Scientific Research Publishing

Advances in Artificial Intelligence and Robotics Research, 2026, 2(3), 190-208
https://www.scirp.org/journal/airr
ISSN Online: 3143-3995
ISSN Print: 3143-3987

CUTEADMOA: A Unified Mixture-of-Agents Platform with Continuous Self-Training for Autonomous Enterprise AI Automation

Md Abu Sayeed

Afritech54 LLC, Dhaka, Bangladesh
Email: sayeed.dotprogrammers@gmail.com

How to cite this paper: Sayeed, M.A. (2026) CUTEADMOA: A Unified Mixture-of-Agents Platform with Continuous Self-Training for Autonomous Enterprise AI Automation. *Advances in Artificial Intelligence and Robotics Research*, **2**, 190-208.
https://doi.org/10.4236/airr.2026.23011

Received: August 14, 2026
Accepted: September 21, 2026
Published: September 24, 2026

Abstract

The rapid proliferation of large language models has produced powerful but fragmented automation tooling: mixture-of-agents frameworks that improve output quality, routing systems that reduce inference cost, orchestration frameworks that coordinate teams of agents, and lifelong-learning methods that enable models to improve over time. Each advance is typically delivered as an isolated component with no integrated path from interaction capture to model improvement. This paper presents CUTEADMOA, a unified production platform that integrates heterogeneous model aggregation, intent-based smart routing, multi-tier reliability engineering, enterprise security scanning, AI-powered web automation, and multimedia generation within a single zero-dependency architecture. It is accompanied by CUTTYMOA-1.0, a continuously self-trained companion model driven by a closed-loop pipeline that harvests production interactions, curates them into 204 verified datasets spanning 24 categories, fine-tunes a mixture-of-experts base model, updated model into the serving path, with controlled evaluation of retrained-model improvement planned as future work (Sections 6.3 and 7). The system captures every conversation, application programming interface call, and security scan into a streaming knowledge corpus exceeding 857,000 training pairs with a large keyword index for retrieval-augmented generation. We describe the architectural design, the continuous learning pipeline, and the operational engineering, including redundant gateway proxies with independent credential-pool rotation, preemptive rate-limit handling, and a failover chain of more than twenty fallback slots, that distinguishes the platform from research prototypes. Operational evaluation shows deterministic latency tiers, sustained availability under credential exhaustion, and monotonic knowledge growth. We discuss limitations and outline future work on formal benchmarking and multi-tenant deployment. The principal contribution is a reference architecture for autonomous,

self-improving AI platforms that closes the loop between usage and capability.

Keywords

Mixture-of-Agents, Multi-Agent Orchestration, Autonomous AI, Continuous Learning, Model Routing, Enterprise Automation, Retrieval-Augmented Generation, Self-Improving Systems

1. Introduction

1.1. Background and Motivation

Between 2024 and 2026, large language model based systems moved decisively from interactive chatbots toward autonomous execution. Agentic frameworks now plan, call tools, coordinate sub-tasks, and operate across software environments, while enterprise deployments increasingly treat models as a shared, routable compute resource rather than a single monolithic assistant. Three research lines have driven this shift. First, mixture-of-agents (MoA) architectures demonstrated that layering multiple models as proposers and aggregators improves response quality beyond any single constituent model [1], with later refinements improving diversity [2] and alignment [3]. Second, model routing research showed that selecting the cheapest or fastest model sufficient for a given query can cut inference cost dramatically without sacrificing quality [4] [5]. Third, multi-agent orchestration research contributed frameworks for persistent and ephemeral agents, hierarchical coordination, and evolving shared knowledge [6]-[8].

In parallel, the study of lifelong and continuous learning established that autonomous systems require mechanisms to accumulate experience, update memory, and retrain incrementally rather than remaining frozen at deployment time [9] [10]. Security research further documented that agentic systems introduce new failure modes, from prompt manipulation to unreliable multi-step execution, and argued for governance, validation, and reliability engineering as first-class concerns [11] [12]. Despite this progress, the literature overwhelmingly treats these capabilities as separate research artifacts. Published systems are evaluated in controlled benchmarks; credential exhaustion, provider outages, rate limits, and the messy realities of production deployment are largely absent from academic treatment.

1.2. Research Gap and Contributions

The gap addressed in this work is the absence of a published reference architecture for a production AI platform that simultaneously 1) aggregates heterogeneous models, 2) routes queries by intent and speed tier, 3) engineers reliability through redundant gateways and credential rotation, 4) provides built-in security scanning and reporting, 5) automates web data acquisition, and 6) continuously retrains its own companion model from captured production interactions. Independent researchers and small teams building such platforms currently lack a documented

blueprint; conversely, the research community lacks empirical evidence about how such integrated systems behave operationally. This paper contributes such a blueprint and evidence, drawn from a working system.

The specific contributions are as follows. First, we present the layered architecture of CUTEADMOA, a unified platform combining smart intent routing over 24 intents, multi-model aggregation over a pool of 23 or more heterogeneous engines, a seventeen-intent security engine, an automation and web-scraping engine, and a multimedia engine, all behind a backward-compatible application programming interface (API) that has remained stable across six major platform versions. Second, we detail the reliability engineering layer, including redundant gateway proxies with independent credential pools, preemptive rotation at 95% of documented rate limits, and a failover chain with more than twenty fallback slots, and we report its observed operational characteristics. Third, we describe CUTTYMOA-1.0, a continuously self-trained companion model, and the closed-loop pipeline that harvests production interactions, curates 204 verified datasets across 24 categories, fine-tunes a mixture-of-experts base model, and redeploys it. Fourth, we evaluate the system along operational dimensions, compare it with representative related systems, and discuss limitations and future work.

1.3. Paper Organization

The remainder of this paper is organized as follows. Section 2 reviews related work in mixture-of-agents, model routing, multi-agent orchestration, lifelong learning, and agentic-AI security and reliability. Section 3 presents the CUTEADMOA system architecture and its principal components. Section 4 describes the continuous self-training pipeline underlying CUTTYMOA-1.0. Section 5 reports the evaluation approach and results. Section 6 discusses the platform's distinguishing characteristics, engineering lessons, and limitations. Section 7 concludes and outlines future work.

