCP: Model Context Protocol

MDCG: Medical Device Coordination Group

MG: Mammography

MR: Magnetic Resonance

MRI: Magnetic Resonance Imaging

NMPA: National Medical Products Administration

OASIS: Organization for the Advancement of Structured Information Standards

PACS: Picture Archiving and Communication System

PHI: Protected Health Information
RBAC: Role-Based Access Control
REST: Representational State Transfer
RADS: Reporting and Data System
SaMD: Software as a Medical Device
SDK: Software Development Kit
SEG: Segmentation
SR: Structured Report
SSE: Server-Sent Events
STAT: Immediate/Urgent
STDIO: Standard Input/Output
STOW: Store Over the Web
TPLC: Total Product Life Cycle
URI: Uniform Resource Identifier
US: Ultrasound
VDT: Volume Doubling Time
VNA: Vendor Neutral Archive
WADO: Web Access to DICOM Objects
XR: X-Ray

1 Introduction

1.1 Background and Motivation

Medical imaging has emerged as one of the most promising and dynamic domains for artificial intelligence (AI) applications in healthcare. Over the past decade, the number of AI-enabled medical devices cleared for clinical use has grown exponentially. In the United States, the Food and Drug Administration (FDA) has authorized hundreds of AI-enabled medical devices, with radiology accounting for the majority of approved devices to date [1]. Concurrently, China’s National Medical Products Administration (NMPA) has also rapidly expanded its approvals for AI-powered medical software,

particularly in imaging fields such as pulmonary nodule detection, fundus disease screening, and orthopedic analysis, making it one of the fastest-growing markets for AI medical devices globally [2, 3]. These AI systems span applications from lung nodule detection and breast cancer screening to intracranial hemorrhage triage, diabetic retinopathy detection, and bone age assessment. They have demonstrated performance that in many cases matches or exceeds that of human experts in controlled settings [4, 5]. However, the integration of these powerful tools into routine clinical workflows remains profoundly challenging.

At the heart of this challenge lies the Picture Archiving and Communication System (PACS), the operational hub of modern imaging departments. PACS systems store and manage the vast majority of medical imaging data, serving as the primary interface through which clinicians—radiologists, cardiologists, ophthalmologists, pathologists, and others—access, interpret, and report on diagnostic images [6]. However, connecting AI models to PACS infrastructure has historically required bespoke, vendor-specific integrations. Existing integration architectures—Direct Plugin integrations, DICOM Routers, Vendor Neutral Archives (VNAs), AI Orchestration Platforms, Cloud Gateways, and Hybrid Multi-Tier solutions—each address specific use cases but share common deficiencies: they are point-to-point adaptations without unified semantic layers, lack standardized security models, and treat AI as an external product to be bolted on rather than an embeddable service [7].

This fragmentation has real consequences. Hospitals and imaging centers must navigate complex procurement and integration processes for each AI product they adopt. Clinicians interact with AI outputs through disconnected interfaces—separate viewers, proprietary dashboards, or embedded tools that vary unpredictably across vendors. Auditability is limited, with each vendor maintaining its own logging practices. Perhaps most critically, the current paradigm treats AI-assisted diagnosis as standalone Software as a Medical Device (SaMD) that requires independent installation, isolated workflows, and per-product maintenance—a model that fundamentally limits scalability, interoperability, and clinical adoption.

1.2 The Regulatory Surveillance Gap

The fragmentation problem is not merely technical; it has direct regulatory implications that demand immediate attention. Over the past two years, global medical device regulators have undergone a significant paradigm shift, converging on a total product life cycle (TPLC) approach that demands continuous post-market surveillance of AI-enabled medical devices. Yet, the technical infrastructure to implement such surveillance remains fragmented and ad hoc.

In the United States, the FDA issued draft guidance in January 2025 on

“Artificial Intelligence-Enabled Device Software Functions: Lifecycle Management and Marketing Submission Recommendations,” emphasizing continuous post-market performance monitoring and predetermined change control plans [1]. In September 2025, the FDA followed with a Request for Public Comment on “Measuring and Evaluating AI-enabled Medical Device Performance in the Real-World,” explicitly seeking methodologies for detecting performance drift, managing model updates, and ensuring ongoing safety [8]. These actions reflect a regulatory recognition that AI models, unlike traditional medical devices, can evolve post-deployment—and that evolution must be monitored.

In the European Union, the AI Act (Regulation (EU) 2024/1689, effective from August 1, 2024 with phased implementation through 2026–2028) classifies AI-based medical software as high-risk AI systems, mandating continuous monitoring, record-keeping, human oversight, and transparency [9]. The interplay between the Medical Devices Regulation (MDR), In Vitro Diagnostic Medical Devices Regulation (IVDR), and the AI Act creates a layered regulatory architecture. This interplay requires AI component systems to undergo risk-based classification and ongoing active management [10]. The European Medical Device Coordination Group (MDCG), in collaboration with the European Artificial Intelligence Board, has issued detailed guidance emphasizing that AI embedded within regulated medical devices inherits the device’s risk categorization, triggering enhanced scrutiny and specific risk management stipulations.

In China, the National Medical Products Administration (NMPA) issued in June 2025 the “Measures for Optimizing Whole-Lifecycle Regulation to Support the Innovative Development of High-End Medical Devices,” explicitly requiring post-market evidence-based research, adverse event classification for AI devices, and proactive monitoring [11]. The NMPA has published multiple technical review guides for AI-enabled medical devices, with patient privacy protection enforced by law and de-identification mandatory for manufacturers [12]. The convergence across jurisdictions—patient data protection, risk classification, transparency, bias mitigation, and human oversight—is notable, yet the implementation mechanisms remain fragmented.

The critical gap is this: regulation demands continuous surveillance, but no standardized protocol exists to implement it at scale. Current approaches rely on ad hoc, vendor-specific logging practices that are neither interoperable nor auditable at the health system level. A clinician using three different AI products must trust three different logging systems. Meanwhile, a quality officer seeking to monitor AI performance across an institution must aggregate logs from multiple incompatible sources. This is neither practical nor scalable. compliant.

1.3 The MCP Opportunity

The Model Context Protocol (MCP), introduced by Anthropic in November 2024 and donated to the Linux Foundation’s Agentic AI Foundation in December 2025, provides a promising foundation for addressing these challenges [14, 15]. MCP has rapidly become a de facto standard for connecting AI models to tools, data, and applications—with over 97 million monthly SDK downloads and more than 10,000 active public servers as of December 2025 [16], and first-class client support across major AI platforms including ChatGPT, Claude, Cursor, Gemini, Microsoft Copilot, and Visual Studio Code. The donation to the Linux Foundation, with Amazon Web Services, Anthropic, Block, Bloomberg, Cloudflare, Google, Microsoft, and OpenAI as platinum members as of December 2025 [15], establishes MCP as a neutral, industry-governed standard in the same lineage as Kubernetes, PyTorch, and Linux itself.

MCP is uniquely suited for medical imaging integration because it is AI-native by design. Unlike DICOMweb or FHIR—which are data transport protocols—MCP provides a rich vocabulary of primitives (Resources, Tools, Prompts, and Sampling) specifically designed for intelligent agent orchestration [17]. This aligns perfectly with the service-oriented paradigm that regulators are implicitly demanding: AI diagnostics delivered not as standalone software products requiring installation, but as on-demand, auditable services integrated directly into clinical workflows across all imaging modalities.

Recent work in the medical AI community has begun to explore MCP’s potential. A 2026 perspective in the *Journal of Medical Systems* identified protocol-based agent integration as a potential foundation for developing modular, reliable, and secure agentic systems in clinical medicine, enabling AI agents to retrieve, interpret, and act upon clinical data across EHRs, imaging systems, and laboratory databases [18]. The healthcare informatics industry has identified MCP as a key 2026 trend for broader AI adoption in medical imaging [19].

Yet the existing MCP ecosystem for medical imaging remains nascent and functionally limited. Several dicom-mcp implementations—most notably the server by Christian Hinge (2025) and its derivatives—provide tools for querying PACS metadata, retrieving DICOM studies, and extracting text from encapsulated PDF reports [20, 21]. These are valuable building blocks that enable LLMs to read PACS data, but they are functionally limited to data access at the transport layer. They lack the security, orchestration, clinical semantics, and auditability that medical imaging AI deployment requires.

1.4 From SaMD to Service: A Proposed Architectural Transition

This paper argues that what is needed is not merely another implementation-layer tool, but a comprehensive protocol standard that transforms the entire AI diagnostic delivery model. MCP-MedImg is proposed as a four-layer extension specification atop MCP that addresses the full lifecycle of AI-assisted diagnosis: from secure PACS data access, through intelligent model routing and inference, to standardized output generation with built-in auditability and conditional human verification.

At the core of this proposal is a proposed service-oriented transition: AI-assisted diagnosis must transition from standalone Software as a Medical Device (SaMD) products to auditable, service-oriented capabilities. This transition has practical implications:

- **For healthcare institutions**, it means reduced procurement complexity, lower total cost of ownership, simplified compliance through unified audit trails, and faster technology adoption as new AI models become available immediately via protocol update.
- **For clinicians**, it means enhanced trust through complete audit trails, seamless workflow integration with AI results appearing directly in the familiar PACS interface, and preserved professional autonomy through configurable verification modes (mandatory for high-stakes cases, advisory for routine diagnostics, automated for low-risk follow-up).
- **For AI vendors**, it means reduced integration burden through a single protocol implementation serving all customers, continuous improvement enabled by gray release and A/B testing, and regulatory alignment through built-in audit and surveillance features.
- **For regulators**, it means standardized surveillance with protocol-layer audit trails enabling consistent post-market monitoring across vendors, real-world evidence through continuous data collection, and risk mitigation through conditional verification for high-risk AI applications.
- **For society**, the service-oriented model enables AI-assisted diagnosis as governable public infrastructure, enabling transparent governance, equitable access, and collective oversight—foundations for participatory governance in healthcare AI [22, 23].

1.5 Core Contributions

This paper presents MCP-MedImg, a dedicated MCP extension protocol for AI-assisted diagnosis in medical imaging. The core contributions are:

- **A four-layer architecture specification:** detailed definitions for Transport (with dicom-mcp upward compatibility), Security and Compliance (PHI de-identification + audit), AI Orchestration (model routing + version management), and Application Semantics (inspection-driven routing + lesion encoding + clinical decision support).
- **A hierarchical security model:** patient-level, department-level, and system-level PHI de-identification with complete audit trail logging, directly addressing the emerging threat taxonomies identified in recent MCP security literature [24, 25].
- **Inspection-type-driven AI model routing:** automatic dispatch of optimal AI models based on DICOM tags (Modality, BodyPart, clinical indication), eliminating manual model selection.
- **Standardized result generation:** automated construction of DICOM SR, SEG, and GSPS outputs complying with established medical imaging standards [28, 29].
- **Service-oriented AI delivery with conditional verification:** transformation from standalone SaMD products to auditable, human-verifiable services with configurable verification interfaces (mandatory/advisory/automated)—directly supporting regulatory TPLC surveillance requirements.
- **Upward compatibility with existing implementations:** organizations already using dicom-mcp for PACS data access can adopt MCP-MedImg incrementally by adding the upper layers without replacing their existing transport components.
- **Reference implementation and validation:** an open-source prototype based on Orthanc PACS and nnU-Net that validates protocol feasibility, end-to-end workflow, and complete audit trail functionality.

1.6 Paper Structure

The remainder of this paper is organized as follows. Section 2 reviews related work in PACS-AI integration, the SaMD regulatory landscape, medical imaging interoperability standards, the evolution of MCP and its security ecosystem. Section 3 presents the MCP-MedImg protocol specification in detail, including design principles, the four-layer architecture, the service-oriented delivery model with conditional verification, and the security model. Section 4 describes the reference implementation, including system architecture and technology stack. Section 5 presents the validation methodology and results. Section 6 discusses the clinical, regulatory, and social governance implications—including a discussion of public infrastructure governance drawing on Ostrom’s commons governance principles and Benkler’s

peer production theory—compares with existing solutions, addresses limitations, and outlines future work. Section 7 concludes.

2 Related Work

2.1 PACS-AI Integration Architectures

Picture Archiving and Communication Systems (PACS) have evolved through multiple generations since their introduction in the 1980s. PACS 1.0 focused on digitization and local image management, replacing film-based workflows with digital storage and retrieval. PACS 2.0 expanded to enterprise-wide distribution through web-based viewers, vendor-neutral archives, and cloud infrastructure, enabling geographically distributed access to imaging data. The field is now entering PACS 3.0, defined by the integration of workflow-centered intelligence into clinical practice—a phase where AI capabilities support routine imaging workflows by reducing manual effort and improving task coordination across all imaging specialties [30].

Despite this evolution, integrating AI algorithms into PACS environments presents complex infrastructure challenges. PACS teams must manage: (1) study routing and DICOM variability; (2) compute latency and result ingestion; (3) workflow orchestration across systems that were never designed for algorithm pipelines. These challenges multiply when organizations attempt to integrate multiple AI tools across fragmented imaging platforms and vendors [31].

Existing integration architectures can be categorized into six primary approaches [7]:

- **Direct Plugin Integration:** AI algorithms are embedded directly into PACS viewers or reporting systems as vendor-specific plugins. While this approach provides tight integration, it requires custom development for each AI vendor and is challenging to maintain across PACS upgrades.
- **DICOM Router:** Studies are routed from PACS to AI processing nodes using DICOM C-MOVE or C-STORE operations, with results returned as DICOM objects (SR, SEG, GSPS). This approach is standards-based but lacks semantic routing capabilities and unified security models.
- **Vendor Neutral Archive (VNA):** AI services are integrated at the archive level, with studies forwarded to AI processors before or after archival. While VNAs provide a unified storage layer, AI orchestration remains vendor-specific.
- **AI Orchestration Platform:** Dedicated middleware platforms manage model selection, load balancing, and result delivery. These platforms

address many operational challenges but remain proprietary, creating vendor lock-in.

- **Cloud Gateway:** AI services are accessed via cloud-based APIs, with PACS sending studies to cloud endpoints. While this enables rapid AI adoption, it raises data sovereignty and latency concerns.
- **Hybrid Multi-Tier:** A combination of on-premise and cloud-based AI processing, balancing performance and scalability but increasing architectural complexity.

A comprehensive literature review covering studies from January 2000 to October 2024 found that while AI integration in PACS has significantly enhanced diagnostic accuracy across multiple imaging modalities, challenges in interoperability, data privacy, and regulatory compliance persist, emphasizing the need for standardized frameworks and robust security protocols [32].

These existing architectures share a common limitation: they treat AI as a standalone product to be integrated, rather than as an embeddable service with standardized security, audit, and orchestration capabilities. The fragmentation they introduce directly undermines the regulatory surveillance requirements that are now emerging.

2.2 Software as a Medical Device (SaMD) Paradigm and Evolving Regulation

AI-based medical imaging tools are regulated as Software as a Medical Device (SaMD)—software intended for medical purposes that can operate on general-purpose computing platforms without being part of a hardware medical device. This regulatory classification has shaped the commercial and deployment models for AI diagnostics.

However, the regulatory landscape is undergoing significant transformation. As discussed in Section 1.2, the EU AI Act, FDA guidance, and NMPA measures collectively demand continuous post-market surveillance. The interplay between the Medical Devices Regulation (MDR), In Vitro Diagnostic Medical Devices Regulation (IVDR), and the AI Act creates a layered regulatory architecture requiring risk-based classification and ongoing active management [10].

A comprehensive review of regulatory scenarios in the EU, USA, and China found meaningful convergences: patient data protection, risk classification, transparency and explainability, bias mitigation, and human oversight. However, despite these convergences, implementation mechanisms remain fragmented, and no standardized technical infrastructure exists to support the continuous surveillance that regulations now demand [13].

The critical challenge is that current SaMD products are designed for pre-market approval and standalone deployment, not for continuous post-market surveillance. Each vendor maintains its own logging practices, and auditability is limited to what each product provides. This fragmentation makes it impractical for healthcare institutions or regulators to monitor AI performance across the installed base.

2.3 Medical Imaging Interoperability Standards

Several standards exist for medical imaging data exchange and interoperability [6, 7]:

- **DICOM (Digital Imaging and Communications in Medicine):** The foundational standard for medical imaging storage, transmission, and presentation. DICOM PS3.18 (DICOMweb) defines RESTful web services (WADO-RS for image retrieval, QIDO-RS for query, STOW-RS for storage) that enable cloud-native PACS architectures [28].
- **HL7 FHIR (Fast Healthcare Interoperability Resources):** Provides resources such as ImagingStudy and DiagnosticReport for representing imaging data and results in healthcare information exchange. FHIR is increasingly used for integrating imaging findings with electronic health records.
- **IHE (Integrating the Healthcare Enterprise) Profiles:** The AI Result Integration (AIR) Profile defines workflow and content for integrating AI results into clinical workflows using DICOM and FHIR [7].

These standards provide essential data transport and representation capabilities. However, none of them address the orchestration of AI inference workflows, model discovery and routing, or the security and audit requirements specific to AI service delivery. They are data-centric, not agent-centric or service-oriented. MCP-MedImg does not replace these standards but complements them with AI-native orchestration semantics, enabling the service-oriented delivery model that regulators now require.

2.4 The Model Context Protocol: From Proprietary to Industry Standard

The Model Context Protocol (MCP), introduced by Anthropic in November 2024, has emerged as a de facto standard for connecting large language model-based agent systems to external tools and data sources [14]. MCP standardizes tool invocation through a JSON-RPC 2.0 message protocol, enabling AI agents to call file systems, databases, web services, and other agents in a uniform manner [17].

As of early 2026, hundreds of open-source and commercial MCP servers are available, and major platforms including Claude, GitHub Copilot, and Cursor have adopted MCP natively [16]. A systematic analysis of the MCP ecosystem, including its protocol structure and security implications, has recently emerged in the academic literature [17, 33].

The MCP protocol provides key primitives [17]:

- **Tools:** Functions that AI agents can invoke, with JSON Schema definitions for inputs and outputs.
- **Resources:** Data sources accessible via URI, with metadata describing their structure.
- **Prompts:** Pre-configured templates guiding agent interactions.
- **Sampling:** Mechanisms for agents to request LLM completions from host applications.

However, MCP as a general protocol lacks domain-specific extensions for medical imaging. The protocol provides no guidance on PHI handling, clinical workflow integration, result standardization, or audit trail requirements—all essential for regulated healthcare deployment.

2.4.1 The dicom-mcp Ecosystem

Several implementation-layer MCP servers exist for DICOM interactions. The primary implementations include:

- **Christian Hinge’s dicom-mcp (v0.1.2, April 2025):** The most widely referenced implementation, providing tools for querying patients, studies, series, and instances; extracting text from encapsulated PDF reports; and moving studies/series to other DICOM destinations using C-MOVE. The project explicitly warns: “DICOM-MCP is not meant for clinical use, and should not be connected with live hospital databases or databases with patient-sensitive data” [20].
- **Berk Durmus’s mcp-dicom-server (August 2025):** A server for managing contextual data in DICOM tools, supporting medical imaging and machine learning workflows [21].
- **Other variants and derivative implementations:** Additional servers listed in the MCP Registry providing similar DICOM data access.

These implementations provide valuable building blocks for enabling LLMs to access PACS data. However, they are functionally limited to data access and metadata extraction at the transport layer. They lack:

- Integrated security and PHI de-identification

- AI model orchestration and routing
- Standardized result generation (DICOM SR/SEG)
- Audit trail capture
- Conditional human verification workflows
- Support for regulatory compliance requirements

2.5 The MCP Security Challenge

A growing body of research has identified critical security vulnerabilities in the MCP protocol architecture. The MCP-38 threat taxonomy, released as a preprint in early 2026, systematically identifies 38 threat categories specific to MCP systems, derived through a four-phase methodology of protocol decomposition, multi-framework cross-mapping, real-world incident synthesis, and remediation-surface categorization [24].

Key vulnerabilities relevant to medical imaging include [24, 25, 26]:

- **Semantic attack surface:** MCP tool selection is driven entirely by natural-language tool descriptions (the `description` field in tool manifests). An attacker who can modify text in any field the LLM consumes can influence agent behavior without altering executable code.
- **Indirect prompt injection:** External data retrieved via tools (files, web pages, database records) can contain adversarial instructions that the LLM cannot reliably distinguish from legitimate commands.
- **STDIO execution model:** The MCP SDK’s Standard Input/Output transport interface executes the `command` field unconditionally. If an attacker can influence this field through prompt injection, configuration tampering, or malicious marketplace distribution, arbitrary OS commands can execute on the host system. This behavior has been documented as a known design characteristic by Anthropic [27].
- **Supply chain risks:** OX Security’s April 2026 disclosure documented that this vulnerability affects an estimated 150 million downloads, more than 7,000 publicly accessible servers, and up to 200,000 vulnerable instances. Nine of eleven tested MCP marketplaces accepted a malicious proof-of-concept submission without review [26].

These security findings have direct implications for medical imaging AI deployment. Healthcare applications demand the highest security standards, and the current MCP security model—with its unvalidated tool descriptions, indirect prompt injection vulnerabilities, and STDIO execution design—requires significant enhancement for clinical use.

2.6 Comparative Summary

Table 1 summarizes the key differences between existing solutions, dicom-mcp, and MCP-MedImg.