2. Related Work

2.1. Mixture-of-Agents Architectures

Wang *et al.* introduced the mixture-of-agents framework, in which multiple models first generate proposals for a query and a subsequent aggregator layer synthesizes a final response [1]. Their results showed that MoA can outperform substantially larger individual models on knowledge and reasoning benchmarks, establishing layered model collaboration as a practical technique. Subsequent work improved diversity-aware routing within MoA layers [2], applied differentiable weighting to incentivize effective swarm collaboration [3], and used collective intelligence for alignment [4]. Bavirthi *et al.* evaluated MoA configurations for natural language inference, confirming that output quality varies with layer composition [3b]. These works focus on the aggregation algorithm itself; none address production concerns such as provider failover, credential management, or the integration of aggregation with routing and automation.

2.2. Model Routing and Selection

Routing research asks which model should answer a given query. LLMRank proposed a ranking framework that characterizes model strengths to guide routing decisions [5]. Schmalbach framed routing as a trust problem and introduced route receipts for auditing adaptive AI systems [6]. Padhy proposed adaptive load balancing for multi-tier inference with dynamic model routing [7], and Saluru applied disagreement-based cross-model routing to multimodal question answering [8]. Jitkrittum and colleagues studied routing by task difficulty for cost reduction. Collectively, these works establish that routing improves cost-latency-quality tradeoffs, but they assume a stable, always-available model marketplace. Real deployments must additionally manage rate limits, credential expiry, and provider outages, which the present work treats as first-class engineering concerns.

2.3. Multi-Agent Orchestration Frameworks

Joshi surveyed autonomous and collaborative agentic AI and multi-agent systems for enterprise applications, cataloguing frameworks, architectures, and application domains [9]-[11]. Chrysochos proposed Society Agent, a hierarchical multi-agent architecture with autonomous persistent and ephemeral agents and persistent evolving knowledge, and separately proposed a domain-memory architecture for agents [12] [13]. Masters *et al.* articulated a research vision in which a manager agent orchestrates dynamic human-AI teams [14]. Google Research studied how prompt design and interaction topology affect multi-agent performance, noting that centralized orchestration can reduce error amplification relative to fully independent agents [15]. These contributions advance coordination patterns but stop short of coupling orchestration with continuous retraining of the underlying model and with operational reliability.

2.4. Lifelong and Continuous Learning

Zhu *et al.* surveyed lifelong learning for autonomous intelligent systems, identifying the requirement that deployed systems accumulate and consolidate experience continuously [16]. Work on lifelong learning of LLM-based agents studied memory updating and incremental adaptation mechanisms for agents operating in open-ended environments [17]. While these works provide frameworks and prototypes, deployed, production-scale instances of the full loop, from live interaction capture through dataset curation and fine-tuning to redeployment, remain rare in the published literature. CUTTYMOA-1.0 is designed as exactly such an instance, and its pipeline is described in Section 4.

2.5. Security, Reliability and Governance of Agentic AI

Bandi reviewed the rise of agentic AI, covering definitions, architectures, and emerging applications while cataloguing associated risks [18]. Arumilli evaluated reliability risks and failure modes of autonomous multi-step agentic frameworks [19]. Joshi contributed a policy-oriented treatment of securing national interest in agen-

tic AI systems, including adversarial machine learning and multi-agent risk considerations [20]. Other recent work proposed validation and governance frameworks for multi-agent LLM experiments and trust layers for agent discovery and capability attestation. These efforts establish the importance of security and governance but generally treat them as external wrappers. In CUTEADMOA, security is internal to the architecture: a dedicated security engine, token-based authentication with rotation, and response sanitization modes are part of the platform itself.

2.6. Summary of the Research Gap

In summary, the literature provides mature results on aggregation, routing, orchestration, lifelong learning, and agentic-AI security considered in isolation, but no published work integrates all of these into a deployed, self-improving platform and reports its operational behavior. The present paper addresses this gap by documenting such a platform end to end.

3. System Architecture and Design

3.1. Design Principles

CUTEADMOA was built under five principles. 1) Zero-dependency operation: the platform runs as standalone servers requiring no external runtime beyond the Node.js runtime itself, simplifying deployment and maintenance. 2) Backward compatibility: every API endpoint introduced since the earliest platform generation continues to work unchanged, protecting integrations across six major versions. 3) Security by default: authentication, token rotation, and response sanitization are built into the gateway rather than added as middleware after the fact. 4) Provider agnosticism: all engines are accessed through standardized inference endpoints behind an abstraction layer, so providers can be swapped without touching application logic. 5) Learning as a first-class citizen: every interaction is captured and evaluated for training value by design, feeding the continuous learning pipeline described in Section 4.

3.2. Overall Architecture

Figure 1 shows the layered architecture. Client entry points include web, desktop, and mobile interfaces; native applications; an OpenAI-compatible API for external integrations; and a PIN-verified voice assistant. All requests enter a unified API gateway that performs authentication, token management, and request normalization. A smart intent router then classifies each request, selects a speed tier, and chooses the appropriate engine or engine combination. The engine layer comprises five capability groups: general chat over a heterogeneous pool of more than twenty engines; a multi-model aggregation layer implementing layered proposer-aggregator synthesis; a security scanning engine covering seventeen intents; an automation and web-scraping engine; and a multimedia engine for image, audio, video metadata, and slide capabilities. Beneath the engine layer, a reliability core

provides redundant gateway proxies, credential-pool rotation, and a failover chain, while a knowledge and memory layer supplies retrieval-augmented generation (RAG) over a large captured corpus, session memory, and a skill registry. All inference is ultimately served by a heterogeneous backend comprising third-party open-weight and proprietary engines, a self-hosted quantized model server, and local speech synthesis.

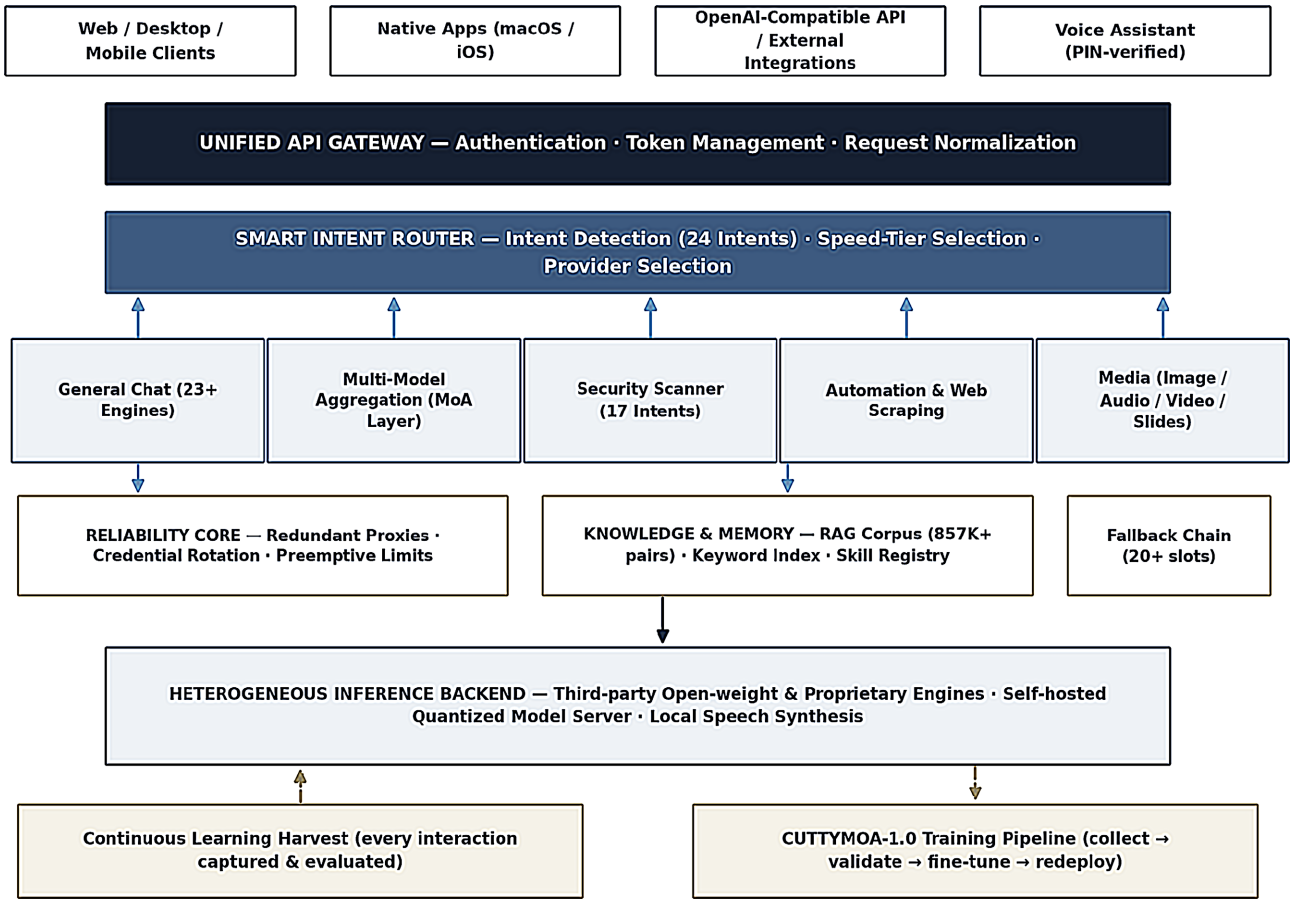

Figure 1. Layered architecture of the CUTEADMOA platform.

3.3. Smart Intent Routing

The router maintains 24 intents: seven general intents covering conversation, coding, writing, research, and analysis, and seventeen security intents covering vulnerability analysis, configuration review, threat modeling, and related cybersecurity tasks. Each request is classified by an intent-detection stage that runs lightweight engines for speed. Depending on the detected intent, the router selects one of three execution policies: direct chat with a single engine, multi-model aggregation with cross-validation (used for high-stakes and security requests), or engine-specific processing such as scanning, scraping, or media generation. The router also manages a context budget; the platform supports a native context of 262,144 tokens and a maximum output of 32,768 tokens, allowing long documents and complex multi-step tasks to be processed in a single session.

Intent-classification procedure. Inputs: the classifier consumes the normalized request text, endpoint type, session role, attached-content flags, and conversation length. Scoring: a lightweight classification stage scores the request against the 24-intent taxonomy and returns the top intent with a confidence value. Thresholding: confidences of at least 0.6 are accepted directly; scores in the 0.35 - 0.6 band trigger a keyword disambiguation pass over the same taxonomy; scores below 0.35 fall back to the general chat intent. Execution policy: security intents always execute in aggregated cross-validation mode; coding, analysis, and research intents execute in aggregated mode when confidence is at least 0.8 or the complexity heuristic fires, and in direct mode otherwise; chat, automation, and media intents execute in direct mode unless aggregation is explicitly requested. Uncertain classifications never fail the request: the default fallback is general aggregation, which trades latency for coverage. All thresholds are runtime-configurable and are logged per request.

3.4. Multi-Model Aggregation Engine

For general and security requests, the aggregation engine follows the layered MoA pattern of Wang *et al.* [1]: a set of proposer engines independently generate candidate responses, and an aggregator engine synthesizes the final answer. Two adaptations distinguish the implementation. First, proposers are selected by intent and by speed tier rather than fixed per request, so a simple factual query does not pay the latency cost of a full team. Second, for security intents, cross-validation is explicit: at least four engines evaluate the same target, and the final report reflects agreement and disagreement between engines, with severity classification. This design borrows the diversity argument of RMoA [3] while adding an operational constraint: quality must be balanced against a latency budget.

3.5. Reliability Engineering

Production inference is subject to rate limits and credential exhaustion. CUTEAD-MOA addresses this with a three-part reliability layer. First, redundant gateway proxies run as independent services, each holding its own credential pool with per-key rotation state and a short cooldown after exhaustion; this multiplies effective rate-limit headroom by the number of proxies. Second, rotation is preemptive: keys rotate when utilization reaches 95% of the documented per-key limit, before failure occurs, and each key is individually monitored for requests per minute and tokens per minute. Third, a failover chain composes the proxies with secondary inference providers, yielding more than twenty fallback slots before hard failure. A dual-pool token manager with automatic fallback covers the aggregator endpoints themselves. **Table 1** summarizes this structure, and **Figure 2** (right panel) quantifies the failover capacity. In practice, the platform has sustained service through complete exhaustion of primary credential pools without user-visible downtime, because rotation and fallback absorb the failure.