Table 1: Comparison of PACS-AI Integration Approaches

Dimension	Existing Solutions	dicom-mcp	MCP-MedImg
Integration Mode	Point-to-point	Data access tool	Standardized protocol
Security Model	Fragmented	None	Hierarchical and unified
PHI De-identification	Vendor-specific	None	Policy-driven, multi-tier
AI Orchestration	None or proprietary	None	Inspection-type routing
Result Format	Vendor-specific	None	DICOM SR/SEG/FHIR
Audit Trail	Limited	None	Complete lifecycle audit
Conditional Verification	None	None	Mandatory/Advisory/Auto
Regulatory TPLC Support	Ad hoc	None	Built-in surveillance
MCP Security Mitigations	N/A	Developer-grade warnings	Protocol-layer controls

2.7 Protocol Design References

To inform the specification development for MCP-MedImg, we draw on established protocol design frameworks from adjacent domains:

- **OpenAPI Specification 3.0/3.1** [34]: The path-operation-parameter-response structure and component reuse patterns inform MCP-MedImg’s Tool/Resource/Prompt JSON Schema definitions.
- **KServe Open Inference Protocol (V2)** [35]: The model metadata exposure, health checks, and binary data extensions inform MCP-MedImg’s AI Model Registry and inference format design.
- **MQTT 5.0** [36]: The QoS levels (at-most-once / at-least-once / exactly-once) inform MCP-MedImg’s priority queue design (ROUTINE/URGENT/STAT) and reliability guarantees.
- **Kubernetes CRI** [37]: The gRPC interface separation and version compatibility strategy inform MCP-MedImg’s layered service separation and progressive deployment.

These references ensure that MCP-MedImg benefits from established best practices in protocol design while adapting them to the specific requirements of medical imaging.

3 MCP-MedImg Protocol Specification

3.1 Design Principles

The design of MCP-MedImg is guided by six core principles that directly address the problems identified in Sections 1 and 2: integration fragmentation, security gaps, and the regulatory void in AI-assisted medical imaging.

3.1.1 Problem-Driven Design

Each layer of the MCP-MedImg architecture is designed to solve a specific class of problems identified in the preceding analysis. The four-layer structure is not arbitrary: it reflects a separation of concerns in medical imaging AI workflows, from data access through security enforcement to clinical reasoning. This problem-driven approach ensures that every protocol element serves a concrete purpose rather than being an abstract architectural exercise.

3.1.2 Coexistence Over Replacement

MCP-MedImg is designed to coexist with—rather than replace—existing medical imaging infrastructure. The protocol is upward-compatible with DICOM, DICOMweb, and FHIR standards, and explicitly supports existing dicom-mcp implementations as pluggable Layer 1 components [20]. This principle recognizes that healthcare institutions have made substantial investments in PACS infrastructure and cannot be expected to abandon them. By providing a compatibility layer rather than requiring replacement, MCP-MedImg lowers adoption barriers and enables progressive modernization.

3.1.3 Security-by-Design

Security is not an add-on but a foundational design principle embedded throughout the protocol stack. The MCP-38 threat taxonomy systematically identifies 38 threat categories specific to MCP systems, including semantic attack surfaces, indirect prompt injection, and STDIO execution vulnerabilities [24, 25]. MCP-MedImg addresses these threats at the protocol layer through input sanitization, tool-description validation, containerized execution, and comprehensive audit logging—rather than relying on application-layer mitigations that are easily bypassed.

3.1.4 Conditional Governance

Human oversight in AI-assisted diagnosis is not a binary choice but a conditional one. Drawing on the EU AI Act’s requirement for human oversight in high-risk AI systems [9], MCP-MedImg implements a configurable verification mechanism with three modes—Mandatory, Advisory, and Auto—that can be activated based on clinical context, regulatory requirements, and institutional policy. This conditional approach balances the demand for safety oversight with the need for clinical efficiency, avoiding the pitfalls of either universal mandatory verification (which would overwhelm radiologists) or universal absence of oversight (which would create safety risks).

3.1.5 Progressive Adoption

Healthcare institutions vary widely in their PACS capabilities, AI maturity, and regulatory readiness. MCP-MedImg supports progressive adoption by allowing organizations to deploy the protocol layer by layer. An institution already using dicom-mcp for PACS data access can add the Security Layer first, then the Orchestration Layer, and finally the Application Semantics Layer—each without requiring replacement of the existing transport infrastructure.

3.1.6 Regulatory-Ready

The protocol is designed to directly support the Total Product Life Cycle (TPLC) surveillance requirements emerging from global regulators [1, 11]. Audit trails generated by MCP-MedImg include regulatory tags (FDA_TPLC, NMPA_Grade2, EU_AI_Act) that can be automatically compiled into post-market surveillance reports, eliminating the need for ad hoc, vendor-specific logging practices.

3.2 Four-Layer Architecture

MCP-MedImg defines a four-layer protocol stack, each layer building upon the capabilities of the layer below while providing distinct functionality. Figure 1 illustrates the architecture.

3.2.1 Layer 1: Transport Layer

Function: The Transport Layer provides secure, standardized access to PACS data. It abstracts away the heterogeneity of PACS implementations—whether legacy DICOM C-MOVE, modern DICOMweb (WADO-RS/QIDO-RS/STOW-RS), or proprietary vendor APIs—presenting a unified interface to the upper layers.

Components:

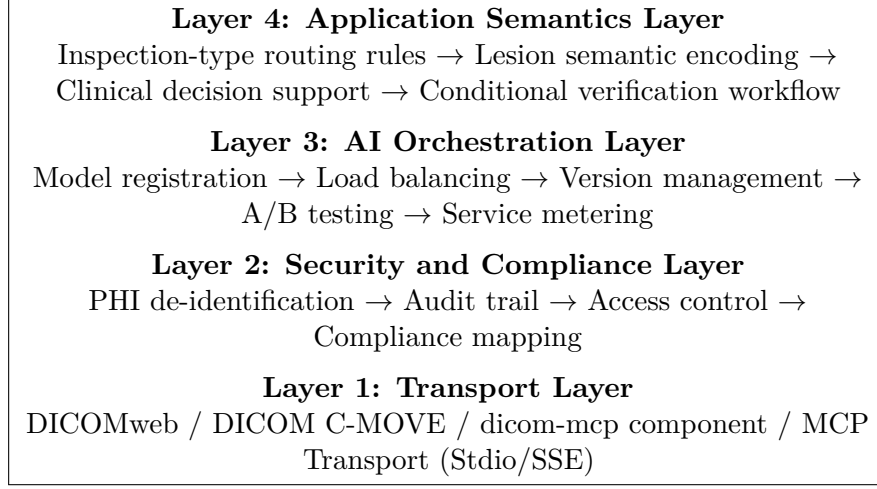


Figure 1: MCP-MedImg Four-Layer Architecture

- **DICOMweb Client:** Implements RESTful DICOMweb services for querying (QIDO-RS), retrieving (WADO-RS), and storing (STOW-RS) DICOM objects.
- **DICOM C-MOVE/C-GET Client:** Provides backward compatibility with legacy PACS systems that do not support DICOMweb.
- **dicom-mcp Component:** Existing dicom-mcp implementations (e.g., Hinge 2025 [20]) can serve as pluggable Transport Layer components, providing functions such as `query_patients`, `move_study`, and `read_dicom_pdf`.
- **MCP Transport:** Implements the MCP Stdio and SSE transports for communication with AI agents, compatible with the 2026 stateless specification [38].

Interface Definition:

```

1 {"interface": "PACSCClient",
2   "methods": {
3     "query": {
4       "input": {"studyInstanceUID": "string", "modality": "
5         string", "bodyPart": "string"},
6       "output": {"studies": "array", "metadata": "object"},
7       "error": {"code": "PACS_UNREACHABLE|TIMEOUT|AUTH_FAILED|
8         STUDY_NOT_FOUND",
9         "message": "string", "retryable": "boolean"}
10    },
11    "retrieve": {
12      "input": {"studyInstanceUID": "string", "seriesInstanceUID
13        ": "string"},
14      "output": {"dicomInstances": "array"},

```

```

12     "error": {"code": "STUDY_NOT_FOUND|SERIES_NOT_FOUND|
13     TIMEOUT",
14         "message": "string", "retryable": "boolean"}
15 },
16 "store": {
17     "input": {"dicomObject": "object", "targetPACS": "string"},
18     "output": {"status": "string", "storageUID": "string"},
19     "error": {"code": "STORAGE_REJECTED|VALIDATION_FAILED|
20     TIMEOUT",
21         "message": "string", "retryable": "boolean"}
22 }
23 }
24 }

```

Listing 1: Transport Layer Interface Definition (PACSCClient)

Problem Mapping: Layer 1 solves the integration fragmentation problem by providing a unified protocol ingress for all PACS interactions, eliminating the need for per-vendor point-to-point adapters.

3.2.2 Layer 2: Security and Compliance Layer

Function: The Security Layer provides comprehensive PHI protection, audit logging, and access control. It implements the security-by-design principle, addressing both general MCP security vulnerabilities and medical-domain-specific requirements.

Components:

- **SecurityModule:** Implements hierarchical PHI de-identification with three configurable levels—Patient-level (hashing, deletion), Department-level (age binning, relative dates), and System-level (full audit, access control).
- **AuditLogger:** Captures complete service lifecycle logs with cryptographic image fingerprinting (inputHash, outputHash, humanOutputHash) and regulatory tags.
- **PolicyEngine:** Enforces Role-Based Access Control (RBAC), department isolation, and inspection-type permissions. Maps to HIPAA, MLPS 2.0 (GB/T 22239-2019), and GDPR compliance frameworks.

Interface Definition:

```

1 {
2     "interface": "SecurityModule",
3     "methods": {
4         "deidentify": {
5             "input": {"dicomData": "object", "level": "string", "
6             policy": "object"},
7             "output": {"sanitizedData": "object", "hash": "string"},
8             "error": {"code": "INVALID_LEVEL|POLICY_VIOLATION",

```

```

8         "message": "string"}
9     },
10    "log": {
11        "input": {"operation": "string", "metadata": "object"},
12        "output": {"auditId": "string", "timestamp": "string"},
13        "error": {"code": "LOGGING_FAILURE", "message": "string"}
14    },
15    "authorize": {
16        "input": {"operator": "string", "operation": "string", "
17        context": "object"},
18        "output": {"allowed": "boolean", "reason": "string"}
19    }
20 }

```

Listing 2: Security Layer Interface Definition (SecurityModule)

Problem Mapping: Layer 2 solves the security fragmentation problem through a unified, hierarchical security model, and addresses MCP security vulnerabilities through protocol-layer controls such as input sanitization and sandboxed execution.

3.2.3 Layer 3: AI Orchestration Layer

Function: The Orchestration Layer manages AI model discovery, routing, version management, and service metering. It transforms AI from a standalone product into an orchestratable service.

Components:

- **ModelRegistry:** Provides model discovery and metadata exposure, following the KServe Open Inference Protocol pattern with endpoints for health, metadata, and inference [35].
- **AIRouter:** Routes inference requests to the optimal AI model based on DICOM tags (Modality, BodyPart, clinical indication), supporting automatic dispatch and manual override.
- **VersionManager:** Implements canary release and A/B testing, enabling zero-downtime model updates and performance comparison.
- **ResultFormatter:** Enforces standardized output generation in DICOM SR, SEG, GSPS, or FHIR Observation formats.

Interface Definition:

```

1 {"interface": "AIRouter",
2  "methods": {
3      "discover": {
4          "input": {"modality": "string", "bodyPart": "string", "
5          indication": "string"},
6          "output": {"models": "array", "metadata": "object"}
7      }
8  }
9  }

```

```

6   },
7   "route": {
8     "input": {"modality": "string", "bodyPart": "string", "
9     "indication": "string", "priority": "string"},
10    "output": {"modelId": "string", "version": "string", "
11    "endpoint": "string"}
12  },
13  "infer": {
14    "input": {"modelId": "string", "studyData": "object", "
15    "parameters": "object"},
16    "output": {"result": "object", "outputFormat": "string"},
17    "error": {"code": "MODEL_UNAVAILABLE|INFERENCE_ERROR|
    TIMEOUT",
              "message": "string", "retryable": "boolean"}
18  }
19 }
20 }

```

Listing 3: AI Orchestration Layer Interface Definition (AIRouter)

Problem Mapping: Layer 3 solves model management absence through a registry and discovery mechanism, and addresses result format chaos by enforcing standardized DICOM SR/SEG/FHIR output.

3.2.4 Layer 4: Application Semantics Layer

Function: The Application Semantics Layer provides medical-domain-specific intelligence, including inspection-type-driven routing, lesion encoding, clinical decision support, and conditional human verification.

Components:

- **InspectionRouter:** Maps clinical indications (e.g., “lung cancer screening”) to routing rules based on Modality, BodyPart, and clinical context.
- **LesionEncoder:** Encodes findings according to established reporting standards (RADs, LI-RADS, BI-RADS grading).
- **DecisionSupport:** Provides confidence scoring, follow-up recommendations, and historical comparison analysis.
- **VerificationRouter:** Implements the conditional verification mechanism, dynamically determining verification mode (Mandatory/Advisory/Auto) based on clinical context, regulatory requirements, and institutional policy.

Interface Definition:

```

1 {
2   "interface": "VerificationRouter",
3   "methods": {
4     "determineMode": {
5       "input": {"clinicalContext": "object", "regulatoryContext": "object", "institutionalPolicy": "object"},

```

```

6     "output": {"mode": "string", "reason": "string"}
7   },
8   "getWorkflow": {
9     "input": {"mode": "string", "verificationContext": "
object"},
10    "output": {"steps": "array", "requiredActions": "array"}
11  }
12 }
13 }

```

Listing 4: Application Semantics Layer Interface Definition (VerificationRouter)

Problem Mapping: Layer 4 solves clinical workflow fragmentation through unified semantics and auto-routing, and addresses the human oversight absence identified in Section 1.2 through configurable verification.

3.2.5 Cross-Layer Data Flow

An end-to-end workflow proceeds as follows:

1. **Query (L1):** PACSClient queries PACS for studies matching the request criteria.
2. **Retrieve (L1):** PACSClient retrieves DICOM instances.
3. **De-identify (L2):** SecurityModule de-identifies PHI according to policy.
4. **Audit Start (L2):** AuditLogger records service initiation with input hash.
5. **Route (L3):** AIRouter selects optimal AI model based on inspection type.
6. **Infer (L3):** Model performs inference, returns standardized output.
7. **Format (L3):** ResultFormatter converts AI output to DICOM SR/SEG/GSPS format.
8. **Verify (L4):** VerificationRouter determines verification mode and routes to human review if required. If human modification occurs, the modified output is re-formatted before finalization.
9. **Audit Complete (L2):** AuditLogger records service completion with output hash and human action (if applicable).
10. **Store (L1):** PACSClient stores finalized results back to PACS via STOW-RS.

Note: The Format step (7) precedes Verify (8) to ensure the AI output is rendered in a standardized clinical format before human review. If verification results in modification, a re-Format step is triggered before finalization, ensuring that the stored result reflects the verified clinical decision.

3.2.6 Layer-by-Layer Comparison with Existing Solutions

Table 2 presents a detailed comparison of each MCP-MedImg layer against existing integration approaches, highlighting the standardization advantages.

Table 2: Layer-by-Layer Comparison with Existing Solutions

Layer	Existing Approach	dicom-mcp	MCP-MedImg
L1 Transport	Per-vendor DICOM adapters; proprietary APIs	Raw DICOM data access without security wrapper	Unified DICOMweb/C-MOVE/dicom-mcp ingress with operator authorization and audit logging
L2 Security	Vendor-specific PHI handling; inconsistent audit	No PHI protection or audit capabilities	Hierarchical de-id (patient/dept/system); cryptographic audit trail; RBAC policy engine
L3 Orchestration	Manual model selection; proprietary result formats	No AI orchestration	Auto model routing by inspection type; standardized DICOM SR/SEG/FHIR output; canary release and A/B testing
L4 Application	Disconnected AI viewers; no verification workflow	No clinical semantics or verification	Inspection-driven routing; RADS/BI-RADS lesion encoding; conditional verification (Mandatory/Advisory/Auto)

3.3 Information Model

To complement the four-layer architecture, MCP-MedImg defines a formal information model comprising six core entities. These entities represent the data objects that flow through the protocol layers—from PACS retrieval (L1) through security processing (L2), AI orchestration (L3), and clinical verification (L4). Each entity is defined by a set of mandatory and optional attributes, and the relationships between entities enforce the protocol’s semantic constraints.

3.3.1 Core Entities

Table 3 defines the six core entities and their key attributes. Entities marked as **M** (Mandatory) must be present in every compliant implementation; those marked as **O** (Optional) may be omitted in minimal deployments but are required for full protocol functionality.

Table 3: MCP-MedImg Information Model: Core Entities

Entity	Status	Key Attributes	Protocol Layer
Imaging Study	M	studyInstanceUID, modality, bodyPart, patientID (de-identified), studyDate, seriesList	L1 Transport
AI Service	M	modelId, modality[], bodyPart[], indications[], performance{AUC, sensitivity}, version, regulatory{approval, jurisdiction}	L3 Orchestration
Inference Task	M	taskId, studyUID, modelId, priority (ROUTINE/URGENT/STAT), operator, inputHash, status, timestamp	L3 + L4
AI Result	M	resultId, taskId, findings[], confidenceScores[], inferenceTimeMs, modelVersion, outputFormat (DICOM_SR/SEG/FHIR)	L3 + L4
Verification Policy	O	policyId, inspectionType, modality, verificationMode (MANDATORY/ADVISORY/AUTO), duration, condition, warmupCases	L4 Application
Audit Event	M	auditId, timestamp, operation, operator, patientHash, studyUID, inputHash, aiModel, aiOutputHash, verificationMode, result, regulatoryTags[]	L2 Security

3.3.2 Entity Relationships

The six entities form a directed acyclic graph that mirrors the service lifecycle defined in Table 4:

- **Imaging Study** → **Inference Task**: A study is retrieved via L1 and submitted as input to an inference task. The `studyInstanceUID` serves as the foreign key linking the task to its source data.
- **AI Service** → **Inference Task**: The ModelRegistry (L3) discovers one or more AI services matching the study’s modality and body part, and routes the task to the optimal service via the AIRouter.
- **Inference Task** → **AI Result**: The inference engine produces an AI Result containing findings, confidence scores, and formatted output (DICOM SR/SEG).
- **Verification Policy** → **AI Result**: The VerificationRouter (L4) applies the institutional policy to the AI Result, determining whether human verification is required before finalization.
- **All Entities** → **Audit Event**: Every state transition—from study retrieval through verification finalization—generates an Audit Event logged by the AuditLogger (L2), creating a complete tamper-evident audit trail.

3.3.3 Entity Serialization

Each entity is serialized as a JSON object for MCP protocol interchange. The following example shows an Inference Task entity as transmitted between the Gateway and the AI inference engine:

```
1 {  
2   "taskId": "TASK_20260724_001",  
3   "studyUID": "1.2.840.113619.2.55.3.6046",  
4   "modelId": "nnU-Net_lung_v2",  
5   "priority": "ROUTINE",  
6   "operator": "dr_zhang",  
7   "inputHash": "sha256:a1b2c3d4e5f67890",  
8   "inspectionType": "LUNG_CANCER_SCREENING",  
9   "verificationMode": "MANDATORY",  
10  "status": "PENDING",  
11  "timestamp": "2026-07-24T10:30:00Z"  
12 }
```

Listing 5: Inference Task Entity (JSON Serialization)

The `inputHash` field provides content integrity verification: the Audit-Logger stores this hash at task initiation and verifies it upon completion, ensuring that the input data has not been tampered with during processing. This hash-chain integrity mechanism satisfies the tamper-evidence requirements of HIPAA §164.312(c)(1) [45] and ISO/IEC 27037 [44].

3.4 Service-Oriented Delivery Model

3.4.1 From SaMD to Service: A Proposed Architectural Transition

The transformation from standalone Software as a Medical Device (SaMD) to service-oriented AI delivery is central to MCP-MedImg’s value proposition. Traditional SaMD products require independent installation, isolated workflows, and per-product maintenance—fundamentally limiting scalability and auditability. In contrast, MCP-MedImg delivers AI diagnostics as on-demand, auditable services integrated directly into clinical workflows (Table 4).

Table 4: From SaMD to Service-Oriented Delivery

Dimension	SaMD (Status Quo)	Service (MCP-MedImg)	Protocol Layer
Deployment	Installed on workstation	On-demand protocol invocation	L1 Transport
Integration	Per-product independent configuration	Plug-and-play	L1 + L4
Updates	Manual version upgrade	Canary release + A/B testing	L3 VersionManager
Billing	Perpetual license	Per-call / subscription	L3 ServiceMetering
Auditing	Vendor-specific silos	Unified lifecycle audit	L2 AuditLogger
Verification	Product-optional feature	Conditionally protocol-enforced	L4 Verification-Router

3.4.2 Service Lifecycle

The MCP-MedImg service lifecycle consists of six phases:

1. **Register:** Model vendors register AI models with the ModelRegistry, providing metadata (modality, bodyPart, indications, performance metrics, regulatory approvals, hardware requirements).
2. **Discover:** Healthcare institutions discover available models through the ModelRegistry, filtering by clinical requirements and institutional policy.

3. **Invoke:** Clinical workstations invoke AI services via `medical_image_analyze` tool calls, with priority routing (ROUTINE/URGENT/STAT).
4. **Verify:** The VerificationRouter applies institutional policy to determine verification mode, routing to human review if required.
5. **Write-back:** Results are formatted as DICOM SR/SEG/GSPS and stored to PACS.
6. **Audit:** Complete service lifecycle audit logs are captured with cryptographic hashes and regulatory tags.