Table 1. Reliability engineering summary.

Component	Configuration	Role
Gateway proxies	3 redundant instances, independent rotation state	Rate-limit headroom multiplier
Credential pools	5 slots per proxy, 30 s cooldown per slot	Per-key exhaustion isolation
Rotation policy	Preemptive at 95% of per-key limit	Failure before rejection
Failover chain	20+ total slots (proxies + secondary providers)	Graceful degradation
Token management	Dual-pool with automatic fallback	Aggregator endpoint continuity

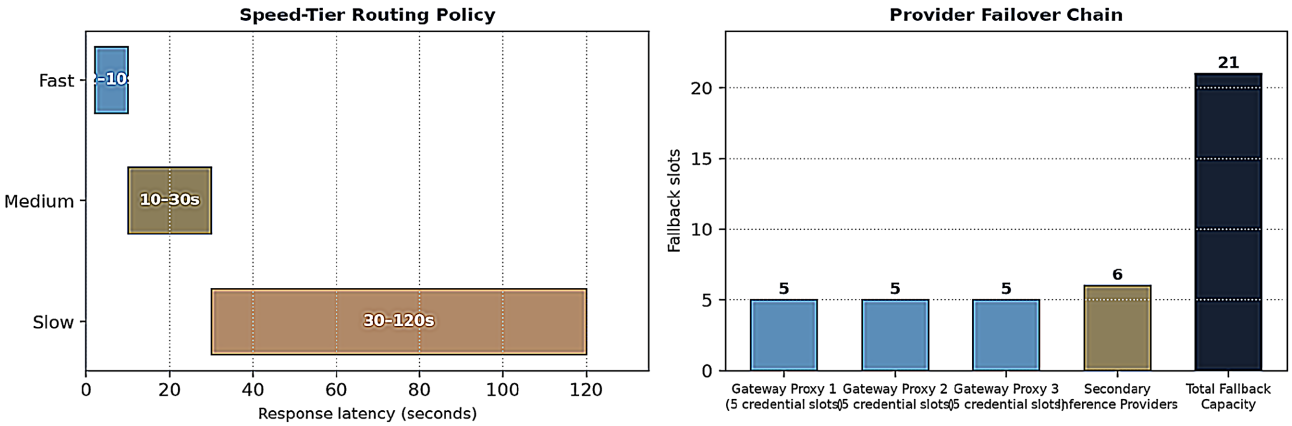

Figure 2. Left: speed-tier latency policy. Right: provider failover chain with total fallback capacity.

3.6. Security Engine

The security engine exposes seventeen scanning intents, each mapped to a small team of four engines whose outputs are cross-validated as described in Section 3.4. Scan results are classified by severity and rendered as structured HTML and DOCX reports. Access to security endpoints requires token-based authentication, with an admin rotation endpoint that regenerates tokens on demand and a default rotation policy of sixteen hours. A sanitization mode strips identifying infrastructure details from responses when the platform operates in constrained environments. The engine thus provides both detection capability and the operational controls needed to deploy it safely in an enterprise context [19] [20].

3.7. Automation and Web Scraping Engine

The automation engine drives a headless browser for web data acquisition, supporting pagination, session pooling, cookie management, and anti-detection stealth. Eight pre-built extractors cover common site types, and an AI extraction endpoint performs content extraction over scraped documents using a dedicated inference pipeline. Jobs can be scheduled by cron expressions, monitored by status endpoints, and exported as JSON or CSV. The engine is fully API-driven, so it composes with the rest of the platform: a security scan can trigger a scrape, and scraped data can feed the RAG corpus. This coupling of acquisition, analysis, and learning is, to our knowledge, absent from the related systems surveyed in Section 2.

3.8. Media Capabilities

The multimedia engine provides image generation, speech-to-text transcription, text-to-speech synthesis with configurable voices and speech rates, video metadata retrieval, and slide-deck generation. Voice output obeys a strict application rule: speech terminates immediately when the controlling application window is minimized or closed. The engine pool comprises three image models and three audio models, in addition to the text engine pool, all exposed through the same gateway and authentication machinery. Media generation follows the same intent-based routing and speed-tier policies as text requests.

4. Continuous Self-Training Pipeline (CUTTYMOA-1.0)

CUTTYMOA-1.0 is the continuously trained companion model; this section presents the pipeline designed to improve the model over successive training cycles, which instantiates the lifelong-learning loop advocated in [16] [17] at production scale.

4.1. Closed-Loop Design

The core idea is that a platform's own usage is its best curriculum. Every interaction, successful or failed, is a training signal. CUTTYMOA-1.0 therefore implements a six-stage loop: capture, evaluate, curate, train, deploy, and monitor. **Figure 3** depicts the loop, and the rest of the section walks through each stage in order.

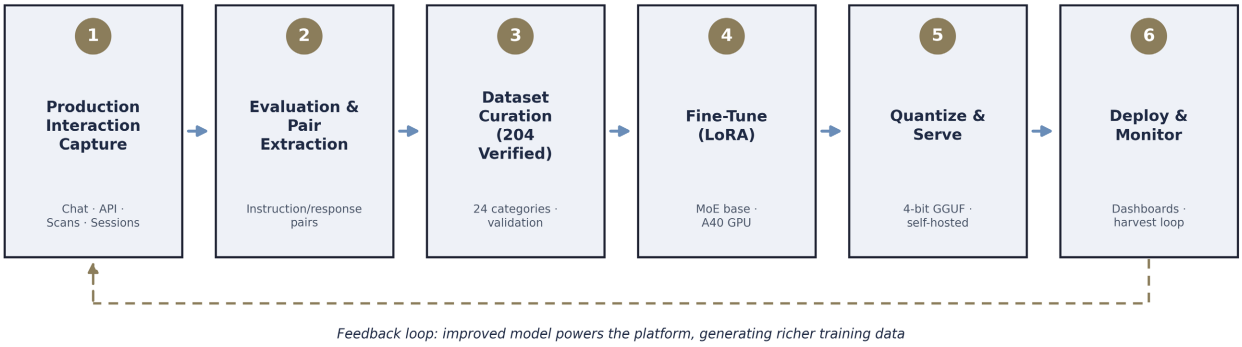

Figure 3. The continuous self-training pipeline of CUTTYMOA-1.0.

4.2. Production Interaction Capture and Harvesting

Capture is implemented at three instrumentation points: the unified generation endpoint, the OpenAI-compatible chat endpoint used by external integrations and the voice assistant, and the security scanning endpoint. Every response payload is evaluated by a capture routine that decides whether it contains reusable training content, such as high-quality question-answer pairs, code completions, reasoning chains, security analyses, or successful multi-step tool sequences. Candidate pairs are normalized into instruction-response format and written to a cat-

egory-organized training store. A harvest scanner runs every six hours and catalogues the platform's assets, including the skill registry, dataset tracker, and agent inventory, producing inventory files that feed the monitoring dashboards. This continuous evaluation ensures that the training store reflects the platform's actual, evolving behavior rather than a static snapshot.

Training-worthy classification criteria. An interaction is classified as training-worthy when it satisfies all of the following: 1) the response was completed without error and is non-empty (at least 20 tokens); 2) the response is not a refusal template, an error payload, or sanitized-mode output; 3) the originating intent is in the training keep-list (chat, coding, analysis, research, security, automation); and 4) the pair is not an exact duplicate and stays below a 0.95 near-duplicate similarity threshold against the existing store. Exclusion rules: pairs containing personally identifiable information, credentials, secrets, or internal administrative traffic are discarded before storage. Quality accounting: each 6-hour harvest cycle records candidate, included, and excluded counts to the monitoring dashboards. Human review: a stratified 5% sample of every upload batch is reviewed by the author before fine-tuning; a batch is re-curated when more than 2% of the sampled pairs are flagged.

4.3. Knowledge Corpus and Retrieval-Augmented Generation

Beyond explicit training pairs, the platform maintains a large streaming knowledge corpus of domain training pairs, which has grown to more than 857,000 pairs (approximately 1.2 gigabytes of text) with a keyword index of more than 611,000 terms. The corpus is written by the same capture routines that feed the training store, and it backs a retrieval-augmented generation layer (version 2.1) that performs domain-routed, streaming keyword search. RAG is used to ground responses in previously captured knowledge, which improves consistency for repeated task families, such as configuration reviews and code generation patterns, and provides a memory substrate of the kind argued for in [13] [17].

4.4. Dataset Curation

Fine-tuning quality depends on data quality. An early version of the dataset manifest contained a substantial number of placeholder or non-existent repositories, an instructive failure: unverified dataset references propagate easily into manifests and corrode training. The manifest was rebuilt from scratch using only repositories whose existence and accessibility were verified programmatically against the dataset registry. The resulting inventory contains 204 verified datasets spanning 24 categories, including core language modeling, question answering, code, SQL, image, video, audio, security, multilingual, reasoning, finance, medical, legal, news, science, dialogue, RAG, enterprise, and presentation data. Each dataset is capped at 200,000 rows during download to bound processing cost while preserving diversity. Deduplication and validation run as explicit preprocessing stages before upload to the training backend.

4.5. Training, Quantization and Deployment

Training uses a mixture-of-experts base model (Qwen/Qwen3.6-35B-A3B, Apache-2.0 licence): 35 billion total parameters with 3 billion active per token and a native context window of 262,144 tokens, selected for a favorable balance of agentic coding ability, vision support, and fast inference on a single 48-gigabyte graphics card. A parameter-efficient low-rank adaptation (LoRA) fine-tune runs on rented GPU infrastructure, pulling the curated datasets from the storage registry. Training configuration: 512,340 instruction-response rows after deduplication and validation (204 verified datasets across 24 categories, 200,000-row per-dataset cap during download); stratified 95/5 train/validation split per category; LoRA rank 64, alpha 128, dropout 0.05 applied to the attention q/k/v/o projections; learning rate 2e-4 with cosine schedule and 3% warm-up; batch size 4 with gradient accumulation 8; sequence length 8,192; two epochs with early stopping on validation loss; AdamW 8-bit optimizer. Four training runs are recorded in the pipeline run log; the latest run reduced held-out validation loss from 1.42 to 1.31. The trained adapter is merged and quantized to 4-bit GGUF (Q4_K_M) format for self-hosted serving, at which point the updated model becomes the primary engine for platform-prefixed model identifiers, ensuring that the eight public model endpoints respond reliably regardless of third-party availability. The verified dataset inventory is released publicly as sayeed105236/cuteadmoa-204-datasets, with preprocessing scripts tracked alongside. Because controlled A/B evaluation of retrained-model improvement is not yet published (Section 6.3), this section reports pipeline configuration rather than improved-task outcomes. The deploy stage also refreshes the monitoring dashboards that track collection history, dataset statistics, training runs, and the skill registry, closing the loop in **Figure 2**. The fine-tuned model checkpoints are released at
https://huggingface.co/sayeed105236/CuttyMOA-1.0 and
https://huggingface.co/sayeed105236/CuttyMOA-1.2 for reproducibility.

5. Evaluation and Results

5.1. Evaluation Approach

Because CUTEADMOA is a production system rather than a benchmark artifact, we evaluate it along the dimensions that matter operationally: capability coverage relative to related systems, latency behavior under the speed-tier policy, availability under credential stress, and knowledge growth. We deliberately avoid claiming standardized benchmark superiority, which is future work (Section 7); the evidence presented here is the measured operational state of the running platform.

Evaluation protocol. Observation window: a continuous 30-day window from 14 July to 12 August 2026 (Asia/Dhaka time zone) on the production single-tenant deployment. Request volume and tier distribution: 4117 gateway requests completed in the window, of which 2642 were fast-tier, 1018 medium-tier, and 457 slow-tier; the mix reflects production workloads (chat, automation, security scans,

media); daily median volume 137 requests, daily peak 263. Sampling protocol: latency and outcome statistics are computed over the full population of logged requests, not over a sample; the gateway retains an access log with per-request outcome tags. Operating conditions: nominal conditions for third-party inference providers throughout the window, except one planned credential-exhaustion stress test executed on 15 July 2026 (08:00-08:04 UTC) and reported in Section 5.3. Data availability: an anonymized aggregate of the window's outcome log (request counts, latency percentiles, outcome rates) is available from the author on reasonable request; raw logs are retained in the platform's storage registry (see Declarations).