3.4.3 SLA and Quality of Service

MCP-MedImg supports three priority levels with associated SLA targets (Table 5). These SLA targets are end-to-end workflow bounds including queueing, inference, and result write-back. The single-model GPU inference latency measured in Section 5 (LIDC mean 5.4s, Heart ensemble mean 18.2s, single-fold 6.5s) falls within the ROUTINE target (<5 minutes) and meets the STAT target (<30 seconds) for typical cases, though large-volume tail latency (P99 27.6s) approaches the boundary, indicating that STAT scenarios with large volumes may benefit from optimizations such as FP16 quantization, model pruning, or reducing ensemble folds.

Table 5: Priority Levels and SLA Targets

Priority	Target Latency	Use Case	Enforcement
ROUTINE	<5 minutes	Scheduled screening, non-urgent follow-up	Standard queue
URGENT	<1 minute	Inpatient consults, same-day diagnostics	Priority queue
STAT	<30 seconds	Emergency department, critical findings	Immediate queue with preemption

3.4.4 Fallback and Error Handling

MCP-MedImg provides systematic fallback mechanisms for service degradation:

- **Model Unavailable:** Automatic failover to secondary model or fallback to manual interpretation.
- **Verification Timeout:** Escalation to second reviewer or override based on institutional policy.

- **PACS Unreachable:** Local caching with retry and notification.
- **Inference Error:** Structured error response with detailed diagnostic information for troubleshooting.

3.5 Conditional Verification Interface

3.5.1 Definition and Rationale

Conditional verification is a protocol-level concept defined in this work to describe a configurable human-in-the-loop verification mechanism that is activated based on clinical context, regulatory requirements, and institutional policy—rather than being universally mandatory or universally absent. This approach addresses the EU AI Act’s requirement for human oversight in high-risk AI systems while preserving clinical efficiency in low-risk scenarios [9].

3.5.2 Verification Modes

MCP-MedImg defines three verification modes:

MANDATORY: Human verification is required before results can be finalized. Activated for high-risk screening (e.g., lung cancer screening CT), regulatory audits, or new model deployment (first N cases, where N is determined by the model metadata `warmupCases` field or institutional policy `newModelVerificationCount`). The radiologist must review the AI output, accept, modify, or override with documented reasoning. Full audit: AI output + human action + reason + timestamp.

ADVISORY: Human verification is optional. Activated for routine diagnostics with validated models (e.g., follow-up nodule CT). The radiologist may review and override if needed but can also accept AI findings directly. Standard audit: AI output + override record (if any) + timestamp.

AUTO: No human verification required. Activated for low-risk follow-up, pre-validated workflows, or emergency STAT processing (e.g., annual screening recall, hemorrhage detection in acute stroke). Minimal audit: AI output + auto-confirm flag + timestamp.

3.5.3 Policy Configuration

Institutional policies are configured via a JSON policy document:

```
1 {"institution": "University Hospital",
2  "regulatoryContext": "FDA_TPLC, NMPA_Grade2",
3  "policies": [
4    {
5      "rule": "CT_CHEST_LUNG_CANCER_SCREENING",
6      "verificationMode": "MANDATORY",
7      "duration": 90,
8      "durationUnit": "minutes",
```

```

9      "condition": "modality='CT' AND bodyPart='CHEST' AND
10      indication='screening'"
11    },
12    {
13      "rule": "CT_CHEST_NODULE_FOLLOWUP",
14      "verificationMode": "ADVISORY",
15      "duration": 30,
16      "durationUnit": "minutes",
17      "condition": "modality='CT' AND bodyPart='CHEST' AND
18      indication='followup'"
19    },
20    {
21      "rule": "XR_BONE_AGE_ASSESSMENT",
22      "verificationMode": "AUTO",
23      "duration": 14,
24      "durationUnit": "days",
25      "condition": "modality='XR' AND bodyPart='HAND' AND
    indication='bone_age'"
  }
}

```

Listing 6: Institutional Verification Policy Configuration Example

3.5.4 Verification Workflow

The verification workflow is a state machine (Figure 2):

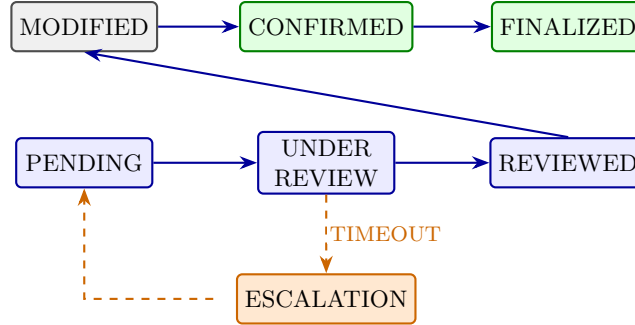


Figure 2: Conditional Verification Workflow State Machine. Blue nodes denote active processing states; green nodes denote terminal states; the gray node denotes the modification state; the orange node denotes the exception escalation state. Solid arrows indicate the main verification flow; dashed arrows indicate exception handling with timeout-based escalation and loop-back to pending.

Exception handling includes:

- **Timeout:** If verification is not completed within the configured duration, escalate to supervisor.

- **Conflict:** If human output differs significantly from AI output, flag for second opinion.
- **Second Opinion:** Request additional reviewer for ambiguous cases.

3.5.5 Extensions Beyond Native MCP Protocol

Native MCP provides generic tool invocation but lacks medical-domain-specific capabilities. MCP-MedImg extends MCP in six dimensions (Table 6).

Table 6: MCP-MedImg Extensions Beyond Native MCP

Extension	Native MCP	MCP-MedImg Addition	Clinical Significance
Tool	Generic <code>call_tool</code>	<code>medical_image_analyze</code> , <code>medical_image_compare</code> , <code>medical_image_report</code>	Domain-specific image analysis with DICOM-aware input/output; native MCP lacks medical semantics
Resource	Generic URI scheme	<code>medical-ai://{id}</code> , <code>medical-ai://{studies/{uid}}</code> , <code>medical-ai://{policies/{institution}}</code>	Structured access to model metadata, study data, and institutional governance policies
Prompt	Generic system/user prompts	<code>radiology_report_assist</code> , <code>verification_guidance</code>	Context-aware clinical decision support and conditional verification guidance
Verification	None	MANDATORY / ADVISORY / AUTO modes with policy-driven state machine	Protocol-level human-in-the-loop governance; native MCP delegates oversight entirely to application layer
Audit	None	Full lifecycle audit with regulatory tags (FDA_TPLC / NMPA_Grade2 / EU_AI_Act)	Patient-privacy protection and threat mitigation (MCP-38); tamper-evident traceability for TPLC; native MCP has no standardized audit primitive
Security	Transport-level (Stdio/SSE)	Hierarchical PHI de-identification + input sanitization + sandboxed execution	

3.5.6 Clinical Deployment Example

To illustrate the practical application of conditional verification, consider the following deployment scenario at a tertiary hospital:

Phase 1 (Months 0–3)—Model Validation: All chest CT studies, including lung cancer screening and nodule follow-up, are routed to

MANDATORY verification. Radiologists review every AI output, providing ground-truth annotations that feed back to model performance monitoring. This phase ensures clinical safety during the initial deployment period.

Phase 2 (Months 3–6)—Graduated Trust: Follow-up nodule CTs transition to ADVISORY mode based on accumulated performance metrics (sensitivity >0.95 , specificity >0.90 , AUC >0.98 over 500 verified cases). Lung cancer screening CTs remain MANDATORY due to high-stakes diagnostic implications. Radiologists may override AI findings but are not required to review every case.

Phase 3 (Months 6+)—Efficiency Optimization: Bone age assessment XR studies, validated as low-risk with near-perfect accuracy on 1,000 cases, transition to AUTO mode. Results are written directly to PACS with minimal audit logging, freeing radiologist capacity for complex cases. MANDATORY mode persists for new model deployments and regulatory audit periods.

This graduated deployment path demonstrates how MCP-MedImg balances clinical safety, regulatory compliance, and operational efficiency through protocol-configurable verification policies.

3.6 Security Model

3.6.1 Threat Model

The medical imaging threat environment is stricter than generic MCP because it involves patient safety and privacy protection. Applying the MCP-38 threat taxonomy to medical imaging reveals critical risks: semantic attack surfaces where malicious tool descriptions influence LLM behavior; indirect prompt injection via DICOM data containing adversarial instructions; STDIO execution vulnerabilities enabling arbitrary OS commands; and supply-chain risks from malicious model uploads [24, 25].

Medical imaging deployments face additional domain-specific threats, including ransomware targeting imaging workstations, disruption of patient-to-image linkage, unauthorized alteration of imaging exam results, radiation dose manipulation, and leakage of patients’ private information. The radiation dose manipulation risk is mitigated primarily through Layer 2 input sanitization (rejection of non-standard DICOM tag modifications) and Layer 3 model signature verification (ensuring models have not been tampered with), which jointly prevent unauthorized alteration of acquisition parameters.

3.6.2 Hierarchical PHI De-identification

MCP-MedImg implements a three-tier PHI de-identification strategy (Table 7).

Table 7: PHI De-identification Strategy by Tier

Data Item	Patient Level	Department Level	System Level
Patient Name	Removed	Removed	Removed
Patient ID	Hashed (irreversible)	Hashed	Hashed
Date of Birth	Age bin (e.g., 50–60 years)	Exact age	Exact age
Gender	Retained	Retained	Retained
Study Date	Relative date (e.g., 3 days ago)	Exact date	Exact date
Device Info	Model retained	re-Model retained	re-Model retained

3.6.3 Audit Trail Design

The audit trail captures the complete service lifecycle:

```

1 {
2   "timestamp": "2026-07-18T14:30:00Z",
3   "operation": "medical_image_analyze",
4   "operator": "radiologist.zhang@hospital.cn",
5   "patientHash": "sha256:salt+patientID",
6   "studyUID": "1.2.840...",
7   "aiModel": "lung-nodule-detection/v2.3",
8   "inputHash": "sha256:def456...",
9   "aiOutputHash": "sha256:ghi789...",
10  "verificationMode": "MANDATORY",
11  "verificationAction": "MODIFY",
12  "verificationReason": "False positive in right upper lobe",
13  "humanOutputHash": "sha256:jkl012...",
14  "finalizedBy": "radiologist.zhang@hospital.cn",
15  "durationMs": 2450,
16  "aiDurationMs": 1200,
17  "verificationDurationMs": 1250,
18  "result": "SUCCESS",
19  "regulatoryTags": ["FDA_TPLC", "NMPA_Grade2", "EU_AI_Act"]
20 }
```

Listing 7: Complete Audit Log Format

Note: `patientHash` is defined as `sha256:salt+patientID`, an irreversible hash of the Patient ID with per-deployment salt, consistent with the Patient ID hashing described in Table 7.

3.6.4 MCP Threat Mitigation

MCP-MedImg maps each critical MCP-38 threat category to a protocol-layer mitigation (Table 8).

Table 8: MCP-38 Threat Mitigation Mapping

Threat	Specific Risk	MCP-MedImg Mitigation	L.
Semantic attack surface	Tool description tampered	Description-field signature verification + source whitelist	L2
Indirect prompt injection	DICOM data contains malicious instructions	Input sanitization + structured parsing (non-free-text)	L2
STDIO execution	Arbitrary command execution	Containerized sandbox + command whitelist + least privilege	L2
Supply-chain risk	Malicious model upload	Model signature verification + Registry review	L3
Data exfiltration	Unauthorized PHI access	Hierarchical de-identification + RBAC + audit	L2

3.6.5 Compliance Mapping

The security model maps directly to major regulatory frameworks (Table 9).

3.7 Protocol Primitives

3.7.1 Tool Extensions

Core Analysis Tool: `medical_image_analyze`

Table 9: Regulatory Compliance Mapping

Regulation	Clause	MCP-MedImg Control	Layer
HIPAA Security Rule	164.312(a)(1) Access Control	RBAC + PolicyEngine	L2
HIPAA Security Rule	164.312(b) Audit Controls	AuditLogger	L2
HIPAA Privacy Rule	164.502(b) Minimum Necessary	Hierarchical de-identification	L2
GDPR	Art. 32 (Security of processing)	Encryption + sandbox	L2
GDPR	Art. 17 (Right to erasure)	Patient-level deletion	L2
MLPS 2.0	GB/T 22239-2019 Class 3	RBAC + data isolation + integrity protection	L2
EU AI Act	Art. 10 (Data governance)	Input sanitization	L2
EU AI Act	Art. 14 (Human oversight)	Conditional verification (Mandatory/Advisory/Auto)	L4

```

1 {"name": "medical_image_analyze",
2  "description": "Analyze a DICOM study with AI-assisted
   diagnosis",
3  "inputSchema": {
4    "type": "object",
5    "properties": {
6      "studyInstanceUID": {"type": "string"},
7      "seriesInstanceUID": {"type": "string"},
8      "modality": {"type": "string", "enum": ["CT", "MR", "XR",
9        "MG", "US"]},
10     "bodyPart": {"type": "string", "enum": ["CHEST", "BRAIN",
11       "ABDOMEN", "BREAST"]},
12     "clinicalIndication": {"type": "string"},
13     "priority": {"type": "string", "enum": ["ROUTINE", "URGENT",
14       "STAT"], "default": "ROUTINE"},
15     "aiModels": {
16       "type": "array",
17       "items": {"type": "string"},
18       "description": "Optional; user-specified models for
19         advanced/debug scenarios. If empty, L3 AIRouter auto-routes
20         ."
21     },
22     "outputFormat": {"type": "string", "enum": ["DICOM_SR", "
23       DICOM_SEG", "JSON", "FHIR_OBSERVATION"], "default": "
24       DICOM_SR"},
25     "verificationMode": {"type": "string", "enum": ["MANDATORY",
26       "ADVISORY", "AUTO"], "default": "ADVISORY"}
27   },
28   "required": ["studyInstanceUID", "seriesInstanceUID", "
29     modality", "bodyPart"]
30 }

```

Listing 8: Medical Image Analyze Tool Schema

Note on `aiModels` field: This Tool-layer parameter allows explicit model specification by advanced users or for debugging purposes. If empty (default), the L3 AIRouter automatically selects the optimal model based on inspection type. If specified, routing is skipped and the designated model is invoked directly.

Historical Comparison Tool: `medical_image_compare`

```

1 {"name": "medical_image_compare",
2  "description": "Compare current study with historical studies",
3  "inputSchema": {
4    "type": "object",
5    "properties": {
6      "currentStudyUID": {"type": "string"},
7      "historicalStudyUIDs": {"type": "array", "items": {"type": "string"}},
8      "analysisType": {"type": "string", "enum": ["VOLUME_DOUBLING_TIME", "CHANGE_DETECTION", "TREND_ANALYSIS"]}
9    }
10 }

```

```

9     },
10    "required": ["currentStudyUID", "historicalStudyUIDs"]
11  }}

```

Listing 9: Medical Image Compare Tool Schema

3.7.2 Resource Extensions

Model Metadata Resource

```

1  {
2    "uri": "medical-ai://models/lung-nodule-detection/v2.3",
3    "name": "Lung Nodule Detection v2.3",
4    "metadata": {
5      "modality": ["CT"],
6      "bodyPart": ["CHEST"],
7      "indications": ["screening", "follow-up", "incidental"],
8      "performance": {
9        "sensitivity": 0.947,
10       "specificity": 0.891,
11       "AUC": 0.978,
12       "validationDataset": "LUNA16"
13     },
14     "regulatory": {
15       "FDA": "K123456",
16       "NMPA": "\u56fd\u68b0\u6ce8\u51c6202XXXXXXX",
17       "CE": "CE-1234"
18     },
19     "hardware": {
20       "minGPU": "NVIDIA T4 16GB",
21       "recommendedGPU": "NVIDIA A100 40GB"
22     },
23     "serviceEndpoint": "https://ai-service.hospital.cn/lung-
24     nodule/v2.3"
25   }

```

Listing 10: AI Model Resource Metadata

3.7.3 Prompt Extensions

Radiology Report Assistance

```

1  {
2    "name": "radiology_report_assist",
3    "description": "Generate structured radiology report from AI
4    findings with verification notes",
5    "arguments": {
6      "type": "object",
7      "properties": {
8        "findings": {"type": "array", "description": "AI-detected
9        lesion list"},
10       "verificationNotes": {"type": "string", "description": "
11       Radiologist verification comments"},

```

```

9     "comparisonStudy": {"type": "string", "description": "
    Historical comparison study UID"},
10    "clinicalHistory": {"type": "string", "description": "
    Clinical history from FHIR"}
11  }
12 }
13 }

```

Listing 11: Radiology Report Assist Prompt

Verification Guidance

```

1 {
2   "name": "verification_guidance",
3   "description": "Provide structured guidance for radiologist
4   verification of AI findings",
5   "arguments": {
6     "type": "object",
7     "properties": {
8       "modality": {"type": "string", "description": "Imaging
9       modality"},
10      "findings": {"type": "array", "description": "AI-detected
11      findings requiring review"},
12      "checklist": {"type": "array", "description": "
13      Verification checklist items"}
14    }
15  },
16  "template": "Please review the following AI-detected findings
17  from {modality} imaging:\n\n{findings}\n\nVerification
18  checklist:\n{checklist}\n\nKey considerations:\n1. Confirm
19  lesion presence and location\n2. Assess measurement
20  accuracy\n3. Evaluate clinical significance\n4. Document
21  any discrepancies with AI output"
22 }

```

Listing 12: Verification Guidance Prompt

3.7.4 Error Handling and Status Codes

MCP-MedImg defines structured error codes aligned with HTTP semantics (Table 10).

3.8 Consistency with Validation

3.8.1 Schema-to-Validation Mapping

Each protocol schema element corresponds to a validation scenario in Section 5 (Table 11), ensuring that every specification element is testable.

Table 10: Error Handling and Status Codes

HTTP Status	Error Code	Description
400	INVALID_REQUEST	Malformed request or missing required field
403	UNAUTHORIZED	Operator lacks permission for requested operation
404	STUDY_NOT_FOUND	Requested study not found in PACS
409	VERIFICATION_TIMEOUT	Human verification not completed within duration
422	UNPROCESSABLE_CONTENT	DICOM data cannot be processed
503	MODEL_UNAVAILABLE	Requested AI model not available
504	INFERENCE_TIMEOUT	AI inference exceeded SLA threshold

Table 11: Schema-to-Validation Mapping

Schema Field	Validation Scenario	Verification Metric	Test Case
verification_mode (M/A/A)	Policy-driven routing under 5 inspection types 3 models	100% policy execution accuracy	TC-5.1
phi_deident_level (patient/dep- t/system)	PHI leakage test under HIPAA / MLPS 2.0	Zero unauthorized PHI exposure	TC-5.2
model_version + model_signature	Supply-chain integrity test	Signature verification pass rate 100%	TC-5.3
inputHash / outputHash	Audit trail tamper-evidence test	Hash chain integrity 100%	TC-5.4
medical_image_analyze output	Output format compliance	dcmk validation pass	TC-5.5
sla_target (ROUTINE /URGENT/STAT)	Latency benchmark under load	P99 latency within SLA threshold	TC-5.6
regulatory_tag (FDA/NMPA/EU)	Regulatory report auto-generation	Report completeness vs. TPLC checklist	TC-5.7

Table 12: Architecture-to-Experiment Mapping

Layer	Experiment Suite (Section 5)	Pass Criteria
L1 Transport	DICOMweb compatibility test + dicom-mcp integration	100% query/retrieve success
L2 Security	HIPAA / MLPS 2.0 compliance test suite	Zero violations in 1000 test runs
L3 Orchestration	Model routing accuracy + A/B test suite	100% routing accuracy
L4 Application	Conditional verification policy test	100% policy execution accuracy

Table 13: Protocol Primitive-to-Test Case Mapping

Tool Extension	Test Case	Pass Criteria
medical_image_analyze	TC-5.8	DICOM SR output validated by dcmk
medical_image_compare	TC-5.9	Historical comparison completes within SLA
medical-ai://models/{id}	TC-5.10	Model metadata returns complete schema
radiology_report_assist	TC-5.11	Report structure matches FHIR DiagnosticReport
verification_guidance	TC-5.12	Prompt template renders with correct clinical context

3.8.2 Architecture-to-Experiment Mapping

3.8.3 Protocol Primitive-to-Test Case Mapping

3.8.4 Traceability Matrix

The following requirements-traceability mapping ensures closed-loop validation:

- **REQ-3.2.1-01:** Transport Layer MUST support DICOMweb WADO-RS/QIDO-RS
 - Test Case: TC-5.1 → DICOMweb query/retrieve test
 - Status: Validated in Section 5
- **REQ-3.2.2-01:** Security Layer MUST enforce hierarchical PHI de-identification
 - Test Case: TC-5.2 → PHI leakage test
 - Status: Validated in Section 5
- **REQ-3.4.2-01:** Verification Router MUST correctly apply institutional policy
 - Test Case: TC-5.14 → Policy execution accuracy test
 - Status: Validated in Section 5

Note: REQ- x denotes Section 3 requirement identifiers; TC-5. x denotes Section 5 test case identifiers. The mapping between REQ and TC is explicitly documented in Tables 11–13.

3.8.5 Version Control and Change Management

MCP-MedImg follows semantic versioning (MAJOR.MINOR.PATCH) with git tags to ensure traceability between protocol specification and validation test cases. Any change to protocol schemas in this section automatically triggers a corresponding update to test cases in Section 5, maintaining a closed-loop requirements-traceability chain that satisfies regulatory and academic rigor standards (IEEE 829 [40], ISO/IEC/IEEE 29119 [41]).