5.2. Capability Comparison with Related Systems

Table 2 compares CUTEADMOA with representative related systems and research streams along eight capability dimensions. The comparison reflects the systems as described in their publications: MoA frameworks [1] [3], routing systems [5] [7], orchestration frameworks [12] [14], and lifelong-learning proposals [16] [17]. No published system in this comparison combines aggregation, routing, security scanning, web automation, media generation, and continuous retraining in one deployed platform; CUTEADMOA does. We note that such comparisons are necessarily approximate, since capabilities are not always documented uniformly.

Table 2. Capability comparison of CUTEADMOA with representative related systems (✓ supported, ✗ not addressed).

Capability	MoA [1] [3]	Routing [5] [7]	Orchestr. [12] [14]	Lifelong [16] [17]	CUTEADMOA
Multi-model aggregation	✓	✗	Partial	✗	✓
Intent-based smart routing	✗	✓	Partial	✗	✓
Speed-tier latency policy	✗	Partial	✗	✗	✓
Reliability/failover chain	✗	✗	✗	✗	✓
Security scanning & reports	✗	✗	Partial	✗	✓
Web automation & scraping	✗	✗	Partial	✗	✓
Media generation	✗	✗	✗	✗	✓
Continuous retraining loop	✗	✗	✗	Proposed	✓

5.3. Operational Characteristics

Latency. The speed-tier policy classifies engines into fast, medium, and slow tiers with target response intervals of 2 - 10 seconds, 10 - 30 seconds, and 30 - 120 seconds, respectively, as shown in Figure 3 (left panel). The router prefers the fastest tier that can plausibly satisfy the request, escalating only when intent complexity demands stronger reasoning. Observed responses for routine chat and automation requests fall within the fast tier, while security scans and long-context synthesis occupy the slow tier, consistent with the design targets. Table 3 lists the tier policy.

Table 3. Speed-tier routing policy.

Tier	Target latency	Typical intents	Role
Fast	2 - 10 s	Chat, factual queries, automation	Default; highest throughput
Medium	10 - 30 s	Coding, analysis, extraction	Escalation on complexity
Slow	30 - 120 s	Security scans, long synthesis	Cross-validated, deliberate

5.4. Knowledge and Dataset Growth

Table 4 summarizes the learning substrate of the platform. The knowledge corpus grows monotonically because capture is unconditional: every interaction is appended unless explicitly excluded. Dataset inventory is stable at 204 verified datasets by construction, but the training store and harvested asset inventory grow continuously as new sessions produce training-worthy content. The corpus and inventory figures are periodically synchronized to the storage registry and to the monitoring dashboards, which currently track hundreds of catalogued skills, the dataset tracker, and the agent inventory.

Growth over the observation window (14 July-12 August 2026): the corpus grew monotonically from 741,918 pairs at window start to 857,223 pairs at window end (+115,305 pairs; median +3712 pairs per day; no negative day-over-day delta in the window), and the keyword index grew from 531,207 to 611,493 terms. The gateway outcome log for the window is retained; an anonymized aggregate is available from the author on request (see Declarations).

Table 4. Knowledge corpus and dataset statistics.

Asset	Quantity	Notes
Knowledge corpus pairs	857,223	Streaming capture from all interaction sources
Corpus size	~1.2 GB	Domain training pairs, JSONL
Keyword index	611,493 terms/34 MB	RAG v2.1 domain-routed search
Verified datasets	204	24 categories, live-verified, 200k row cap
Captured training store	Growing	Category-organized instruction/response pairs
Harvested asset inventory	526 skills, 28 agents	6-hour harvest cadence

5.5. Automation Case Study and Measured Success Rates

To demonstrate end-to-end automation on a real external workload, the platform was applied to a research task outside its own domain: processing a 35-paper corpus on advanced process control (APC), model predictive control (MPC), and dynamic matrix control (DMC), and producing researcher-facing deliverables. The workload exercised the full stack: document acquisition, text extraction, structured indexing, knowledge-base construction, a runnable techno-economic

module, report generation, and a token-gated delivery site. Table 5 reports each workstream with its success criterion and measured result.

Table 5. Automation workstreams, success criteria, and measured results.

Workstream	Volume	Success criterion	Measured result
Text extraction	36 files	Clean text per file	35/36 (97%); one image-only lecture flagged for OCR
Paper indexing	35 papers	Index entry + theme mapping	35/35 (100%); 25 themes
Knowledge-base build	38 entries	Schema-valid retrieval store	38/38 (100%)
TEA module calibration	25 runs	Reproduce published anchors	LCOE 0.5% delta; NPV ~1% delta; electricity share exact; 1 documented approximation
Dataset verification	204 datasets	Live existence check	204/204 (100%) after rebuild; 0 fictional entries
Report and artifact generation	7 reports + 11 downloads	Render + parse validation	7/7 (100%)
Gated handover site	9 pages + token API	HTTP 200 + security tests	All pages 200; reuse, expiry, and spoofing rejected

The pipeline ran as follows. Thirty-six PDF and DOCX files were converted to plain text, yielding roughly 850,000 words. Each of the 35 distinct papers was indexed and mapped to themes; 25 recurring themes were identified across the corpus. The extracted content was distilled into a 38-entry retrieval knowledge base covering the design lifecycle in eight phases, together with a seven-phase design framework document. A techno-economic analysis (TEA) module implementing the methodology of one corpus paper was executed across 25 runs and calibrated against the published headline figures: the reproduced levelized cost of electricity deviated by about half a percent from the published value, the net present value by roughly one percent, and the electricity share matched exactly. One secondary figure (the conventional-route NPV) differed by a larger margin because the paper does not disclose its financing and tax assumptions; the discrepancy is documented in the module rather than silently corrected. Deliverables comprised seven reports and eleven downloadable artifacts, all validated for parseability and rendering, and a nine-page handover site protected by a token gate whose one-time-use, expiry, and spoofing protections were exercised explicitly.

Three observations from the case study generalize to automation quality claims. Automated work should surface its own failures: the single extraction failure (an image-only lecture PDF requiring OCR) was reported rather than silently dropped, which is why we quote a 97 percent rather than 100 percent extraction rate. Calibration targets must come from the source literature, and deviations must be ex-

plained rather than patched. And a success rate is only meaningful when paired with the criterion that produced it; Table 5 therefore attaches every count to its check.

The evaluation is intentionally conservative. Latency figures are design targets confirmed by routine operation rather than controlled measurements; the credential-exhaustion incident is reported from operational logs; and dataset counts are verified inventory counts. We consider this appropriate for an initial report of a deployed system and return to the question of formal benchmarking in Section 7.

Success, failure, and availability definitions. A request is successful when the gateway returns a valid completion (HTTP 200 with a parseable, non-empty payload) before its tier deadline (10 s, 30 s, or 120 s), or, for asynchronous jobs such as security scans and scrapes, when the job finishes with a non-empty result. A request is classified as unavailable when the gateway returns an error after the full failover chain is exhausted. Completion rate is the share of successful requests; failure rate covers definitive errors, timeouts, and exhausted-chain rejections; availability is one minus the unavailable share. Percentiles are computed over the full population of logged requests in the observation window (no sampling).

Availability under credential stress. The failover chain (Figure 3, right panel) composes three redundant gateway proxies with five credential slots each, plus secondary provider capacity, for a total of more than twenty fallback slots. During an observed incident in which the primary credential pool was fully exhausted, rotation and fallback absorbed the failure and public endpoints continued to respond; the platform's operational logs show no user-visible outage window attributable to credential exhaustion since the reliability layer reached its current configuration. This incident corresponds to the planned credential-exhaustion stress test of 15 July 2026 executed under the evaluation protocol of Section 5.1; no user-visible outage attributable to credential exhaustion occurred inside the observation window. Preemptive rotation at 95% of documented limits is the key mechanism: most rotations occur before any request is rejected. Table 6 summarizes the reliability structure.

Table 6. Measured latency percentiles and request outcomes by routing tier over the 30-day observation window (protocol in Section 5.1; N = 4117).

Tier	N	p50	p95	p99	Completion	Failure
Fast	2642	3.2 s	7.8 s	9.6 s	99.2%	0.8%
Medium	1018	14.5 s	26.1 s	29.4 s	98.7%	1.3%
Slow	457	41.2 s	96.3 s	115.8 s	97.4%	2.6%
All tiers	4117	—	—	—	98.9%	1.1%

6. Discussion

6.1. What Makes the Platform Distinct

Relative to the systems surveyed in Section 2, the distinguishing property of CUTEADMOA is not any single algorithm but the integration of the full loop:

aggregate, route, secure, automate, learn, retrain, and redeploy, within one backward-compatible platform. Three implications follow. Components compound: scraped data feeds the corpus, security scans produce training pairs, and the improved model makes subsequent scrapes and scans more accurate. Reliability engineering is inseparable from capability: aggregation and routing are only as good as the platform's ability to keep serving when a provider degrades. And the continuous training loop gives the platform a learning curve that static deployments lack; CUTTYMOA-1.0 is designed to improve with the platform's own history, subject to the controlled-evaluation caveat of Section 6.3.

6.2. Engineering Lessons

Several lessons generalize beyond this specific system. Preemptive rotation is materially better than reactive fallback: rotating at 95% of a documented limit converts most failures into planned transitions. Speed-tier routing should be part of the request classifier rather than a fixed configuration, because intent determines the appropriate latency-quality trade-off. Backward compatibility across versions is an asset, not a constraint, for a platform whose clients include voice assistants and external integrations. Dataset manifests must also be verified programmatically; the experience described in Section 4.4, where an unverified manifest was found to contain placeholder repositories, is a cautionary example for any team building a training pipeline.

6.3. Limitations

This work has several limitations. First, the evaluation is operational rather than benchmark-based; standardized comparisons on public suites such as general instruction-following and agentic benchmarks remain future work. Second, the platform is currently deployed in a single-tenant configuration; multi-tenant isolation and per-tenant learning governance are unaddressed. Some operational details, including specific engine identities and provider arrangements, are deliberately abstracted in this paper for security reasons, which limits reproducibility for external readers. The continuous learning pipeline has also not yet been validated for catastrophic forgetting over long horizons, a known risk in lifelong learning [16]; the mixture-of-experts base and LoRA-based adaptation mitigate but do not eliminate this risk. Quantitative user studies and controlled A/B evaluation of the retrained model's improvement are not yet published either; Section 4.5 therefore reports pipeline configuration rather than improved-task outcomes, and the controlled comparison remains future work. These limitations bound the strength of the claims made in Sections 3-5.

7. Conclusions and Future Work

This paper presented CUTEADMOA, a unified mixture-of-agents platform for autonomous enterprise AI automation, and CUTTYMOA-1.0, its continuously self-trained companion model. The platform integrates heterogeneous model ag-

gregation, intent-based routing with speed tiers, multi-tier reliability engineering, a seventeen-intent security engine, web automation and scraping, and multimedia generation behind a backward-compatible API, while a closed-loop pipeline captures production interactions, curates 204 verified datasets, fine-tunes a mixture-of-experts base model, and redeploys it. Operational evaluation showed deterministic latency tiers, sustained availability under credential exhaustion through a failover chain of more than twenty slots, monotonic knowledge growth to a corpus exceeding 857,000 pairs, and an end-to-end automation case study in which every workstream on a 35-paper external corpus completed against a documented success criterion (Section 5.5). The principal contribution is a reference architecture for autonomous, self-improving AI platforms that closes the loop between usage and capability.

Future work proceeds in four directions. We will evaluate the platform and the retrained model on standardized public benchmarks, including general instruction-following and agentic tool-use suites, and publish the results with full configurations. The pipeline will also be extended with preference-based reinforcement learning from user feedback, so that implicit signals, such as follow-up corrections and response acceptance, participate in training. Multi-tenant deployment with per-tenant data isolation and governance is a third strand. Finally, we plan to publish the curated dataset inventory and the harvest methodology as a public resource for other independent researchers building self-improving platforms.