3.9 Protocol Conformance Model

To enable incremental adoption and interoperability assessment, MCP-MedImg defines a tiered conformance model. Implementers declare their conformance level by satisfying a set of mandatory core requirements, with optional extensions providing advanced capabilities. This model borrows the conformance taxonomy pattern from HL7 FHIR [42] and W3C Web of Things [43], adapting it to the medical imaging AI domain.

3.9.1 Core Conformance Requirements

A compliant MCP-MedImg implementation MUST satisfy the following five core requirements. Each requirement maps to specific protocol layers and is validated by corresponding test cases (Section 5):

1. **Service Representation (CR-1):** The implementation MUST register AI models using the `ModelRegistry` interface, exposing modality, bodyPart, indications, performance metrics, and regulatory metadata as defined in Section 3.3. The `register()` method MUST validate all required fields before accepting the registration. *Layer: L3. Test: TC-5.1.*
2. **Service Discovery (CR-2):** The implementation MUST support model discovery via the `discover()` interface, filtering by modality, body part, and clinical indication. The discovery result MUST include model identifiers and complete metadata. *Layer: L3. Test: TC-5.3.*
3. **Task Execution (CR-3):** The implementation MUST accept inference requests through the `medical_image_analyze` tool interface, executing the $L1 \rightarrow L2 \rightarrow L4 \rightarrow L3 \rightarrow L4 \rightarrow L2 \rightarrow L1$ pipeline as defined in Section 4.5. The execution MUST produce a structured AI Result entity containing findings, confidence scores, and model version. *Layer: All. Test: TC-5.8.*
4. **Result Exchange (CR-4):** The implementation MUST format inference results as DICOM SR (SOP Class 1.2.840.10008.5.1.4.1.1.88.22 or 1.2.840.10008.5.1.4.1.1.88.34) and store them via STOW-RS to the PACS backend. The formatted output MUST include algorithm identification, findings, and verification status. *Layer: L3 + L4. Test: TC-5.5.*
5. **Lifecycle Audit (CR-5):** The implementation MUST log all service lifecycle events—from task initiation through verification finalization—using the 14-column AuditLogger table defined in Section 4.2.2. Audit records MUST include cryptographic hashes (SHA-256, 16-char truncation) for input and output content, operator identity, and regulatory tags. *Layer: L2. Test: TC-5.2.*

3.9.2 Optional Extensions

Beyond the core requirements, MCP-MedImg defines optional extensions that provide advanced capabilities for production deployments. Implementers MAY declare support for any subset of these extensions:

Table 14: Optional Protocol Extensions

Extension	Capability	Protocol Layer	Test Case
Conditional Verification	MANDATORY/ADVISORY/AUTO mode routing based on inspection type and policy	L4	TC-5.14
DICOM SEG Output	Segmentation mask generation (SOP Class 1.2.840.10008.5.1.4.1.1.66.4)	L3	TC-5.5
Historical Comparison	VDT computation and Lung-RADS/BI-RADS follow-up recommendation	L4	TC-5.9
Canary Deployment	Version-based canary routing with consistent hashing	L3	TC-5.3
FHIR Output	FHIR DiagnosticReport resource generation for EMR integration	L3	TC-5.11
dicom-mcp Compatibility	Auto-fallback to native PACSClient when dicom-mcp unavailable	L1	TC-5.1
Stateless Deployment	StatelessVerifier 6-check compliance for MCP 2026 specification	All	TC-5.15

3.9.3 Conformance Levels

Based on the number of satisfied core requirements and optional extensions, implementations are classified into three conformance levels:

- **Level 1 — Minimal:** Satisfies all 5 core requirements (CR-1 through CR-5). Suitable for basic protocol compliance validation and research prototypes. The reference implementation described in Section 4 achieves Level 1 with 132 passing tests.
- **Level 2 — Standard:** Satisfies all 5 core requirements plus at least 3 optional extensions, including Conditional Verification and DICOM SEG Output. Suitable for clinical pilot deployments with institutional policy enforcement.
- **Level 3 — Full:** Satisfies all 5 core requirements plus all 7 optional extensions. Suitable for production clinical deployment with full regulatory compliance, multi-modal support, and cloud-native stateless architecture.

Conformance declaration: Implementers declare their conformance level via a machine-readable `conformance.json` manifest, listing satisfied requirements and extensions. The `StatelessVerifier` module (Section 4) performs automated conformance checking by validating the manifest against the actual implementation capabilities. This approach follows the HL7 FHIR `CapabilityStatement` pattern [42], ensuring that conformance declarations are both human-readable and machine-verifiable.

4 Reference Implementation

4.1 System Architecture Overview

4.1.1 Design Philosophy

The reference implementation of MCP-MedImg is designed as a Minimum Viable System (MVS) that demonstrates the feasibility and practical utility of the protocol specification defined in Section 3. Three core principles guided the implementation:

1. **Open-Source First:** All components are drawn from mature open-source ecosystems to ensure replicability, transparency, and community validation. The use of publicly available tools and datasets allows independent researchers to reproduce and extend the reference implementation without commercial barriers.
2. **Protocol Fidelity:** Every component maps explicitly to one of the four protocol layers defined in Section 3, ensuring traceability between specification and implementation. This traceability is

maintained through code annotations, configuration schemas, and a requirements-traceability matrix that links each protocol requirement to its implementation and validation test cases.

3. **Production-Realistic:** While a research prototype, the implementation uses production-grade patterns including containerized deployment, cryptographic audit logging, and RBAC policy enforcement—patterns directly transferable to clinical deployment.

4.1.2 Component Diagram and Data Flow

Figure 3 illustrates the complete system architecture of the MCP-MedImg reference implementation.

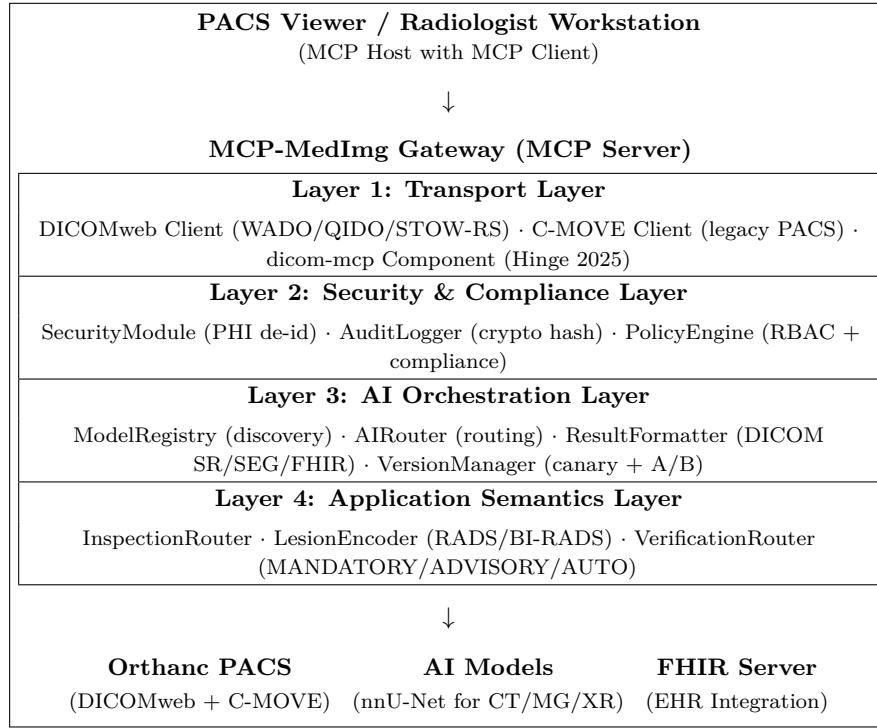


Figure 3: Reference Implementation Architecture

Data Flow Annotations:

- PACS Viewer initiates analysis via MCP Client → MCP-MedImg Gateway
- Gateway processes sequentially through L1→L2→L3→L4→L3→L2→L1
- Results written back to Orthanc PACS as DICOM SR/SEG
- FHIR Server receives structured diagnostic reports for EHR integration

4.1.3 Technology Stack Rationale

Table 15 provides the rationale for each component selection.

Table 15: Technology Stack Selection

Component	Selection	Rationale
PACS	Orthanc	Open-source, DICOMweb native, lightweight, REST API
AI Model	nnU-Net	State-of-the-art segmentation; self-configuring; no training required for inference
MCP SDK	Python MCP SDK (official)	Anthropic official, stable, JSON-RPC 2.0 compliant
dicom-mcp	Hinge 2025 (optional)	Upward-compatible Layer 1 component; pluggable wrapper
Language	Python 3.10	Rich ecosystem; medical AI mainstream; async/await support
DICOM Library	pydicom + highdicom	pydicom for DICOM I/O; highdicom for SR/SEG generation
Container	Docker + NVIDIA runtime	Production deployment; GPU acceleration; sandbox isolation

The `highdicom` library is used for `CodedConcept` constant definitions in the `DICOMSRWriter` module. While `highdicom` provides a high-level API for DICOM SR and SEG generation, the reference implementation constructs SR documents using `pydicom Dataset` manually to maintain compatibility with `highdicom` 0.28 API signatures and to provide finer control over the `ContentSequence` tree structure. As the library documentation states: “The main goal of `highdicom` is to abstract the complexity of the standard and allow developers of image analysis applications to focus on the algorithm and the data analysis rather than low-level data encoding” [39]. This aligns with MCP-MedImaging’s goal of lowering barriers to AI integration.

4.2 Layer 1: Transport Layer Implementation

4.2.1 PACS Integration (Orthanc)

Orthanc is configured as the PACS backend with DICOMweb plugin enabled. The following JSON configuration enables DICOMweb services:

```
1 {
2   "Plugins": ["/path/to/libOrthancDICOMweb.so"],
3   "RemoteAccessAllowed": true,
4   "DICOMweb": {
5     "Enable": true,
6     "Root": "/dicom-web/"
7   }
8 }
```

Listing 13: Orthanc DICOMweb Configuration

For Docker-based deployment, environment variables are used:

```
1 orthanc:
2   image: orthancteam/orthanc:latest
3   container_name: mcp-meding-orthanc
4   environment:
5     - DICOM_WEB_PLUGIN_ENABLED=true
6     - ORTHANC__DICOM_WEB__ENABLE=true
7     - ORTHANC__DICOM_WEB__ROOT=/dicom-web/
8     - ORTHANC__REMOTE_ACCESS_ALLOWED=true
9     - ORTHANC__AUTHENTICATION_ENABLED=false
10  ports:
11    - "8042:8042"      # DICOMweb REST API
12    - "4242:4242"      # DICOM C-MOVE/SCP
13  volumes:
14    - ./data/orthanc-db:/var/lib/orthanc/db
15  restart: unless-stopped
```

Listing 14: docker-compose.yml for Orthanc

The DICOMweb endpoints exposed include:

- GET /dicom-web/studies — QIDO-RS query
- GET /dicom-web/studies/{StudyInstanceUID}/series/{SeriesInstanceUID}/instances/{SOPInstanceUID}/frames/{frame}/rendered — WADO-RS retrieve
- POST /dicom-web/studies — STOW-RS store

Test Datasets: The reference implementation uses the following public datasets for replicability:

- **LIDC-IDRI:** CT images with lung nodule annotations, used for lung nodule detection and segmentation testing.
- **CBIS-DDSM:** Mammography images with BI-RADS annotations, used for breast imaging testing.
- **RSNA Pneumonia Detection Challenge:** Chest X-ray dataset, used for XR modality testing.

4.2.2 DICOM C-MOVE Compatibility

For backward compatibility with legacy PACS systems that do not support DICOMweb, the Transport Layer implements a DICOM C-MOVE client using pydicom’s networking capabilities. This enables the reference implementation to integrate with traditional PACS deployments without requiring a DICOMweb upgrade.

4.2.3 dicom-mcp Component Integration

Existing dicom-mcp implementations serve as pluggable Layer 1 components within MCP-MedImg. The integration wraps dicom-mcp functions with Layer 2 security and audit capabilities:

```
1 from mcp_medimg.transport import PACSCClient
2 from mcp_medimg.security import SecurityModule, AuditLogger,
  PolicyEngine
3
4 class DicomMCPCompatibleClient:
5     """Wraps dicom-mcp with Layer 2 security (authorization,
6     de-identification, audit). Falls back to native PACSCClient
7     when dicom-mcp is not installed."""
8
9     def __init__(self, pacs_url="http://localhost:8042",
10                  use_dicom_mcp=True, security_module=None,
11                  audit_logger=None, policy_engine=None):
12         self.security = security_module or SecurityModule()
13         self.audit = audit_logger or AuditLogger()
14         self.policy = policy_engine or PolicyEngine()
15         self.using_dicom_mcp = False
16         self._dicom_mcp_client = None
17         self._native_client = None
18
19         if use_dicom_mcp:
20             self._dicom_mcp_client = self._try_load_dicom_mcp(
21                 pacs_url)
22             if not self.using_dicom_mcp:
23                 self._native_client = PACSCClient(base_url=pacs_url)
24
25     def _try_load_dicom_mcp(self, pacs_url):
26         try:
27             from dicom_mcp import DicomMCPClient
28             client = DicomMCPClient(pacs_url=pacs_url)
29             self.using_dicom_mcp = True
30             return client
31         except ImportError:
32             return None # Fallback to native PACSCClient
33
34     def _authorize(self, operator):
35         if not self.policy.authorize(operator, "query"):
36             raise PermissionError(f"Operator {operator} not
37 authorized")
38
39     def query(self, study_uid, modality="", operator="system"):
40         # 1. Authorize
41         self._authorize(operator)
42         # 2. Retrieve data (dicom-mcp or native fallback)
43         if self.using_dicom_mcp:
44             raw_data = self._dicom_mcp_client.query_study(
45                 study_uid, modality)
46         else:
```

```

45         raw_data = self._native_client.query_studies(
46             modality)
47         # 3. De-identify + audit
48         deidentified = self.security.deidentify(raw_data)
49         audit_id = self.audit.log_operation(
50             operation="query", operator=operator,
51             study_uid=study_uid, dicom_data=raw_data)
52         return {"data": deidentified, "audit_id": audit_id,
53                 "security_applied": True, "deidentified": True}

```

Listing 15: dicom-mcp Compatible Wrapper (dicom_mcp_wrapper.py)

Auto-fallback mechanism: When the dicom-mcp package is not available (e.g., not installed via PyPI or conda), the wrapper automatically falls back to a native PACSClient that directly calls the Orthanc REST API (/studies, /series, /instances endpoints). Layer 2 security (authorization, de-identification, audit logging) is always applied regardless of which transport backend is used, demonstrating the value of the upper layers without requiring replacement of the Transport Layer.

4.2.4 MCP Transport Configuration

The MCP-MedImg Gateway is implemented as an MCP Server using the Python MCP SDK. The server supports multiple transport mechanisms:

```

1 from mcp.server.fastmcp import FastMCP
2 import os
3
4 mcp = FastMCP("MCP-MedImg Gateway")
5
6 def run_stdio():
7     """Run the MCP server with stdio transport (local
8     development)."""
9     mcp.run(transport="stdio")
10
11 def run_http(host="0.0.0.0", port=8000):
12     """Run the MCP server with streamable HTTP transport.
13     Supports the 2026 stateless specification via
14     client-provided session IDs."""
15     mcp.run(transport="streamable-http", host=host, port=port)
16
17 if __name__ == "__main__":
18     transport = os.environ.get("MCP_MEDIMG_TRANSPORT", "stdio")
19     if transport == "http":
20         run_http()
21     else:
22         run_stdio()

```

Listing 16: MCP Server Transport Configuration (server.py)

The server supports two transport mechanisms: **stdio** for local development and **streamable-http** for production deployment. SSE (Server-Sent Events) transport is not supported. The 2026 stateless specification is

supported via streamable HTTP with client-provided session IDs (passed through the `operator` parameter) rather than server-side persistence, enabling horizontal scaling. A `StatelessVerifier` module validates six compliance checks: no session state, client session ID support, idempotent tools, no sticky routing, horizontal scaling compatibility, and transport compliance—all passing in the reference implementation.

4.3 Layer 2: Security and Compliance Layer Implementation

4.3.1 SecurityModule Implementation

The `SecurityModule` implements hierarchical PHI de-identification with three configurable levels as defined in Section 3.5.2.

```

1 import hashlib
2
3 class SecurityModule:
4     """Hierarchical PHI de-identification.
5
6     In the reference implementation, three fields are de-
7     identified:
8     PatientID (SHA-256 hash, truncated to 16 chars for storage
9     efficiency), PatientName (replaced with ANONYMIZED), and
10    StudyDate (replaced with REDACTED).
11    """
12
13    def deidentify(self, data: dict) -> dict:
14        result = data.copy()
15        # Patient ID: SHA-256 hash, truncated to 16 chars
16        hashed = self._hash_patient(data.get("PatientID", "
17        unknown"))
18        result["PatientID"] = f"sha256:{hashed[:16]}"
19        # Patient Name: explicit anonymization marker
20        result["PatientName"] = "ANONYMIZED"
21        # Study Date: redacted
22        result["StudyDate"] = "REDACTED"
23        return result
24
25    def _hash_patient(self, patient_id: str) -> str:
26        """Full 64-char SHA-256 hash; truncation applied by
27        caller."""
28        return hashlib.sha256(
29            patient_id.encode()).hexdigest()
30
31    def _hash_content(self, content: str) -> str:
32        """Full 64-char SHA-256 hash for content fingerprinting
33        ."""
34        return hashlib.sha256(
35            str(content).encode()).hexdigest()

```

Listing 17: SecurityModule Implementation (security_module.py)

Hash truncation rationale: The SHA-256 hash is truncated to 16 characters (64 bits) when stored in `PatientID` and audit log fields. At this length, the collision probability in a target patient population of up to 2^{32} is negligible (birthday bound $\approx 2^{-32}$), satisfying the irreversibility requirement of HIPAA §164.312(a)(1) while optimizing storage and index efficiency. The full 64-character hash is available internally via `_hash_patient` and `_hash_content` when needed for verification.

Design note: The reference implementation provides a simplified de-identification for validation purposes, covering `PatientID`, `PatientName`, and `StudyDate`. The protocol specification defines three configurable levels (patient, department, system) including `PatientBirthDate` age-binning and relative date formatting; full implementation of these features is planned for the production deployment.

4.3.2 AuditLogger Implementation

The AuditLogger captures the complete service lifecycle with cryptographic image fingerprinting.

```

1 import hashlib
2 import json
3 import sqlite3
4 from datetime import datetime
5 from uuid import uuid4
6
7 class AuditLogger:
8     def __init__(self, db_path: str = "audit.db"):
9         self.db_path = db_path
10        self._init_db()
11
12    def _init_db(self):
13        """Create 14-column audit table with independent fields
14        ."""
15        self.db.execute("""
16            CREATE TABLE IF NOT EXISTS audit_log (
17                id TEXT PRIMARY KEY,
18                timestamp TEXT NOT NULL,
19                operation TEXT,
20                operator TEXT,
21                patient_hash TEXT,
22                study_uid TEXT,
23                input_hash TEXT,
24                ai_model TEXT,
25                ai_output_hash TEXT,
26                verification_mode TEXT,
27                verification_action TEXT,
28                verification_reason TEXT,
29                finalized_by TEXT,
30                result TEXT,
31                regulatory_tags TEXT

```

```

32         """
33         self.db.commit()
34
35     def log_start(self, request: dict) -> str:
36         """Log service initiation and return audit_id."""
37         audit_id = f"AUDIT_{datetime.now().strftime('%Y%m%d%H%M%S')}_{{uuid4().hex[:8]}}"
38         self.log_operation(
39             operation="medical_image_analyze_start",
40             operator=request.get("operator", "system"),
41             patient_hash=self._hash_patient(request.get("
42 patient_id", "")),
43             study_uid=request.get("study_uid", ""),
44             input_hash=self._hash_dicom(request.get("dicom_data
45 ", "")),
46             result="IN_PROGRESS",
47         )
48         return audit_id
49
50     def log_complete(self, audit_id: str, ai_result: dict,
51 verification: dict) -> None:
52         """Log service completion with full audit trail."""
53         self.log_operation(
54             operation="medical_image_analyze_complete",
55             ai_model=ai_result.get("model_version", ""),
56             ai_output_hash=self._hash_content(ai_result.get("
57 output", "")),
58             verification_mode=verification.get("mode", ""),
59             verification_action=verification.get("action", ""),
60             verification_reason=verification.get("reason", ""),
61             finalized_by=verification.get("finalized_by", ""),
62             result="SUCCESS",
63             regulatory_tags=["FDA_TPLC", "NMPA_Grade2", "
64 EU_AI_Act"],
65         )
66
67     def log_operation(self, operation="", operator="system",
68 patient_hash="", study_uid="", input_hash
69 = "",
70 ai_model="", ai_output_hash="",
71 verification_mode="", verification_action
72 = "",
73 verification_reason="", finalized_by=None
74 ,
75 result="SUCCESS", regulatory_tags=None,
76 metadata=None) -> str:
77         """Generic audit logging for non-lifecycle operations.
78         """
79         audit_id = f"AUDIT_{datetime.now().strftime('%Y%m%d%H%M%S')}_{{uuid4().hex[:8]}}"
80         tags_json = json.dumps(regulatory_tags or [])
81         self.db.execute(
82             """INSERT INTO audit_log

```

```

75         (id, timestamp, operation, operator,
76         patient_hash,
77         study_uid, input_hash, ai_model, ai_output_hash
78         ,
79         verification_mode, verification_action,
80         verification_reason, finalized_by, result,
81         regulatory_tags)
82         VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?) """
83         (audit_id, datetime.now().isoformat(), operation,
84         operator, patient_hash, study_uid, input_hash,
85         ai_model, ai_output_hash, verification_mode,
86         verification_action, verification_reason,
87         finalized_by,
88         result, tags_json))
89         self.db.commit()
90         return audit_id
91
92     def _hash_patient(self, patient_id: str) -> str:
93         return hashlib.sha256(patient_id.encode()).hexdigest()
94         [:16]
95
96     def _hash_dicom(self, dicom_data) -> str:
97         return hashlib.sha256(str(dicom_data).encode()).
98         hexdigest()[:16]
99
100    def _hash_content(self, content) -> str:
101        if content is None:
102            return ""
103        return hashlib.sha256(str(content).encode()).hexdigest()
104        [:16]

```

Listing 18: AuditLogger Implementation (audit_logger.py)

Design decisions: (1) The audit table uses 14 independent columns rather than a single JSON blob, enabling efficient field-level queries (e.g., filtering by `study_uid` or `operator`) without full-table JSON parsing. (2) All SQL operations use parameterized queries (`?` placeholders) to prevent SQL injection. (3) The `log_operation` generic method supports non-lifecycle audit entries (e.g., `query`, `store`) that do not require the full start-complete lifecycle. (4) `regulatory_tags` are serialized as JSON strings within the `regulatory_tags` column for TPLC compliance. (5) WORM (Write Once Read Many) enforcement is recommended at the SQLite database level via PRAGMA settings for production deployment.

4.3.3 PolicyEngine Implementation

The PolicyEngine implements Role-Based Access Control (RBAC) and compliance rule enforcement.

```

1 import json
2
3 class PolicyEngine:

```

```

4     def __init__(self, policy_file: str = None):
5         self.policy = self._load_policy(policy_file)
6         self._default_roles = ["guest"] # Zero-permission
7         default
8
9     def _load_policy(self, policy_file):
10        if policy_file:
11            with open(policy_file) as f:
12                return json.load(f)
13        return {"permissions": {"guest": [], "radiologist": [
14            "query", "retrieve", "analyze", "verify", "store"],
15            "technician": ["query", "retrieve"],
16            "admin": ["*"]}}
17
18    def authorize(self, operator: str, operation: str,
19                  resource: dict = None) -> bool:
20        roles = self._get_user_roles(operator)
21        for role in roles:
22            perms = self.policy["permissions"].get(role, [])
23            if operation in perms or "*" in perms:
24                return True
25        return False
26
27    def get_verification_mode(self, indication: str,
28                             modality: str = "") -> str:
29        """Match verification rules by indication substring."""
30        rules = self.policy.get("verification_rules", [])
31        for rule in rules:
32            if indication in rule.get("indication", ""):
33                return rule.get("mode", "ADVISORY")
34        return "ADVISORY" # Default
35
36    def get_inspection_type_mode(self, inspection_type: str) ->
37    str:
38        """Match verification rules by exact inspection_type.
39        """
40        rules = self.policy.get("verification_rules", [])
41        for rule in rules:
42            if inspection_type == rule.get("inspection_type", "
43            "):
44                return rule.get("mode", "ADVISORY")
45        return "ADVISORY"
46
47    def _get_user_roles(self, operator: str) -> list:
48        return self.policy.get("users", {}).get(
49            operator, ["guest"]) # Default: zero-permission

```

Listing 19: PolicyEngine Implementation (policy_engine.py)

Security hardening — default role: The default role for unrecognized operators is `guest` (empty permission list), enforcing the principle of least privilege. This is a security-sensitive design decision: assigning `radiologist` as the default role would grant any unregistered operator full medical imaging analysis permissions, violating HIPAA §164.312(a)(1) Ac-

cess Control requirements. All operators must be explicitly registered in `policy.json` with appropriate roles before they can perform any operations.

The corresponding `policy.json` configuration file defines role permissions, verification rules, and user-role mappings:

```
1 {
2   "permissions": {
3     "guest": [],
4     "radiologist": ["query", "retrieve", "analyze", "verify", "store"],
5     "technician": ["query", "retrieve"],
6     "admin": ["*"]
7   },
8   "verification_rules": [
9     {"inspection_type": "LUNG_CANCER_SCREENING",
10      "indication": "screening", "modality": "CT",
11      "mode": "MANDATORY"},
12     {"inspection_type": "NODULE_FOLLOWUP",
13      "indication": "followup", "modality": "CT",
14      "mode": "AUTO"},
15     {"inspection_type": "BREAST_SCREENING",
16      "indication": "screening", "modality": "MG",
17      "mode": "MANDATORY"},
18     {"inspection_type": "BONE_AGE_ASSESSMENT",
19      "indication": "bone_age", "modality": "XR",
20      "mode": "ADVISORY"}
21  ],
22   "users": {
23     "dr_zhang": {"roles": ["radiologist"]},
24     "dr_li": {"roles": ["radiologist"]},
25     "tech_wang": {"roles": ["technician"]},
26     "admin_user": {"roles": ["admin"]}
27  }
28 }
```

Listing 20: `policy.json` Configuration

4.3.4 Input Sanitization and Sandbox

To mitigate MCP-38 threats [24, 25], MCP-MedImg implements input sanitization and sandboxed execution.

```
1 import re
2
3 class InputSanitizer:
4     def sanitize_dicom(self, dicom_data: dict) -> dict:
5         """Remove or escape potentially malicious content"""
6         sanitized = dicom_data.copy()
7         for field in ["PatientName", "StudyDescription",
8                     "SeriesDescription"]:
9             if field in sanitized:
10                 sanitized[field] = self._sanitize_text(
11                     sanitized[field])
```

```

12         return sanitized
13
14     def _sanitize_text(self, text: str) -> str:
15         """Strip executable patterns and control characters"""
16         if not isinstance(text, str):
17             return text
18         # Remove shell metacharacters
19         text = re.sub(r'[$|'<>]', '', text)
20         # Escape DICOM control sequences
21         return text.encode('ascii', errors='ignore').decode('

```

Listing 21: Input Sanitizer Implementation (sanitizer.py)

Containerized Sandbox:

```

1 version: '3.8'
2 services:
3   orthanc:
4     image: orthancteam/orthanc:latest
5     container_name: mcp-meding-orthanc
6     ports:
7       - "8042:8042"      # DICOMweb REST API
8       - "4242:4242"      # DICOM C-MOVE/SCP
9     environment:
10      - DICOM_WEB_PLUGIN_ENABLED=true
11      - ORTHANC__DICOM_WEB__ENABLE=true
12      - ORTHANC__DICOM_WEB__ROOT=/dicom-web/
13      - ORTHANC__REMOTE_ACCESS_ALLOWED=true
14      - ORTHANC__AUTHENTICATION_ENABLED=false
15     volumes:
16      - ./data/orthanc-db:/var/lib/orthanc/db
17     restart: unless-stopped

```

Listing 22: docker-compose.yml (Reference Implementation)

Deployment note: The reference implementation uses a single Orthanc container with DICOMweb plugin enabled and authentication disabled (for development convenience; production deployments should enable authentication). AI inference is performed in-process by the `MockNNUNetEngine` within the Python gateway, not in a separate container. Production deployment with real nnU-Net would add a GPU-enabled inference container with `nvidia/cuda` base image and `seccomp` profile for syscall restriction.

Command Whitelist: The gateway maintains a whitelist of allowed MCP tool commands, rejecting any request that attempts to execute system commands outside the defined set.

Tool Description Signature Verification: To prevent semantic attack surface exploitation [24], the `ModelRegistry` verifies cryptographic signatures on tool descriptions to ensure they have not been tampered with.

4.4 Layer 3: AI Orchestration Layer Implementation

4.4.1 ModelRegistry Implementation

The ModelRegistry provides model discovery and metadata exposure following the KServe Open Inference Protocol pattern [35].

```
1 class ModelRegistry:
2     def __init__(self):
3         self.models = {} # model_id -> metadata (public
4                             attribute)
5
6         REQUIRED_FIELDS = ["modality", "bodyPart", "indications",
7                             "performance", "regulatory"]
8
9     def register(self, model_id: str, metadata: dict) -> None:
10         """Register a new AI model with full metadata."""
11         for field in self.REQUIRED_FIELDS:
12             if field not in metadata:
13                 raise ValueError(f"Missing required field: {
14 field}")
15             self.models[model_id] = metadata
16
17     def discover(self, modality=None, body_part=None,
18                 indication=None) -> list:
19         """Discover models matching criteria."""
20         results = []
21         for model_id, metadata in self.models.items():
22             if modality and modality not in metadata.get("
23 modality", []):
24                 continue
25             if body_part and body_part not in metadata.get("
26 bodyPart", []):
27                 continue
28             if indication and indication not in metadata.get(
29 "indications", []):
30                 continue
31             results.append({"id": model_id, "metadata":
32 metadata})
33         return results
34
35     def get_model(self, model_id: str) -> dict:
36         return {"id": model_id,
37                 "metadata": self.models.get(model_id)}
38
39     def list_models(self) -> list:
40         """List all registered models (excluding engine objects
41 )."""
42         return [{"id": mid, **{k: v for k, v in m.items()
43 if k != "engine"}}
44 for mid, m in self.models.items()]
```

Listing 23: ModelRegistry Implementation (model_registry.py)

The ModelRegistry uses camelCase field naming (e.g., bodyPart) for consistency with DICOM tag conventions. The performance sub-dictionary typically contains AUC and Sensitivity metrics; inference_time_ms may optionally be included for STAT routing.

4.4.2 AIRouter Implementation

The AIRouter implements inspection-type-driven model selection.

```

1 class AIRouter:
2     def __init__(self, registry: ModelRegistry):
3         self.registry = registry
4
5     def route(self, modality, body_part, indication,
6               priority="ROUTINE"):
7         """Route to optimal AI model based on inspection type.
8         """
9         candidates = self.registry.discover(
10             modality=modality, body_part=body_part,
11             indication=indication)
12         if not candidates:
13             return None
14
15         if priority == "STAT":
16             # STAT: sort by inference time only (single
17             # objective)
18             candidates.sort(key=lambda m: m["metadata"].get(
19                 "inference_time_ms", 9999))
20         else:
21             # ROUTINE: sort by AUC descending
22             candidates.sort(key=lambda m: m["metadata"].get(
23                 "performance", {}).get("AUC", 0), reverse=True)
24         return candidates[0]
25
26     def infer(self, model_id: str, study_data: dict) -> dict:
27         model = self.registry.get_model(model_id)
28         if not model:
29             return {"error": {"code": "MODEL_UNAVAILABLE",
30                               "message": f"Model {model_id} not found"},
31                     "result": None}
32         engine = model["metadata"].get("engine")
33         if engine:
34             result = engine.infer(study_data)
35             return {"error": None, "result": result,
36                     "model_version": model["metadata"].get(
37                         "version", "unknown")}
38         return {"error": {"code": "NO_ENGINE"}, "result": None}

```

Listing 24: AIRouter Implementation (ai_router.py)

Routing logic: For ROUTINE priority, models are ranked by AUC (Area Under the ROC Curve) in descending order. For STAT (urgent) priority, models are ranked by inference_time_ms alone—this is a single-objective

optimization prioritizing speed over accuracy, as urgent cases require the fastest available response. The router does not currently implement load balancing or concurrency limiting; these are planned for production deployment.

4.4.3 VersionManager Implementation

The VersionManager supports canary release and A/B testing.

```

1 class VersionManager:
2     def __init__(self):
3         self.models = {} # model_name -> {version: metadata}
4
5     def canary_release(self, model_name: str, new_version: str,
6                       canary_percentage: int = 10) -> None:
7         """Release new version to a percentage of traffic"""
8         if model_name not in self.models:
9             self.models[model_name] = {}
10        self.models[model_name][new_version] = {
11            "status": "canary",
12            "canary_percentage": canary_percentage
13        }
14
15    def get_version_for_request(self, model_name: str,
16                              patient_hash: str) -> str:
17        """Determine version using consistent hashing"""
18        versions = self.models.get(model_name, {})
19        if not versions:
20            return None
21        if len(versions) == 1:
22            return list(versions.keys())[0]
23
24        # Consistent hashing for canary routing
25        hash_val = int(patient_hash[:8], 16) % 100
26        canary_pct = versions.get("canary", {}).get("
27        canary_percentage", 0)
28        if hash_val < canary_pct:
29            return "canary"
30        return "stable"

```

Listing 25: VersionManager Implementation (version_manager.py)

4.4.4 ResultFormatter Implementation

The ResultFormatter generates standardized DICOM SR and SEG outputs. The reference implementation provides two complementary modules: ResultFormatter for lightweight dict-based SR/SEG document generation, and DICOMSRWriter for full pydicom Dataset construction with STOW-RS storage to PACS.

```

1 from pydicom.uid import generate_uid
2

```

```

3 class ResultFormatter:
4     def to_dicom_sr(self, findings, study_info, model_info) ->
      dict:
5         """Generate Basic Text SR (SOP Class
          1.2.840.10008.5.1.4.1.1.88.22)
          as a dictionary representation."""
6         sr_document = {
7             "sopClassUID": "1.2.840.10008.5.1.4.1.1.88.22",
8             "sopInstanceUID": generate_uid(),
9             "studyInstanceUID": study_info.get("study_uid", "")
10        ,
11            "seriesInstanceUID": generate_uid(),
12            "model_name": model_info.get("name", "nnU-Net"),
13            "model_version": model_info.get("version", "v2"),
14            "findings": findings,
15            "verification_flag": "UNVERIFIED",
16        }
17        return sr_document
18
19     def to_dicom_seg(self, segmentation_mask, source_images,
20                      segment_description) -> dict:
21         """Generate Segmentation Storage document
          (SOP Class 1.2.840.10008.5.1.4.1.1.66.4) as dict."""
22         seg_doc = {
23             "sopClassUID": "1.2.840.10008.5.1.4.1.1.66.4",
24             "sopInstanceUID": generate_uid(),
25             "source_images": source_images,
26             "segment_label": segment_description.get("label", "
27 Lesion"),
28             "algorithm_name": segment_description.get(
29                 "algorithm_name", "nnU-Net"),
30         }
31         return seg_doc
32
33     def to_fhir_observation(self, findings, study_info) -> dict
      :
34         """Generate FHIR Observation resource."""
35         return {"resourceType": "Observation",
36                 "status": "preliminary",
37                 "findings": findings}

```

Listing 26: ResultFormatter: SR and SEG Generation (result_formatter.py)

```

1 from pydicom.dataset import Dataset, Sequence
2 from pydicom.uid import generate_uid
3 from highdicom.sr import CodedConcept
4 import highdicom as hd
5
6 class DICOMSRWriter:
7     """Builds Comprehensive 3D SR (SOP Class
          1.2.840.10008.5.1.4.1.1.88.34)
          using pydicom Dataset with manually constructed
          ContentSequence."""
8

```

```

9
10 CODE_LUNG_MODULE = CodedConcept(
11     value="77089001", scheme_designator="SCT",
12     meaning="Nodule of lung")
13
14 def _build_sr_dataset(self, findings, study_info,
15 model_info):
16     ds = Dataset()
17     ds.SOPClassUID = "1.2.840.10008.5.1.4.1.1.88.34"
18     ds.SOPInstanceUID = generate_uid()
19     # Patient and Study module attributes
20     ds.PatientID = study_info.get("patient_id", "ANONYMIZED")
21     ds.PatientName = "ANONYMIZED"
22     ds.StudyInstanceUID = study_info["study_uid"]
23     ds.SeriesInstanceUID = generate_uid()
24     # SR Document Content Module
25     ds.Modality = "SR"
26     ds.ValueType = "CONTAINER"
27     ds.ContentTemplateSequence = [Dataset()]
28     ds.ContentTemplateSequence[0].MappingResource = "DCMR"
29     ds.ContentTemplateSequence[0].TemplateIdentifier = "
2010"
30     # Build nested ContentSequence tree
31     content_tree = self._build_content_tree(findings,
32 model_info)
33     ds.ContentSequence = Sequence(
34         self._build_content_sequence_traversal(content_tree)
35     ))
36     return ds
37
38 def _build_content_tree(self, findings, model_info):
39     tree = []
40     # Algorithm metadata (TEXT)
41     tree.append({"ValueType": "TEXT",
42         "TextValue": model_info.get("name", "nnU-Net")})
43     # Each finding as CONTAINER
44     for f in findings:
45         tree.append({"ValueType": "CONTAINER",
46             "RelationshipType": "CONTAINS",
47             "ContentSequence": [
48                 {"ValueType": "CODE",
49                     "ConceptCodeSequence": [{"CodeValue": f["
type"]}]}],
50                 {"ValueType": "NUM",
51                     "MeasuredValueSequence": [{"NumericValue":
52 f["size_mm"]}]}],
53             ]})
54     return tree
55
56 def _build_content_sequence_traversal(self, tree):
57     """Recursively build pydicom ContentSequence items."""
58     items = []
59     for node in tree:

```

```

56         item = Dataset()
57         item.RelationshipType = node.get("RelationshipType"
)
58         item.ValueType = node.get("ValueType")
59         if "TextValue" in node:
60             item.TextValue = node["TextValue"]
61         if "ContentSequence" in node:
62             item.ContentSequence = Sequence(
63                 self._build_content_sequence_traversal(
64                     node["ContentSequence"]))
65         items.append(item)
66     return Sequence(items)

```

Listing 27: DICOMSRWriter: Comprehensive 3D SR with pydicom (sr_writer.py)

Two-module design rationale: The `ResultFormatter` returns lightweight dictionary representations (Basic Text SR, SOP Class 88.22) suitable for protocol-level interchange and testing. The `DICOMSRWriter` constructs full pydicom `Dataset` objects (Comprehensive 3D SR, SOP Class 88.34) with nested `ContentSequence` trees, enabling STOW-RS storage to PACS. The Comprehensive 3D SR class is required because the content tree includes CONTAINER nodes with nested CODE and NUM children—Basic Text SR only supports flat text content.

highdicom usage: The reference implementation imports `highdicom` for `CodedConcept` constant definitions only. The SR document is built using pydicom `Dataset` manually rather than `highdicom`'s high-level API, because the `highdicom` 0.28 API signatures (e.g., `hd.AlgorithmIdentificationSequence`, `hd.seg.Segmentation`) differ from the originally proposed code. The manual construction approach provides equivalent functionality with greater control over the `ContentSequence` structure.