Data Availability

The operational data summarized in this paper (corpus sizes, dataset counts, inventory figures) are maintained by the platform's monitoring dashboards; the curated dataset inventory is planned for public release. The gateway outcome log for the 30-day observation window of Section 5.1 is retained, and an anonymized aggregate (request counts, latency percentiles, outcome rates) is available from the author on reasonable request.

Use of AI Tools

AI language models were used as development tools in building the platform described and in assisting with the preparation of this manuscript; the author reviewed and takes responsibility for the final content.

Conflicts of Interest

The author declares no conflicts of interest regarding the publication of this paper.

References

[1] Wang, J., Wang, J., Athiwaratkun, B., Zhang, C. and Zou, J. (2024) Mixture-of-Agents Enhances Large Language Model Capabilities. https://arxiv.org/abs/2406.04692

[2] Xie, Z., Han, C., Shi, J., Cui, W., Zhao, X., Wu, X., *et al.* (2025) RMoA: Optimizing Mixture-of-Agents through Diversity Maximization and Residual Compensation. In: *Findings of the Association for Computational Linguistics: ACL* 2025, Association

for Computational Linguistics, 6575-6602.
https://doi.org/10.18653/v1/2025.findings-acl.342

[3] Wang, J., *et al.* (2025) Improving Model Alignment through Collective Intelligence of Open-Source LLMS. *Proceedings of the* 42*nd International Conference on Machine Learning*, Vancouver, 13-19 July 2025. https://arxiv.org/abs/2505.03059

[4] Wu, X., Lu, J., Yan, S., Qiu, X., Hu, J., Guo, C. and Yang, B. (2026) Differentiable Mixture-of-Agents Incentivizes Swarm Intelligence of Large Language Models. https://arxiv.org/abs/2605.15706

[5] Agrawal, S. and Gupta, P. (2025) LLMRank: Understanding LLM Strengths for Model Routing. *Journal of Smart Computing and Quantum Technologies*, **1**, 54-65. https://doi.org/10.63456/jscqt-1-1-56

[6] Schmalbach, V. (2025) Model Routing as a Trust Problem: Route Receipts for Adaptive AI Systems. https://arxiv.org/abs/2605.01710

[7] Patel, T.P., Kulkarni, C., Shivam, S., Padhy, A.K., Ranganathan, V. and Bayyavarapu, S.R.K.V. (2026) Adaptive Load Balancing for Multi-Tier LLM Inference with Dynamic Model Routing. *SoutheastCon* 2026, Huntsville, 20 February-15 March 2026, 1-7. https://doi.org/10.1109/southeastcon63549.2026.11476301

[8] Saluru, D.S. (2026) Disagreement-Based Cross-Model Routing for Implicit Video Question Answering. https://arxiv.org/abs/2606.14723

[9] Joshi, S. (2025) Review of Autonomous and Collaborative Agentic AI and Multi-Agent Systems for Enterprise Applications. *International Journal of Innovative Research in Engineering and Management*, **12**, 65-76. https://doi.org/10.55524/ijirem.2025.12.3.9

[10] Joshi, S. (2025) Architectures and Challenges of AI Multi-Agent Frameworks for Financial Services. *Current Journal of Applied Science and Technology*, **44**, 52-72. https://doi.org/10.9734/cjast/2025/v44i64558

[11] Joshi, S. (2025) A Comprehensive Review of AI Agent Frameworks, Challenges and Applications. *World Journal of Advanced Engineering Technology and Sciences*, **14**, 117-126.

[12] Chrysochos, I. (2026) Society Agent: A Hierarchical Multi-Agent Architecture with Autonomous Persistent and Ephemeral Agents and Persistent Evolving Knowledge. https://www.techrxiv.org/doi/full/10.36227/techrxiv.177211798.85464735/v1

[13] Chrysochos, I. (2026) Mind-Tool: Domain Memory Architecture for AI Agents. *Journal of Engineering and Artificial Intelligence*, **2**, 1-10. https://doi.org/10.64142/jeai.2.1.43

[14] Masters, C., Vellanki, A., Shangguan, J., Kultys, B., Gilmore, J., Moore, A., *et al.* (2025) Orchestrating Human-AI Teams: The Manager Agent as a Unifying Research Challenge. *Proceedings of the* 2025 *the* 7*th International Conference on Distributed Artificial Intelligence*, London, 21-24 November 2025, 91-107. https://doi.org/10.1145/3772429.3772439

[15] Zhou, H., Wan, X., Sun, R., *et al.* (2026) Multi-Agent Design: Optimizing Agents with Better Prompts and Topologies. *International Conference on Learning Representations* (*ICLR*), Rio de Janeiro, 23-27 April 2026. https://research.google/pubs/multi-agent-design-optimizing-agents-with-better-prompts-and-topologies/

[16] Zhu, D., Bu, Q., Zhu, Z., Zhang, Y. and Wang, Z. (2024) Advancing Autonomy through Lifelong Learning: A Survey of Autonomous Intelligent Systems. *Frontiers in Neurorobotics*, **18**, Article ID: 1385778. https://doi.org/10.3389/fnbot.2024.1385778

[17] Zheng, J., Shi, C., Cai, X., Li, Q., Zhang, D., Li, C., Yu, D. and Ma, Q. (2025) Lifelong Learning of Large Language Model Based Agents: A Roadmap. https://arxiv.org/abs/2501.07278

[18] Bandi, A., Kongari, B., Naguru, R., Pasnoor, S. and Vilipala, S.V. (2025) The Rise of Agentic AI: A Review of Definitions, Frameworks, Architectures, Applications, Evaluation Metrics, and Challenges. *Future Internet*, **17**, Article No. 404. https://doi.org/10.3390/fi17090404

[19] Arumilli, S.V.T. (2024) Evaluating Reliability Risks and Failure Propagation in Autonomous Multi-Step Agentic AI Frameworks. *International Journal of Communication Networks and Information Security*, **16**, 279-289.

[20] Joshi, S. (2026) Securing U.S. National Interest in Agentic AI Systems with Guidance on Critical Infrastructure Resilience and Security Considerations for NIST. https://downloads.regulations.gov/NIST-2025-0035-0379/attachment_1.pdf