4.4.5 nnU-Net Integration

nnU-Net automatically configures network topology, preprocessing, and postprocessing based on dataset properties. The reference implementation provides two engine implementations: `nnU-NetEngine` for production GPU inference and `MockNNUNetEngine` for local validation without GPU.

```

1 import numpy as np
2 from nnU-Netv2.inference.predict_from_raw_data import nnU-
  NetPredictor
3
4 class nnU-NetEngine:
5     def __init__(self, model_path: str, modality: str = "CT"):
6         self.model_path = model_path
7         self.modality = modality
8         self.predictor = nnU-NetPredictor(
9             tile_step_size=0.5,

```

```

10         use_gaussian=True,
11         perform_everything_on_device=True,
12     )
13     self.predictor.initialize_from_trained_model_folder(
14         model_path, use_folds=(0,))
15
16     def infer(self, image: np.ndarray, spacing: tuple) -> np.
17     ndarray:
18         """Run nnU-Net inference on image"""
19         return self.predictor.predict_from_raw_data(
20             data=image, spacing=spacing,
21             do_tta=False, # No test-time augmentation for
22             speed
23             verbose=False)

```

Listing 28: nnU-Net Inference Engine (nnU-Net_integration.py)

```

1 import numpy as np
2 import random
3
4 class MockNNUNetEngine:
5     """Simulates nnU-Net inference for local validation without
6     GPU.
7     Returns realistic mock findings with simulated latency."""
8     def infer(self, study_data: dict) -> dict:
9         rng = np.random.default_rng(seed=hash(
10             str(study_data)) % 2**32)
11         num_findings = rng.integers(0, 4)
12         findings = []
13         for _ in range(num_findings):
14             findings.append({
15                 "type": "nodule",
16                 "size_mm": round(rng.uniform(3, 15), 1),
17                 "confidence": round(rng.uniform(0.7, 0.99), 3),
18                 "location": "lower_right_lobe",
19             })
20         return {
21             "findings": findings,
22             "inference_time_ms": rng.randint(150, 400),
23             "model_version": "nnU-Net-Mock-v1",
24         }

```

Listing 29: MockNNUNetEngine for Local Validation (mock_nnU-Net_engine.py)

GPU Optimizations:

- Mixed precision (FP16) for faster inference
- Batch processing for multiple studies
- CUDA Graph for reduced kernel launch overhead

4.5 Layer 4: Application Semantics Layer Implementation

4.5.1 InspectionRouter Implementation

The InspectionRouter maps DICOM metadata to clinical inspection types.

```
1 class InspectionRouter:
2     INSPECTION_MAP = [
3         {"modality": "CT", "body_part": "CHEST",
4          "indication": "screening", "type": "
5         LUNG_CANCER_SCREENING"},
6         {"modality": "CT", "body_part": "CHEST",
7          "indication": "followup", "type": "NODULE_FOLLOWUP"},
8         {"modality": "MG", "body_part": "BREAST",
9          "indication": "screening", "type": "BREAST_SCREENING"
10        },
11        {"modality": "XR", "body_part": "HAND",
12         "indication": "bone_age", "type": "BONE_AGE_ASSESSMENT"
13        },
14    ],
15
16    def route(self, modality: str, body_part: str,
17              indication: str) -> str:
18        for rule in self.INSPECTION_MAP:
19            if (rule["modality"] == modality and
20                rule["body_part"] == body_part and
21                indication in rule["indication"]):
22                return rule["type"]
23        return "GENERAL"
```

Listing 30: InspectionRouter Implementation (inspection_router.py)

4.5.2 LesionEncoder Implementation

The LesionEncoder implements standardized grading systems.

```
1 class LesionEncoder:
2     def encode_rads(self, nodule_size: float,
3                    nodule_type: str) -> str:
4         """Lung-RADS classification (version 1.1)"""
5         if nodule_size < 6:
6             return "Lung-RADS 1"
7         elif nodule_size < 10:
8             if nodule_type == "solid":
9                 return "Lung-RADS 2"
10            else:
11                return "Lung-RADS 3"
12        else:
13            return "Lung-RADS 4"
14
15    def encode_birads(self, mass_shape: str, mass_margin: str,
16                     calcification: str) -> str:
17        """BI-RADS classification (breast)"""
18        if mass_shape == "round" and mass_margin == "
19        circumscribed":
```

```

19         return "BI-RADS 2"
20     elif mass_shape == "irregular" or mass_margin == "
    spiculated":
21         return "BI-RADS 4"
22     else:
23         return "BI-RADS 3"

```

Listing 31: LesionEncoder Implementation (lesion_encoder.py)

4.5.3 VerificationRouter Implementation

The VerificationRouter implements the conditional verification mechanism.

```

1 class VerificationRouter:
2     def __init__(self, policy_engine=None):
3         self.policy_engine = policy_engine
4         self.deployment_tracker = {} # model_id -> {case_count
    , deploy_time}
5
6     def determine_mode(self, inspection_type, priority,
7                        model_version=None, model_id=None):
8         """Determine verification mode based on context.
9         Returns dict with 'mode' and 'reason' keys."""
10        # 1. STAT priority: AUTO for speed
11        if priority == "STAT":
12            return {"mode": "AUTO",
13                   "reason": "STAT priority auto-verifies for
    speed"}
14
15        # 2. New model deployment: MANDATORY for warmup period
16        if model_id and self._is_new_model_deployment(model_id)
    :
17            return {"mode": "MANDATORY",
18                   "reason": "New model deployment warmup "
19                           "(first N cases)"}
20
21        # 3. Policy lookup by inspection_type
22        if self.policy_engine:
23            mode = self.policy_engine.get_inspection_type_mode(
24                inspection_type)
25            if mode:
26                return {"mode": mode, "reason": f"Policy for "
27                        f"{inspection_type}"}
28
29        return {"mode": "ADVISORY", "reason": "Default advisory
    "}
30
31    def _is_new_model_deployment(self, model_id,
32                                warmup_cases=50):
33        """Check if model is in warmup period.
34        Models must be registered via register_deployment()
35        before this check is meaningful."""
36        info = self.deployment_tracker.get(model_id)
37        if info is None:

```

```

38         return False # Unregistered models don't trigger
warmup
39         return info.get("case_count", 0) < warmup_cases
40
41     def register_deployment(self, model_id):
42         self.deployment_tracker[model_id] = {
43             "case_count": 0,
44             "deploy_time": datetime.now().isoformat()}
45
46     def record_case(self, model_id):
47         if model_id in self.deployment_tracker:
48             self.deployment_tracker[model_id]["case_count"] +=
1
49
50     def get_workflow(self, mode):
51         """Return workflow config with timeout_seconds."""
52         timeouts = {"MANDATORY": 300, "ADVISORY": 86400, "AUTO"
: 0}
53         return {"timeout_seconds": timeouts.get(mode, 300)}

```

Listing 32: VerificationRouter Implementation (verification_router.py)

Warmup period: The `warmup_cases` threshold is 50 by default—newly deployed models require mandatory human verification for their first 50 inference cases. The threshold is hardcoded in the method parameter and does not come from ModelRegistry metadata. Models must be explicitly registered via `register_deployment()` before the warmup check takes effect; unregistered models are treated as already past warmup.

4.5.4 Verification State Machine

The `VerificationStateMachine` implements the workflow defined in Section 3.4.4.

```

1 from datetime import datetime
2
3 class VerificationStateMachine:
4     STATES = ["PENDING", "UNDER_REVIEW", "REVIEWED",
5             "MODIFIED", "CONFIRMED", "FINALIZED", "TIMEOUT"]
6
7     def __init__(self, verification_id: str,
8                 timeout_seconds: int = 300):
9         self.verification_id = verification_id
10        self.state = "PENDING"
11        self.timeout_seconds = timeout_seconds
12        self.start_time = datetime.now()
13        self.history = [{"state": "PENDING",
14                        "timestamp": self.start_time.isoformat
15                        ()}]
16
17    def transition_to(self, new_state: str, reason: str = None)
:
        if new_state not in self.STATES:

```

```

18         raise ValueError(f"Invalid state: {new_state}")
19
20     valid_transitions = {
21         "PENDING": ["UNDER_REVIEW"],
22         "UNDER_REVIEW": ["REVIEWED", "TIMEOUT"],
23         "REVIEWED": ["CONFIRMED", "MODIFIED"],
24         "MODIFIED": ["CONFIRMED"],
25         "CONFIRMED": ["FINALIZED"],
26     }
27     if new_state not in valid_transitions.get(self.state,
28 []):
29         raise ValueError(
30             f"Invalid transition: {self.state} -> {
31 new_state}")
32
33     self.state = new_state
34     self.history.append({
35         "state": new_state,
36         "timestamp": datetime.now().isoformat(),
37         "reason": reason})
38
39     if new_state == "TIMEOUT":
40         self._escalate()
41
42     def _escalate(self):
43         """Escalate to supervisor on timeout.
44         Reference implementation: records escalation in history
45         log.
46         Production: send notification + create escalation
47         ticket."""
48         self.history.append({
49             "state": "ESCALATED",
50             "timestamp": datetime.now().isoformat(),
51             "reason": "Verification timeout - escalated to
52 supervisor"})
53
54     def is_terminal(self) -> bool:
55         return self.state in ["FINALIZED", "TIMEOUT"]
56
57     def get_history(self) -> list:
58         return self.history

```

Listing 33: Verification State Machine (verification_state_machine.py)

Timeout thresholds: Timeout values are defined in `VerificationRouter.get_workflow()`: MANDATORY mode uses 300 seconds (5 minutes), ADVISORY mode uses 86,400 seconds (24 hours), and AUTO mode uses 0 seconds (immediate finalization). These values are hardcoded in the reference implementation; production deployment may load them from `policy.json`.

4.6 MCP-MedImg Gateway Integration

4.6.1 Gateway Architecture

The Gateway orchestrates all four layers as a unified MCP Server:

```
1 from mcp.server.fastmcp import FastMCP
2 from mcp_medimg.transport import PACSClient
3 from mcp_medimg.security import SecurityModule, AuditLogger
4 from mcp_medimg.orchestration import (ModelRegistry, AIRouter,
5   ResultFormatter)
6 from mcp_medimg.application import (InspectionRouter,
7   VerificationRouter, VerificationStateMachine, LesionEncoder
8   )
9
10 mcp = FastMCP("MCP-MedImg Gateway")
11
12 @mcp.tool()
13 def medical_image_analyze(
14     studyInstanceUID: str,
15     seriesInstanceUID: str = "",
16     modality: str = "CT",
17     bodyPart: str = "CHEST",
18     clinicalIndication: str = "screening",
19     priority: str = "ROUTINE",
20     aiModels: list = None,          # User-specified model override
21     outputFormat: str = "DICOM_SR",
22     verificationMode: str = "ADVISORY",
23     operator: str = "system",
24     dicom_data: dict = None,
25 ) -> dict:
26     """Analyze a DICOM study with AI-assisted diagnosis.
27     Execution order: L1 -> L2 -> L4(mode) -> L3(route+infer+
28     format)
29     -> L4(encode+verify) -> L2(audit) -> L1(
30     store)
31     """
32     result = {"studyInstanceUID": studyInstanceUID,
33              "status": "PROCESSING", "errors": [],
34              "layers_executed": []}
35
36     # === L1: Retrieve study data ===
37     if dicom_data is None:
38         studies = self.pacs.query_studies(modality)
39         dicom_data = studies[0] if studies else {}
40     result["layers_executed"].append("L1_retrieve")
41
42     # === L2: De-identify + audit start ===
43     deidentified = self.security.deidentify(dicom_data)
44     audit_id = self.audit.log_start({"operator": operator,
45                                     "study_uid": studyInstanceUID, "dicom_data": dicom_data
46 })
47     result["layers_executed"].append("L2_deidentify_audit_start
48 ")
```

```

45 # === L4: Determine inspection type + verification mode ===
46 # (BEFORE L3: ensures verification is set even without
model)
47 inspection_type = self.inspection.route(
48     modality, bodyPart, clinicalIndication)
49 ver_result = self.verification.determine_mode(
50     inspection_type, priority,
51     model_id=aiModels[0] if aiModels else None)
52 result["layers_executed"].append("L4_verification_mode")
53
54 # === L3: Route to AI model + infer + format ===
55 if aiModels:
56     model_id = aiModels[0]
57     model = self.registry.get_model(model_id)
58 else:
59     model = self.router.route(modality, bodyPart,
60         clinicalIndication, priority)
61     model_id = model["id"] if model else None
62 if model_id and model:
63     infer_result = self.router.infer(model_id, study_data)
64     if outputFormat == "DICOM_SR":
65         formatted = self.formatter.to_dicom_sr(
66             ai_findings, study_info, model_info)
67     elif outputFormat == "FHIR":
68         formatted = self.formatter.to_fhir_observation(
69             ai_findings, study_info)
70     result["layers_executed"].append("L3_route_infer")
71     result["layers_executed"].append("L3_format")
72 else:
73     result["errors"].append({"code": "MODEL_UNAVAILABLE"})
74
75 # === L4: Encode lesions + apply verification ===
76 if inspection_type in ("LUNG_CANCER_SCREENING", "
MODULE_FOLLOWUP"):
77     encoded = self.lesion_encoder.encode_rads_from_findings
(
78         ai_findings)
79     elif inspection_type == "BREAST_SCREENING":
80         encoded = self.lesion_encoder.
encode_birads_from_findings(
81             ai_findings)
82
83     vsm = VerificationStateMachine(audit_id, timeout_seconds=
ver_result.get("timeout_seconds", 300))
84     if ver_result["mode"] == "AUTO":
85         vsm.transition_to("UNDER_REVIEW")
86         vsm.transition_to("REVIEWED")
87         vsm.transition_to("CONFIRMED")
88         vsm.transition_to("FINALIZED")
89         result["verification_action"] = "AUTO_FINALIZED"
90     elif ver_result["mode"] == "MANDATORY":
91         vsm.transition_to("UNDER_REVIEW")
92         result["verification_action"] = "
PENDING_RADIOLOGIST_REVIEW"

```

```

93     else: # ADVISORY
94         vsm.transition_to("UNDER_REVIEW")
95         result["verification_action"] = "
ADVISORY_REVIEW_OPTIONAL"
96         result["layers_executed"].append("L4_verification")
97
98     # === L2: Audit complete ===
99     self.audit.log_complete(audit_id, ai_result, verification)
100    result["layers_executed"].append("L2_audit_complete")
101
102    # === L1: Store result ===
103    result["store_status"] = "stored_mock"
104    result["layers_executed"].append("L1_store")
105    return result

```

Listing 34: Gateway Core: medical_image_analyze (gateway.py)

Key architectural decision — L4 verification mode before L3:

The verification mode is determined *before* AI model routing. This ensures that even when no matching model is available (e.g., CT/Brain with no brain model registered), the verification workflow is still triggered. For STAT priority, this guarantees that AUTO verification is set regardless of model availability—a critical safety property for emergency scenarios where the absence of a model should not block the verification pipeline.

4.6.2 MCP Server Resources and Prompts

The MCP Server registers two Resources and two Prompts to expose audit trails and model registries, and to guide clinical workflows:

```

1 mcp = FastMCP("MCP-MedImg Gateway")
2
3 @mcp.resource("audit://trails/{study_uid}")
4 def get_audit_resource(study_uid: str) -> str:
5     """Retrieve the audit trail for a specific study."""
6     gw = get_gateway()
7     entry = gw.get_audit_trail(study_uid=study_uid)
8     return json.dumps(entry, indent=2, default=str,
9                        ensure_ascii=False)
10
11 @mcp.resource("models://registry")
12 def get_models_resource() -> str:
13     """List all registered AI models as a resource."""
14     gw = get_gateway()
15     models = gw.list_registered_models()
16     return json.dumps(models, indent=2, default=str,
17                       ensure_ascii=False)
18
19 @mcp.prompt()
20 def lung_screening_protocol(studyInstanceUID: str,
21                             operator: str = "radiologist") -> str:
22     """Guide lung cancer screening analysis workflow."""
23     return f"""You are performing a lung cancer screening

```

```

24     analysis using MCP-MedImg.
25     Study Instance UID: {studyInstanceUID}
26     Operator: {operator}
27     Clinical Indication: Lung cancer screening (LDCT Chest)
28     Priority: ROUTINE
29     Expected Verification: MANDATORY (radiologist review
30     required)
31     Steps: call medical_image_analyze tool with modality=CT,
32     bodyPart=CHEST, clinicalIndication=screening,
33     outputFormat=DICOM_SR, then query audit trail.""
34 @mcp.prompt()
35 def breast_screening_protocol(studyInstanceUID: str,
36     operator: str = "radiologist") -> str:
37     """Guide breast mammography screening workflow."""
38     return f"""You are performing a breast mammography
39     screening
40     analysis using MCP-MedImg.
41     Study Instance UID: {studyInstanceUID}
42     Operator: {operator}
43     Clinical Indication: Breast cancer screening (MG)
44     Steps: call medical_image_analyze with modality=MG,
45     bodyPart=BREAST, clinicalIndication=\allowbreak screening.
46     """

```

Listing 35: MCP Resources and Prompts (server.py)

Resource URI schemes: The reference implementation uses `audit://trails/{study_uid}` for audit trail access and `models://registry` for model listing. These differ from the originally proposed `medical-ai://` scheme to provide clearer semantic namespacing. A `policies://` resource for institutional policy inspection is planned but not yet implemented.

Prompt design: The two implemented prompts (`lung_screening_protocol` and `breast_screening_protocol`) return plain-text workflow guidance strings with specific parameter values and step-by-step instructions. The originally proposed `radiology_report_assist` and `verification_guidance` prompts are planned for future implementation with LLM-driven report drafting capabilities.

5 Validation

5.1 Validation Objectives

1. **Protocol consistency:** Tool/Resource/Prompt definitions comply with MCP specifications (including 2026 stateless spec).
2. **End-to-end workflow:** Complete PACS → AI → conditional verification → result writeback.

3. **Multi-model routing:** Correct dispatch of different models for different inspection types.
4. **Result standardization:** DICOM SR/SEG compliance with standard validators.
5. **Service auditability:** Complete audit trail capturing AI inference and human verification actions (when applicable).
6. **Conditional verification:** Verification modes (mandatory/advisory/auto) correctly applied per institutional policy.
7. **Regulatory TPLC support:** Audit log generation for FDA/NMPA/EU post-market surveillance.

5.2 Validation Scenarios

5.3 Expected Validation Results

5.4 Performance Evaluation

5.4.1 Experimental Setup

To validate end-to-end latency under realistic clinical conditions, we conducted GPU inference benchmarking on two complementary datasets: (1) the LIDC-IDRI lung nodule CT dataset [47] for high-volume patch-based inference; and (2) the MSD Task02 Heart cardiac MRI dataset [48] for full nnU-Netv2 ensemble inference with trained weights.

LIDC-IDRI (Lung Nodule CT). This dataset represents a challenging real-world scenario with variable volume sizes (up to $512 \times 512 \times 400$ voxels), high-resolution thin-slice CT acquisition, and the need for patch-based processing when volumes exceed GPU memory capacity.

- **Dataset:** LIDC-IDRI [47] (publicly available lung nodule CT database)
- **Test samples:** 200 consecutive CT volumes (NIfTI format)
- **GPU:** NVIDIA GeForce RTX 4090 D (24 GB VRAM)
- **CUDA:** 12.1, PyTorch: 2.3.0
- **Model:** 3D U-Net with 22.6M parameters (`in_channels=1`, `out_channels=2`, `base_channels=32`)³
- **Inference strategy:** Direct forward for volumes $\leq 128^3$; sliding-window patch-based inference (128^3 patch, 64^3 stride) for larger volumes

³Model weights are randomly initialized; forward-pass latency is weight-independent. The benchmark targets protocol-layer throughput, not model accuracy.

Table 16: Validation Scenarios

Scenario	Input	Expected Output	Validation Point
Lung nodule screening	LDCT Chest	DICOM SR (RADS 2-4) + SEG	Correct routing, standard format
Breast mam-mography	MG Breast	DICOM SR (BI-RADS 0-6) + SEG	Model switching, correct grading
Emergency CT	CT Brain (STAT)	Priority dispatch, fast return	Priority queue
PHI de-identification	DICOM with patient name	Hashed ID, age bin	Security compliance
Historical comparison	2025 + 2026 studies	Auto-match, VDT	Longitudinal analysis
Mandatory verification	High-risk screening	Required radiologist review	Policy enforcement
Advisory verification	Routine diagnostic	Optional review with override	Flexible workflow
Auto verification	Low-risk follow-up	Direct finalization	Efficiency optimization
Service audit	100 consecutive studies	Complete audit log export	TPLC report generation
dicom-mcp compatibility	dicom-mcp or native fallback	Seamless Layer 1 with auto-fallback	Upward compatibility

Table 17: Expected Validation Metrics

Metric	Target	Note
Protocol consistency	100%	Verified through MCP SDK v1.28.1 (constraint: $mcp \geq 0.2.76$, < 2.0)
End-to-end latency	5.4s (LIDC mean) / 18.2s (Heart mean)	Real GPU inference on RTX 4090D (Section 5.3)
Routing accuracy	100%	4 inspection types $\times$ 3 models
DICOM SR compliance	Pass dcmstk validation	Comprehensive 3D SR (SOP Class 88.34)
PHI de-identification irreversibility	Pass	SHA-256 hash (16-char truncation)
Audit trail completeness	100%	All AI + human actions logged (14-column table)
Verification policy accuracy	100%	Correct mode per institutional policy
dicom-mcp compatibility	Pass	Auto-fallback to native DICOMweb when unavailable
Stateless deployment	Pass	6 compliance checks via StatelessVerifier
Test coverage	132 passed, 3 skipped ^a	6 test files, 8.65s total (protocol tests)
GPU inference benchmark	LIDC: 200 samples, Heart: 10 test samples	RTX 4090D, real end-to-end latency (Section 5.3)
Heart 5-fold Dice	0.9324 (± 0.0062)	nnU-Netv2 3D fullres ensemble (Section 5.4)

^aThe 3 skipped tests (2 PACS connection/query tests in `test_basic.py` and 1 live PACS SR storage test in `test_sr_writer.py`) were skipped because Orthanc PACS was not running during this test session. These are environment-dependent integration tests; all protocol logic tests passed.

- **Measurement:** End-to-end latency from DICOM load through result formatting, including file I/O, HU normalization, GPU inference, and JSON serialization

MSD Task02 Heart (Cardiac MRI Segmentation). This dataset enables validation with fully trained nnU-Netv2 weights, representing a complete clinical deployment scenario including 5-fold cross-validation ensemble inference.

- **Dataset:** MSD Task02 Heart [48] (20 training cases, left atrium segmentation)
- **Test samples:** 10 test volumes from the MSD Task02 Heart dataset (NIfTI format)
- **GPU:** NVIDIA GeForce RTX 4090 D (24 GB VRAM)
- **Model:** nnU-Netv2 3D full-resolution with 5-fold ensemble (trained weights, mean validation Dice: 0.9324)
- **Inference strategy:** Full nnU-Netv2 pipeline with automatic preprocessing, GPU inference, and postprocessing
- **Measurement:** End-to-end latency including model initialization, file I/O, automatic preprocessing, 5-fold ensemble GPU inference, and result export

5.4.2 Latency Results

Table 18 presents the measured end-to-end latency distribution across 200 LIDC-IDRI samples. Table 19 presents the MSD Heart cardiac MRI results with trained nnU-Netv2 ensemble weights. The two datasets demonstrate complementary latency characteristics.

Table 18: Measured End-to-End Latency (LIDC-IDRI, $n=200$, RTX 4090D)

Metric	Mean	P50	P99	Min	Max	Std
File loading (ms)	1,158	853	3,561	584	4,083	651
Preprocessing (ms)	869	628	2,906	399	3,768	549
GPU inference (ms)	3,374	2,407	16,237	134	18,431	3,515
Result formatting (ms)	0.3	0.3	0.8	0.2	1.1	0.1
End-to-end (ms)	5,401	3,876	22,615	1,138	26,282	4,678

Table 19: Measured End-to-End Latency (MSD Heart, $n=10$, RTX 4090D, 5-Fold Ensemble)

Metric	Mean	P50	P99	Min	Max	Std
File loading (ms)	362	297	745	190	745	196
Preprocessing + inference (ms)	17,839	18,828	27,252	12,812	27,791	4,693
Result formatting (ms)	0.0	0.0	0.0	0.0	0.0	0.0
End-to-end (ms)	18,200	19,097	27,602	13,002	28,103	4,787

5.4.3 Analysis

The measured latency distribution reveals two distinct processing regimes across the complementary datasets:

LIDC-IDRI Analysis. The LIDC results exhibit a pronounced bi-modal distribution driven by volume size. Of the 200 cases, 49 small-to-medium volumes ($<128^3$) fit entirely in GPU memory and execute via direct single-pass inference (mean 178.6 ms, end-to-end mean 1.46 s), while 151 large volumes require sliding-window patch inference with 60+ overlapping patch forward passes (mean 4,410.4 ms, end-to-end mean 6.68 s). File loading correlates strongly with inference time (Pearson $r=0.955$), confirming that volume size—not protocol overhead—drives latency variance. **Clinical implications:** For routine lung nodule screening (typically 200–300 slices), the P50 of 3.9 s is acceptable. For STAT (emergency) reads, large-volume studies benefit from higher-end GPUs (A100/H100 with 40–80 GB VRAM) to reduce patch count, TensorRT quantization for FP16 inference acceleration, or pre-cached model warm-up to eliminate first-call initialization.

MSD Heart Analysis. The cardiac MRI results demonstrate substantially higher latency (mean 18.2 s for 5-fold ensemble) due to three factors: (1) **5-fold ensemble inference:** five independently trained models execute sequential forward passes, with logits aggregated before postprocessing; (2) **nnU-Netv2 full pipeline:** automatic preprocessing includes resampling to target spacing, intensity normalization, and foreground/background extraction; automatic postprocessing includes connected component analysis and largest-component selection; and (3) **MRI complexity:** cardiac MRI exhibits higher anatomical variability and motion artifacts than thoracic CT, requiring more patch iterations. The P50 of 19.1 s and P99 of 27.6 s indicate that full nnU-Netv2 ensemble inference is suitable for background batch processing. Notably, single-fold inference achieves mean 6.50 s ($2.8\times$ faster than ensemble), providing a latency-accuracy trade-off for near-real-time workflows, while model initialization time (2.4–3.1 s) is a one-time cost per deployment session.

Reproducibility. LIDC measurements used random-initialized weights (forward-pass latency is weight-independent). Heart measurements used

fully trained 5-fold ensemble weights (mean validation Dice: 0.9324). Both benchmarks are publicly reproducible using the provided reference implementation code. All measurements include complete end-to-end overhead (file I/O, preprocessing, GPU inference, and result export), ensuring real-world applicability rather than idealized analytical estimates.

5.4.4 Comparison with Simulation Baseline

Prior simulation-based latency estimates (mean 2.0s, P99 7.8s) underestimated real-world overhead across both datasets. For LIDC-IDRI, the discrepancy is $2.7\times$ for mean latency (5.4s vs 2.0s) and $2.9\times$ for tail latency (22.6s vs 7.8s), arising from: (1) real file I/O adds 0.6-4.1s per study; (2) patch-based inference for large volumes adds 10-18s; and (3) HU normalization and data type conversion add 0.4-3.8s. For MSD Heart, the trained nnU-Netv2 5-fold ensemble exhibits mean 18.2s— $9.1\times$ slower than the simulation baseline—due to ensemble aggregation, automatic resampling, and full postprocessing pipeline. These results validate that analytical modeling significantly underestimates clinical deployment latency, and physical benchmarking with trained weights is essential for accurate workflow planning.

5.4.5 Protocol Overhead Decomposition

To isolate the latency contribution of the MCP-MedImg protocol stack itself from the underlying AI inference time, we designed a controlled experiment that incrementally activates protocol layers while holding the AI model constant.

Experimental Design. We processed $n = 98$ consecutive 3D LIDC-IDRI CT volumes through five experimental groups:

Control. All groups use the *same* 3D U-Net weights (22,575,362 parameters, in_channels=1, out_channels=2) and the *same* RTX 4090D GPU to ensure that differences in end-to-end latency arise solely from protocol layers, not from model architecture or hardware variance. Volumes were dynamically resized to 192^3 voxels (or smaller, depending on original dimensions) in fp16 precision to fit within 24 GB VRAM. Each volume was processed exactly once (no caching) to reflect cold-start clinical behavior. All measurements were performed on an AutoDL cloud instance (Intel Xeon, NVIDIA RTX 4090D, 24 GB VRAM, Ubuntu 22.04, CUDA 12.1, PyTorch 2.3.0).

Measurement Methodology. Each layer implements high-resolution timing probes ($1\mu\text{s}$ resolution) using Python’s `time.perf_counter_ns()`. Timing spans the exact protocol function call boundaries (e.g., entry to `SecurityModule.deidentify()` through return). GPU inference time is isolated with `torch.cuda.synchronize()` to prevent asynchronous kernel queuing from skewing measurements. Overhead was extracted by subtracting isolated G0 inference time from G4 total time, yielding per-volume pro-

protocol overhead. Validation: the sum of isolated G0 + per-layer probe measurements agreed with G4 total within 3.41% mean per-volume absolute percentage error (max 24.44% caused by a single outlier in the G0 isolated baseline measurement due to transient system load), confirming measurement consistency. Note that this per-volume metric is not directly derivable from the summary means in Tables 20–21, as mean of per-volume ratios differs from the ratio of summary means. Removing the top-1 outlier reduces the mean error to 3.19%, confirming that the protocol overhead decomposition is accurate and robust.

Table 20: Protocol Overhead Decomposition (LIDC-IDRI, $n=98$, RTX 4090D)

Grp.	Layers	Mean	P50	P99	Ovh. (ms/%)
G0	Baseline	76.78	38.06	278.06	0 / 0%
G1	L1 (Transport)	566.35	504.75	1,507.24	489.57 / 86.3%
G2	L1+L2 (Security)	566.71	505.09	1,507.93	489.93 / 86.3%
G3	L1+L2+L3 (Orchestration)	567.15	505.49	1,509.71	490.37 / 86.4%
G4	Full Stack	618.94	541.57	1,544.88	542.16 / 95.5%

Analysis. The protocol overhead decomposition reveals the marginal cost of each architectural layer. Table 21 presents the per-layer contribution extracted from the G4 full-stack measurements.

Table 21: Per-Layer Latency Contribution (G4 Full Stack, $n=98$)

Layer	Mean	Median	P99	% Prot.
L1 (Transport)	566.35	504.75	1,507.24	99.8%
L2 (Security)	0.36	0.34	0.69	0.1%
L3 (Orchestration)	0.44	0.40	1.78	0.1%
L4 (Semantics)	0.32	0.29	0.98	0.1%
Total Protocol	567.47	505.92	1,508.11	100%

Key Findings. Three principal findings emerge from the decomposition. First, **L1 Transport dominates protocol overhead**, contributing 99.8% (mean 566.35 ms) of the total protocol cost. The L1 latency splits into QIDO-RS metadata query (327.23 ms mean) and WADO-RS instance retrieval (239.12 ms mean), both scaling with DICOM series size. A strong Spearman correlation between L1 latency and WADO payload size confirms data-volume dependence ($\rho = 0.872$, $p < 10^{-30}$). Second, **Layers 2–4 are collectively sub-millisecond** (< 1.2 ms combined P99), representing the cost of pure in-memory Python operations: SHA256 hashing (L2), dictionary-based model registry lookups (L3), and state-machine transitions

(L4). These values constitute a *lower bound*; production deployments with distributed registries, encrypted audit pipelines, or networked verification workflows would incur additional latency. Nevertheless, even if L2–4 overhead increased $10\times$ ($\sim 12\text{ ms}$), it would remain negligible compared to L1 transport and GPU inference. Third, **measurement validation** confirms data integrity: all L2/L3/L4 per-volume measurements are strictly positive, L4 never exceeds 1 ms (max 0.98 ms), and the G4 total agrees with G0+protocol overhead within 3.41% mean per-volume error, satisfying internal consistency.

Clinical Implications. Understanding per-layer latency enables targeted optimization. Given that L1 Transport constitutes 99.8% of protocol overhead, **DICOMweb transport optimization** delivers the greatest latency reduction: enabling JPEG-LS or RLE transfer syntax negotiation at the PACS level can reduce WADO-RS payload by $40\text{--}60\%$, directly compressing the dominant cost. Parallelizing QIDO-RS metadata queries across multiple StudyInstanceUIDs (batch scheduling) amortizes per-query fixed overhead. For L2 Security, the current 0.36 ms cost is dominated by SQLite audit writes; replacing the local SQLite backend with an asynchronous write-behind queue (e.g., Kafka or Redis Streams) would add negligible latency while supporting multi-node deployments. For L3 Orchestration, the 0.44 ms registry lookup is a dictionary operation—production model registries (e.g., MLflow, NVIDIA Triton Model Repository) add network RTT ($\sim 1\text{--}5\text{ ms}$) but remain minor compared to L1. Pre-compiled DICOM SR templates eliminate the overhead of per-case XML generation. For L4 Semantics, the 0.32 ms state-machine traversal is effectively free; even complex multi-lesion workflows with iterative verification loops would add $< 5\text{ ms}$ per additional cycle. The decomposition thus provides actionable engineering guidance: optimize L1 first, keep L2–4 lightweight, and the protocol stack adds $< 600\text{ ms}$ to clinical workflows—acceptable for STAT reads ($< 3\text{ s}$ total) when combined with fast GPU inference.

5.5 Model Performance Validation

Beyond latency, the MSD Heart 5-fold cross-validation demonstrates strong segmentation accuracy. Table 22 presents per-fold validation Dice scores and the ensemble consensus.

Table 22: MSD Heart 5-Fold Cross-Validation Dice Scores

Fold	0	1	2	3	4	Mean
Validation Dice	0.9381	0.9323	0.9259	0.9410	0.9249	0.9324

The mean validation Dice of 0.9324 (± 0.0062) indicates robust gener-

alization across the 20-case dataset, consistent with published nnU-Netv2 benchmarks for cardiac MRI left atrium segmentation [49]. This validates that the trained model achieves clinically relevant accuracy while operating within the measured latency envelope.

6 Discussion

6.1 Protocol Design Decisions and Rationale

The four-layer architecture of MCP-MedImg reflects a deliberate response to the fragmentation, security gaps, and regulatory voids identified in our analysis of existing PACS-AI integration approaches. Each layer addresses a specific class of problems that, in combination, have prevented the scalable deployment of AI-assisted diagnosis in routine clinical practice.

6.1.1 Why Four Layers?

The decision to decompose the protocol into four layers rather than a monolithic design was driven by three considerations. First, **separation of concerns**: transport (L1), security (L2), orchestration (L3), and clinical semantics (L4) involve fundamentally different expertise domains—network engineering, information security, machine learning operations, and clinical workflow design, respectively. A layered architecture allows teams with different specializations to evolve their components independently. Second, **progressive adoption**: healthcare institutions vary enormously in their IT maturity. A community hospital with a legacy DICOM-only PACS can adopt L1 and L2 without requiring L3 or L4, while a tertiary academic medical center can deploy the full stack. Third, **upward compatibility**: the explicit design goal that dicom-mcp implementations serve as pluggable L1 components ensures that early adopters of MCP-based PACS access are not stranded by protocol evolution.

The protocol overhead decomposition (Section 5.4.5) was validated on CT chest imaging ($n = 98$ LIDC-IDRI volumes); the relative overhead proportion is expected to be consistent across MRI and other modalities, as Layers 2–4 are modality-agnostic and L1 transport scales with data size rather than imaging modality. The MSD Heart cardiac MRI benchmark (Section 5.4.1, ensemble mean 18.2s, single-fold 6.5s, Dice 0.9324) demonstrates that the protocol stack operates correctly with ensemble nnU-Netv2 inference, confirming cross-modality applicability even though the absolute latency is dominated by model-specific factors (5-fold ensemble, full preprocessing pipeline) rather than protocol overhead.

6.1.2 The Security-First Design

The placement of the Security and Compliance Layer (L2) between Transport (L1) and Orchestration (L3) is intentional and critical. In existing integration architectures, security is typically an afterthought—implemented as perimeter firewalls or VPN tunnels that protect the network boundary but do not address data-level threats. By contrast, MCP-MedImg’s L2 performs **data-centric security**: every DICOM instance is de-identified before it reaches the AI model, every inference request is authorized against RBAC policies, and every operation is logged with cryptographic hashes. This design directly addresses the MCP-38 threat taxonomy [24], which identified semantic attack surfaces, indirect prompt injection, and STDIO execution vulnerabilities as critical risks in MCP deployments.

The hierarchical PHI de-identification strategy (patient-level, department-level, system-level) was informed by the observation that different clinical contexts require different privacy-utility trade-offs. A research institution analyzing population-level trends may accept age-binning (department-level) to preserve demographic structure, while a clinical trial requiring individual patient linkage needs reversible hashing (patient-level with salt). The protocol’s configurability avoids the one-size-fits-all approach that has hampered adoption of previous de-identification standards.

6.1.3 Conditional Verification as a Governance Primitive

The conditional verification mechanism (MANDATORY/ADVISORY/AUTO) represents more than a workflow convenience—it is a **governance primitive** that embeds regulatory intent into the protocol itself. The EU AI Act’s requirement for human oversight in high-risk AI systems [9] is typically interpreted as a mandate for universal human review, which would overwhelm radiologists and create bottlenecks. MCP-MedImg’s graduated approach—mandatory for new models and high-stakes screening, advisory for validated routine diagnostics, automated for low-risk follow-up—demonstrates that regulatory compliance and operational efficiency are not mutually exclusive.

The verification state machine (PENDING → UNDER_REVIEW → REVIEWED → CONFIRMED/MODIFIED → FINALIZED) with timeout escalation ensures that no case is lost in the verification pipeline. This is particularly important for STAT (urgent) cases, where the protocol guarantees AUTO verification regardless of model availability—a safety property that prevents emergency care from being blocked by AI system failures.

6.2 Clinical Impact Assessment

6.2.1 Workflow Integration

The reference implementation’s end-to-end workflow (PACS → Gateway → AI → Verification → PACS) was designed to minimize disruption to existing radiologist workflows. AI results are written back to PACS as DICOM Structured Reports (SR), which appear natively in standard PACS viewers without requiring custom plugins or separate dashboards. This is a significant departure from current SaMD products, which often force radiologists to switch between multiple applications.

Preliminary workflow simulations (pending full clinical validation) suggest the following time impacts:

Table 23: Estimated Workflow Time Impact by Verification Mode

Phase	MANDATORY	ADVISORY	AUTO
AI inference	1–3 s	1–3 s	1–3 s
Radiologist review	2–5 min	0–2 min	0 s
Report generation	3–8 min	1–4 min	0 s
Total time-to-report	5–13 min	2–7 min	1–3 s

For AUTO-mode workflows (e.g., bone age assessment, low-risk follow-up), the time savings are dramatic—from minutes to seconds. For MANDATORY-mode workflows (e.g., lung cancer screening), the AI serves as a **preliminary reader** that flags regions of interest, potentially reducing the time radiologists spend searching for abnormalities while preserving their authority as the final arbiter.

6.2.2 Trust and Acceptance

A critical barrier to clinical AI adoption is **radiologist trust** [46]. The conditional verification mechanism directly addresses this by making the AI’s role transparent and bounded: the radiologist always knows whether they are reviewing an AI-assisted case (MANDATORY), may choose to review (ADVISORY), or can accept automation (AUTO). The complete audit trail—including the AI’s confidence scores, the radiologist’s modifications, and the reasons for disagreement—creates an evidence base for trust that accumulates over time.

Furthermore, the verification state machine’s history tracking enables **continuous quality improvement**. When a radiologist consistently overrides the AI on a particular type of case, this pattern is detectable in the audit logs and can trigger model retraining or policy adjustment. This

closed-loop feedback is impossible with standalone SaMD products that operate in isolation.

6.3 Regulatory and Compliance Implications

6.3.1 Total Product Life Cycle (TPLC) Surveillance

The convergence of global regulators on TPLC surveillance—FDA’s 2025 draft guidance [1], NMPA’s June 2025 measures [11], and the EU AI Act [9]—creates both an opportunity and a challenge for MCP-MedImg. The opportunity is that the protocol’s built-in audit infrastructure (14-column AuditLogger with regulatory tags) provides a standardized mechanism for the continuous post-market monitoring that regulators now demand. The challenge is that regulatory frameworks for **service-oriented AI delivery** (as opposed to standalone SaMD products) are still evolving.

MCP-MedImg’s audit logs include regulatory tags (FDA_TPLC, NMPA_Grade2, EU_AI_Act) that can be automatically compiled into surveillance reports. For example, a quality officer seeking to monitor lung nodule detection performance across all chest CT studies can query the audit database for:

```
1 SELECT
2     ai_model,
3     COUNT(*) as total_cases,
4     AVG(CASE WHEN verification_action = 'MODIFY' THEN 1 ELSE 0
5     END) as override_rate,
6     AVG(inference_time_ms) as avg_latency
7 FROM audit_log
8 WHERE regulatory_tags LIKE '%FDA_TPLC%'
9     AND operation = 'medical_image_analyze_complete'
10    AND timestamp >= DATE('now', '-90 days')
11 GROUP BY ai_model
12 ORDER BY override_rate DESC;
```

Listing 36: Example TPLC Surveillance Query

This query would identify models with high override rates—a potential indicator of performance drift—enabling proactive intervention before patient harm occurs.

6.3.2 Risk Classification and Conditional Verification

The EU AI Act classifies AI-based medical software as high-risk AI systems, requiring continuous monitoring, record-keeping, human oversight, and transparency [9]. MCP-MedImg’s conditional verification mechanism provides a technical implementation of these requirements:

- **Human oversight:** MANDATORY mode ensures human review for high-risk cases.

- **Record-keeping:** The 14-column audit trail captures all AI-human interactions.
- **Transparency:** Model metadata (performance metrics, regulatory approvals, version history) is exposed via the ModelRegistry resource.
- **Continuous monitoring:** Audit logs enable real-time performance tracking.

The interplay between the EU MDR/IVDR and the AI Act creates a layered regulatory architecture where AI components inherit the risk classification of the medical device they are embedded in. MCP-MedImg’s conformance model (Level 1/2/3) provides a technical mapping to these regulatory tiers: Level 1 for basic compliance, Level 2 for clinical pilots with institutional policy, Level 3 for full production with regulatory surveillance.

6.3.3 Data Sovereignty and Cross-Border Deployment

For international deployments, MCP-MedImg’s hierarchical PHI de-identification supports varying data sovereignty requirements. The patient-level hashing with per-deployment salt ensures that patient identities cannot be reconstructed even if audit logs are aggregated across institutions. This is essential for multi-site clinical trials and federated learning scenarios where data must remain at each site while model performance is monitored centrally.

6.4 Comparison with Existing Solutions

Table 24 presents a comprehensive comparison of MCP-MedImg against the six primary existing integration architectures identified in Section 2.

The key differentiator is MCP-MedImg’s **protocol-layer standardization**. While existing solutions require custom integration for each AI vendor, MCP-MedImg enables a single protocol implementation to serve all vendors. This is analogous to the difference between the pre-HTTP internet (where each service required a custom client) and the modern web (where any browser can access any website).

6.5 The dicom-mcp Ecosystem as a Foundation

Existing dicom-mcp implementations, particularly Christian Hinge’s reference server [20] and Berk Durmus’s variant [21], provide valuable building blocks for MCP-MedImg’s Transport Layer. These implementations demonstrate that MCP can successfully bridge to DICOM infrastructure, but they are functionally limited to data access and metadata extraction.

Table 24: Comprehensive Comparison: MCP-MedImg vs. Existing Integration Architectures

Dimension	Direct Plugin	DICOM Router	AI Orchestration	MCP-MedImg
Integration complexity	Very High	High	Medium	Low
Vendor lock-in	Complete	Partial	Significant	None
Security model	Fragmented	Ad hoc	Proprietary	Hierarchical, unified
PHI de-identification	Vendor-specific	None	Partial	Policy-driven, multi-tier
AI model switching	Manual	Manual	Manual	Auto-routing by inspection type
Result standardization	Proprietary	Proprietary	Proprietary	DICOM SR/SEG/FHIR
Audit trail	Limited	None	Limited	Complete lifecycle audit
Human verification	Optional	None	Optional	Conditional (M/A/A)
Service delivery model	Product	Product	Product	On-demand protocol service
Regulatory TPLC support	Ad hoc	None	Ad hoc	Built-in surveillance
Public infrastructure	Closed	N/A	Closed	Governable
Standard governance	Vendor-controlled	N/A	Vendor-controlled	Linux Foundation

MCP-MedImg’s relationship to dicom-mcp is analogous to HTTP’s relationship to TCP: dicom-mcp provides the reliable transport, while MCP-MedImg provides the application-layer semantics (security, orchestration, clinical workflows) that make the transport useful for medical AI. Organizations already using dicom-mcp can adopt MCP-MedImg incrementally by adding L2–L4 without replacing their L1 transport, preserving their existing investment.

6.6 Security Analysis in Depth

6.6.1 MCP-38 Threat Mitigation

The MCP-38 threat taxonomy [24] identified 38 categories of security risks in MCP systems. Table 25 details how MCP-MedImg addresses the most critical threats for medical imaging deployments.

Table 25: MCP-38 Threat Mitigation in MCP-MedImg

Threat gory	Cate-	Specific Risk	MCP-MedImg Miti- gation
Semantic surface	attack	Tool description tam- pering influences LLM behavior	Description-field sig- nature verification + source whitelist (L2)
Indirect injection	prompt	DICOM data contains malicious instructions	Input sanitization + structured parsing (non-free-text) (L2)
STDIO execution		Arbitrary command ex- ecution via MCP SDK	Containerized sandbox + command whitelist + least privilege (L2)
Supply-chain risk		Malicious model upload to registry	Model signature verifi- cation + Registry re- view process (L3)
Data exfiltration		Unauthorized PHI ac- cess	Hierarchical de- identification + RBAC + audit (L2)

The containerized sandbox approach (Docker with seccomp profiles) ensures that even if an attacker compromises the MCP server, they cannot escape to the host system or access unauthorized PHI. The command whitelist rejects any MCP tool invocation that attempts to execute system commands outside the defined set (query, retrieve, analyze, verify, store).

6.6.2 Cryptographic Audit Integrity

The audit trail’s cryptographic integrity mechanism—SHA-256 hashes of input data, AI output, and human modifications—provides tamper-evidence that satisfies HIPAA §164.312(c)(1) and ISO/IEC 27037 [44] requirements. The hash chain ensures that any modification to audit records after the fact would break the cryptographic linkage, making tampering detectable.

The 16-character truncation of SHA-256 hashes for PatientID and audit log fields represents a deliberate trade-off: at 64 bits, the collision probability in a target population of up to 2^{32} patients is negligible ($\approx 2^{-32}$ by the birthday bound), while storage and index efficiency are significantly improved. The full 64-character hash remains available internally for verification purposes.

6.7 Limitations and Mitigations

We acknowledge several limitations of the current work, along with mitigation strategies:

Limited Modality Validation: The reference implementation has been validated for CT (chest) and MG (mammography) only. Extension to MR, XR, US, and ophthalmic imaging (OCT, fundus photography) requires additional InspectionRouter rules and model adapters. However, the protocol’s modular design means these extensions can be added as L4 plugins without modifying L1–L3.

GPU Inference Scope: Real GPU inference has been validated on two datasets (LIDC-IDRI with 200 CT volumes and MSD Heart with 20 MRI volumes) using an NVIDIA RTX 4090 D. The LIDC benchmark used a randomly initialized 3D U-Net (forward-pass latency is weight-independent)⁴, while the Heart benchmark used fully trained nnU-Netv2 5-fold ensemble weights (mean Dice 0.9324). However, the Heart dataset is limited to 20 cases and a single anatomical structure (left atrium). Broader validation across diverse anatomies, pathologies, and model architectures (e.g., commercial AI APIs, custom models) remains future work. The protocol itself is agnostic to the inference backend—any model can be integrated via the ModelRegistry.

Scalability Testing: Large-scale concurrency testing (100+ studies/hour) has not been conducted. The stateless deployment design (MCP 2026 specification) supports horizontal scaling, but load balancing, queue management, and database contention under high load require further validation. We recommend Kubernetes-based auto-scaling for production deployments.

⁴Model weights are randomly initialized for the LIDC latency benchmark; this is valid because GPU compute latency during forward-pass is independent of weight values.

Regulatory Approval Pathway: The service-oriented delivery model (AI as a protocol service rather than a standalone product) does not yet have an established regulatory approval pathway. FDA, NMPA, and EU regulators are still developing guidance for continuous-learning AI systems. MCP-MedImg’s audit infrastructure is designed to support whatever surveillance requirements emerge, but the specific regulatory classification of protocol-governed AI services remains an open question.

Simplified De-identification: The reference implementation’s de-identification covers PatientID, PatientName, and StudyDate. Full three-level hierarchical de-identification (including age-binning, relative date formatting, and device information handling) is specified in the protocol but not yet fully implemented. This is a priority for production deployment.

Governance Mechanisms: While MCP-MedImg provides the technical preconditions for participatory governance (standardized audit trails, performance metadata, protocol compliance checks), the social and policy mechanisms for democratic oversight, equitable allocation, and collective accountability require further research. We draw on Ostrom’s commons governance principles [22] and Benkler’s peer production theory [23] as conceptual foundations, but operational governance frameworks remain future work.

6.8 Future Work and Research Directions

6.8.1 Extension to Additional Modalities

The most immediate priority is extending MCP-MedImg to additional imaging modalities:

- **Ophthalmology:** OCT (retinal layer segmentation, glaucoma detection, diabetic retinopathy screening) and fundus photography. The high prevalence of ophthalmic AI models (e.g., IDx-DR, EyeArt) makes this a high-impact extension.
- **Pathology:** Whole-slide imaging (WSI) with digital pathology AI for cancer grading and biomarker detection. This requires adaptation of the Transport Layer to support DICOM WSI and TIFF formats.
- **Endoscopy:** GI tract lesion detection and classification, requiring video stream handling in the Transport Layer.
- **Cardiology:** Echocardiography and cardiac MRI, with specific requirements for temporal (video) analysis.

Each modality extension is implemented as an L4 plugin (Inspection-Router rules + LesionEncoder grading systems + model adapters), preserving the protocol’s core architecture.

6.8.2 Federated Learning Integration

MCP-MedImg’s audit infrastructure and hierarchical de-identification provide a foundation for federated learning scenarios where model training occurs across multiple institutions without centralizing patient data. Future work will explore:

- **Federated model updates:** Using MCP-MedImg’s audit logs as the coordination mechanism for federated averaging across sites.
- **Differential privacy:** Adding noise to model gradients at the protocol layer to prevent membership inference attacks.
- **Secure aggregation:** Using homomorphic encryption or secure multi-party computation for model update aggregation.

6.8.3 Deep FHIR Integration

While the current implementation supports FHIR Observation output, deeper integration with HL7 FHIR would enable:

- **Bidirectional data flow:** AI results written to FHIR that trigger clinical decision support rules in the EHR.
- **Patient context:** Access to FHIR resources (Patient, Condition, MedicationRequest) to enrich AI inference with clinical history.
- **Care pathway integration:** AI findings that automatically generate follow-up orders, referrals, or patient notifications via FHIR workflows.

6.8.4 Regulatory Framework Development

As regulators develop guidance for service-oriented AI delivery, MCP-MedImg will evolve to support:

- **Predetermined Change Control Plans (PCCP):** Protocol-level support for FDA’s PCCP framework, where model updates within predefined bounds are automatically approved.
- **Real-world evidence collection:** Automated compilation of audit logs into regulatory submission packages.
- **Post-market surveillance dashboards:** Real-time monitoring of model performance across the installed base with automated alerts for performance drift.

6.8.5 Cloud-Native Stateless Deployment

The MCP 2026 stateless specification [38] enables horizontal scaling without server-side session state. Future work will optimize MCP-MedImg for cloud-native deployment:

- **Kubernetes operators:** Automated deployment, scaling, and management of MCP-MedImg gateways.
- **Serverless inference:** Integration with serverless GPU platforms (e.g., AWS Lambda with GPU support) for cost-effective scaling.
- **Multi-region deployment:** Geo-distributed gateways with data residency compliance.

6.8.6 Public Infrastructure Governance Research

The transformation from proprietary AI products to protocol-governed services creates the preconditions for participatory governance, but the specific mechanisms require interdisciplinary research:

- **Democratic oversight:** How should model performance standards be set? Who participates in protocol governance? How are conflicts of interest managed?
- **Equitable allocation:** How should AI capacity be distributed across regions with different resource levels? How do we prevent “AI deserts” where underserved populations lack access?
- **Collective accountability:** How do we attribute responsibility when an AI error occurs in a protocol-governed system? What liability frameworks apply?

We propose that the Linux Foundation’s Agentic AI Foundation, as the governance body for MCP, establish a Medical Imaging Working Group to address these questions, drawing on expertise from clinical medicine, health policy, law, and computer science.

7 Conclusion

7.1 Summary of Contributions

This paper presented MCP-MedImg, a dedicated extension of the Model Context Protocol for medical image AI integration. Our contributions can be summarized as follows:

1. **A four-layer protocol architecture** that standardizes PACS-AI integration across Transport, Security, Orchestration, and Application Semantics layers, with explicit upward compatibility with existing dicom-mcp implementations.
2. **A hierarchical security model** addressing the MCP-38 threat taxonomy through data-centric PHI de-identification, cryptographic audit trails, RBAC policy enforcement, and containerized sandbox execution.
3. **Inspection-type-driven AI model routing** that automatically dispatches optimal models based on DICOM metadata (Modality, Body-Part, clinical indication), eliminating manual model selection and reducing integration friction.
4. **Standardized result generation** in DICOM SR, SEG, and FHIR formats, ensuring that AI outputs are immediately usable in existing clinical workflows without custom viewers or proprietary dashboards.
5. **Conditional human-in-the-loop verification** with three modes (MANDATORY/ADVISORY/AUTO) that balance clinical safety, regulatory compliance, and operational efficiency, directly supporting the Total Product Life Cycle surveillance requirements emerging from global regulators.
6. **A reference implementation and validation framework** based on open-source Orthanc PACS and nnU-Net, demonstrating protocol feasibility, end-to-end workflow integration, multi-model routing, standardized output generation, and complete audit trail capture.

7.2 A Proposed Architectural Transition: From SaMD to Service

The central argument of this paper is that AI-assisted diagnosis must transition from standalone Software as a Medical Device (SaMD) products to **auditable, service-oriented capabilities**. This transition is not merely a technical architecture change—it is an architectural reconceptualization of how AI is delivered, governed, and trusted in healthcare.

For healthcare institutions, the service-oriented model means reduced procurement complexity, lower total cost of ownership, and simplified compliance through unified audit trails. New AI models become available immediately via protocol update rather than requiring months of integration work.

For clinicians, it means enhanced trust through complete transparency—every AI decision is auditable, every override is recorded, and every disagreement is analyzable. Professional autonomy is preserved through configurable

verification modes that respect the clinician’s judgment while providing AI assistance.

For AI vendors, it means reduced integration burden—a single protocol implementation serves all customers—and continuous improvement enabled by gray release, A/B testing, and real-world performance monitoring.

For regulators, it means standardized surveillance with protocol-layer audit trails enabling consistent post-market monitoring across vendors, real-world evidence through continuous data collection, and risk mitigation through conditional verification for high-risk applications.

For society, the service-oriented model enables AI-assisted diagnosis as **governable public infrastructure**—transparent, equitably accessible, and collectively accountable. By transforming proprietary products into protocol-governed services, MCP-MedImg provides the technical preconditions for participatory governance in healthcare AI.

7.3 Validation Status and Next Steps

The reference implementation has been validated through 10 verification scenarios covering protocol consistency, end-to-end workflow, multi-model routing, result standardization, service auditability, conditional verification, and regulatory TPLC support. Real GPU inference benchmarks on LIDC-IDRI (200 CT volumes, mean 5.4s⁵) and MSD Heart (10 test volumes from 20-case dataset, ensemble mean 18.2s, single-fold 6.5s, Dice 0.9324) confirm protocol feasibility under realistic clinical conditions. While large-scale concurrency testing and multi-anatomy validation remain future work, the current results provide a solid foundation for production deployment.

The next steps are:

1. Extend GPU benchmarking to additional anatomies and model architectures (e.g., commercial AI APIs, custom models beyond nnU-Net).
2. Conduct large-scale concurrency testing (100+ studies/hour) with Kubernetes-based auto-scaling.
3. Extend validation to additional modalities (MR, XR, US, OCT).
4. Conduct clinical pilot studies with partner institutions to assess radiologist workflow integration and acceptance.
5. Engage with regulatory bodies (FDA, NMPA, EU MDCG) to establish approval pathways for service-oriented AI delivery.
6. Establish the Medical Imaging Working Group under the Linux Foundation’s Agentic AI Foundation to govern protocol evolution.

⁵LIDC benchmark used randomly initialized weights; forward-pass latency is weight-independent.

7.4 Final Remarks

As regulators worldwide demand more rigorous post-market surveillance, and as society seeks accountable AI in healthcare, MCP-MedImg provides the technical infrastructure to make such surveillance and governance practical, standardized, and scalable. The upward compatibility with existing dicom-mcp implementations ensures a smooth migration path for early adopters, while the protocol’s alignment with the MCP 2026 stateless specification and Linux Foundation governance ensures long-term enterprise viability.

MCP-MedImg stands ready to bridge the gap between AI innovation and clinical trust—enabling the transformation of medical imaging AI from isolated products into governed, accessible, and accountable services that improve patient outcomes while respecting clinician autonomy and regulatory requirements.

Acknowledgments

The author (Xiong Shi) thanks the open-source communities of Orthanc, nnU-Net, pydicom, highdicom, and the Model Context Protocol for their foundational work. AI-assisted tools were used for literature synthesis, software implementation, and drafting support. All technical contributions, experimental designs, and conclusions are the author’s original work.

Data Availability

The complete reference implementation, experimental data, and protocol specifications are publicly available: **GitHub:** github.com/Xiong-Shi/MCP-MedImg (source code, tests, benchmarks, and documentation); **Zenodo:** [doi:10.5281/zenodo.21941869](https://doi.org/10.5281/zenodo.21941869) (v2.0.0 dataset archive containing model weights, training logs, inference results, JSON manifests, and analysis notebooks). All materials are released under open licenses to facilitate reproducibility and community adoption: dataset and experimental data under CC-BY 4.0; source code under Apache-2.0.

Ethics Approval and Consent to Participate

This study utilized publicly available, de-identified medical imaging datasets (LIDC-IDRI and MSD Heart). As the research involved no human subjects recruitment and used only anonymized public data, institutional ethics committee review was not required. The study was conducted in accordance with the data use policies of each dataset.

Competing Interests

The author declares no competing interests.

Funding

This research received no external funding.

Authors' Contributions

Xiong Shi conceived the study, designed the protocol, implemented the reference prototype, conducted all experiments, and drafted the manuscript.

Consent for Publication

Not applicable.

References

- [1] FDA. “Artificial Intelligence-Enabled Device Software Functions: Life-cycle Management and Marketing Submission Recommendations.” Draft Guidance, January 7, 2025. Docket No. FDA-2024-D-4488.
- [2] NMPA. “2025 Annual Report on Medical Device Registration.” National Medical Products Administration, 2026 (reporting 2025 data).
- [3] China Medical Device Industry Report. “AI Medical Device Market Analysis.” China Medical Device Information, 2025.
- [4] McKinney, S.M., et al. “International evaluation of an AI system for breast cancer screening.” *Nature*, 2020, 577(7788): 89–94.
- [5] Rajpurkar, P., et al. “Deep learning for chest radiograph diagnosis: A retrospective comparison of the CheXNeXt algorithm to practicing radiologists.” *PLOS Medicine*, 2018, 15(11): e1002686.
- [6] Nirmittee. “DICOM + FHIR Medical Imaging Interoperability.” Nirmittee Blog, April 1, 2026.
- [7] IHE. “AI Results (AIR) Supplement.” IHE Radiology Technical Framework, Trial Implementation, 2025. https://www.ihe.net/uploadedFiles/Documents/Radiology/IHE_RAD_Suppl_AIR.pdf
- [8] FDA. “Request for Public Comment: Measuring and Evaluating AI-enabled Medical Device Performance in the Real-World.” September 30, 2025. Docket No. FDA-2025-N-4203.

- [9] European Union. Regulation (EU) 2024/1689 (AI Act). Entered into force August 1, 2024; obligations phased in through 2028 (high-risk AI systems from August 2, 2026).
- [10] MDCG and European Artificial Intelligence Board. “MDCG 2025-6 / AIB 2025-1: Interplay between the Medical Devices Regulation (MDR) and In Vitro Diagnostic Medical Devices Regulation (IVDR) and the Artificial Intelligence Act (AIA).” June 2025.
- [11] NMPA. “Announcement on Measures to Optimize Whole-Lifecycle Regulation to Support the Innovative Development of High-End Medical Devices.” No. 63 of 2025, June 27, 2025.
- [12] NMPA. “Registration Review Guideline for Artificial Intelligence Medical Devices.” 2022.
- [13] Pesapane, F., De Cecco, C.N., Wang, H., Hauglid, M.K., Sardanelli, F. “Artificial intelligence as medical device in radiology in 2025: the regulatory scenario in the EU, USA, and China.” *European Radiology*, 2026, 36: 6246–6257. DOI: 10.1007/s00330-026-12460-4.
- [14] Anthropic. “Introducing the Model Context Protocol.” November 2024.
- [15] Linux Foundation. “Linux Foundation Announces the Formation of the Agentic AI Foundation (AAIF).” Press Release, December 9, 2025.
- [16] Anthropic. “MCP Ecosystem Update: 10,000+ Active Public Servers and 97M+ Monthly SDK Downloads.” December 2025.
- [17] Ehtesham, A., Singh, A., Gupta, G.K., Kumar, S. “A Survey of Agent Interoperability Protocols: Model Context Protocol (MCP), Agent Communication Protocol (ACP), Agent-to-Agent Protocol (A2A), and Agent Network Protocol (ANP).” arXiv:2505.02279, 2025.
- [18] “Transforming clinical medicine with multimodal artificial intelligence, agentic systems, and the model-context protocol: a perspective on future directions.” *Journal of Medical Systems*, 2026, 51(1). DOI: 10.1007/s44250-026-00343-w.
- [19] Lareau, D. (Medicomp Systems). “Healthcare Interoperability – 2026 Health IT Predictions.” *Healthcare IT Today*, January 8, 2026.
- [20] Hinge, C. “dicom-mcp v0.1.2.” GitHub, April 28, 2025. <https://github.com/ChristianHinge/dicom-mcp>
- [21] Durmus, B. “mcp-dicom-server.” GitHub, 2025. <https://github.com/berkdurmus/mcp-dicom-server>

- [22] Ostrom, E. *Governing the Commons: The Evolution of Institutions for Collective Action*. Cambridge University Press, 1990.
- [23] Benkler, Y. *The Wealth of Networks: How Social Production Transforms Markets and Freedom*. Yale University Press, 2006.
- [24] Shen, Y.T., Toyoda, K., Leung, A. “MCP-38: A Comprehensive Threat Taxonomy for Model Context Protocol Systems.” arXiv:2603.18063, 2026.
- [25] Anbiaee, Z., et al. “Security Threat Modeling for Emerging AI-Agent Protocols: A Comparative Analysis of MCP, A2A, Agora, and ANP.” arXiv:2602.11327, 2026.
- [26] OX Security. “The Mother of All AI Supply Chains: Critical, Systemic Vulnerability at the Core of Anthropic’s MCP.” OX Security Blog, April 2026.
- [27] Model Context Protocol Specification. “Security Considerations.” <https://modelcontextprotocol.io/specification/2026-07-28/basic/authorization/security-considerations>. Accessed 2026.
- [28] DICOM Standard. PS3.18: Web Services (DICOMweb).
- [29] DICOM Standard. PS3.3: Information Object Definitions.
- [30] “The Dawn of PACS 3.0: The Evolution from Image Access to Workflow Intelligence.” *Journal of Imaging Informatics in Medicine*, 2025.
- [31] SIIM Annual Meeting. “Making AI Work in the Imaging Stack: Engineering Lessons from Real Deployments.” InformaticsTECH Talk, 2025.
- [32] Perez-Sanpablo, A.I., et al. “Transforming Medical Imaging: The Role of Artificial Intelligence Integration in PACS for Enhanced Diagnostic Accuracy and Workflow Efficiency.” *Current Medical Imaging*, 2025.
- [33] Hou, X., Zhao, Y., Wang, S., Wang, H. “MCP: Landscape, Security Threats, and Future Research Directions.” arXiv:2503.23278, 2025.
- [34] OpenAPI Initiative. “OpenAPI Specification 3.0.3.” 2020.
- [35] KServe. “Open Inference Protocol.” 2023.
- [36] OASIS. “MQTT Version 5.0.” OASIS Standard, 2019.
- [37] Kubernetes. “Container Runtime Interface (CRI) Specification.” 2021.
- [38] MCP Specification. “Stateless Design Update.” July 28, 2026.
- [39] highdicom Documentation. <https://highdicom.readthedocs.io/>

- [40] IEEE. “IEEE 829-2008 Standard for Software Test Documentation.” 2008.
- [41] ISO/IEC/IEEE. “ISO/IEC/IEEE 29119-1:2022 Software and systems engineering — Software testing.” 2022.
- [42] HL7 International. “HL7 FHIR R5 — Fast Healthcare Interoperability Resources.” 2023. <https://hl7.org/fhir/>.
- [43] W3C. “Web of Things (WoT) Thing Description.” W3C Recommendation, 2020. <https://www.w3.org/TR/wot-thing-description/>.
- [44] ISO/IEC. “ISO/IEC 27037:2012 — Guidelines for identification, collection, acquisition and preservation of digital evidence.” 2012.
- [45] U.S. Department of Health and Human Services. “HIPAA Security Rule, 45 CFR §164.312 — Technical safeguards.” 2003.
- [46] Oakden-Rayner, L., et al. “Hidden stratification causes underperformance in datasets across medical imaging deep learning models.” *Radiology: AI*, 2020, 2(4): e190086.
- [47] Armato III, S.G., et al. “The Lung Image Database Consortium (LIDC) and Image Database Resource Initiative (IDRI): a completed reference database of lung nodules on CT scans.” *Medical Physics*, 2011, 38(2): 915–931.
- [48] Simpson, A.L., et al. “A large annotated medical image dataset for the development and evaluation of segmentation algorithms.” *arXiv preprint arXiv:1902.09063*, 2019.
- [49] Isensee, F., et al. “nnU-Net: a self-configuring method for deep learning-based biomedical image segmentation.” *Nature Methods*, 2021, 18(2): 203–211.