Journal of Computer Science and Technology Studies
ISSN: 2709-104X
DOI: 10.32996/jcsts
Journal Homepage: www.al-kindipublisher.com/index.php/jcsts

JCSTS
AL-KINDI CENTER FOR RESEARCH AND DEVELOPMENT

| RESEARCH ARTICLE

Agentic AI Modernization: Transforming Institutional Infrastructure Through Orchestrated Multi-Agent LLM Framework

Mahesh Kumar Damarched
Enterprise Programmer Analyst, University of Louisville, USA
Corresponding Author: Mahesh Kumar Damarched, **E-mail**: mahesh.damarched@gmail.com

| ABSTRACT

While managing constrained funds and strict regulatory requirements, the higher education institutions are under unprecedented pressure to modernize outdated information systems, such as mainframe-based Student Information Systems (SIS), custom registration platforms, legacy Learning Management Systems (LMS) and Enterprise Resource Planning (ERP) deployments. The complexity of institutional governance is being overlooked by the conventional single-agent based approaches to legacy modernization, which is delaying the digital transformation and creating security vulnerabilities. In order to achieve end-to-end code analysis, intelligent planning, safe migration and rigorous validation through specialized agents coordinated by institutional governance patterns, this research presents a novel agentic architecture using multi-agent Large Language Model (LLM) frameworks, created especially for higher education legacy system modernization. Crucially, this research innovates the deployment process, where we suggest an on-premises implementation strategy that natively protects sensitive student and faculty data, while maintaining GDPR, CCPA and FERPA compliance. Which projects to be challenging by cloud-based solutions, as these introduce data residency and compliance complexities. For COBOL/MUMPS/PL-I legacy codebases, our research demonstrated 87% successful modernization rate, with a 65% decrease in manual intervention and a 78% improvement in documentation accuracy. By mapping multi-agent workflows to existing institutional governance structures, academic committees, change boards and divisional responsibility models, the framework accomplishes institutional alignment, thereby increasing the credibility and organizational compatibility of agentic modernization solutions. This study offers institutions a revolutionary route to modernization, that maintains institutional data sovereignty, while significantly cutting modernization timelines and costs by bridging cutting-edge AI research with useful higher education IT strategy.

| KEYWORDS

Agentic AI, Multi-Agent LLM Frameworks, Legacy System Modernization, Higher Education IT, Data Privacy Compliance, FERPA, GDPR, CCPA, On-Premises Deployment, Educational Technology Infrastructure, Artificial Intelligence, Large Language Models

| ARTICLE INFORMATION

ACCEPTED: 12 January 2026 **PUBLISHED:** 09 February 2026 **DOI:** 10.32996/jcsts.2026.8.4.1

1. INTRODUCTION

1.1 The Higher Education Legacy System Crisis

The field of higher education is at an inflection point [[1]]. Where outdated Student Information System (SIS) infrastructure has been cited, by 65% of higher education Chief Information Officers (CIOs), as the main obstacle to institutional digital transformation according to Gartner research [[2]]. This figure conceals a more complex reality, where many universities run interconnected ecosystems of legacy systems, that were developed over three to five decades, resulting in technical debt that currently accounts for 60–80% of institutional IT budgets [[3]]. These systems include IBM mainframe platforms running COBOL applications, custom-built registration systems written in MUMPS or PL/I, decades-old LMS customizations built on proprietary architectures, and enterprise deployments of Oracle PeopleSoft on aging infrastructure [[4]].

1.2 Compounding Regulatory and Compliance Pressures

Simultaneously, regulatory complexity has exponentially increased. Higher education institutions now navigate multiple, intersecting compliance frameworks [[5]]:

- **FERPA** (Family Educational Rights and Privacy Act, 1974): As the FERPA rule is very stringent and strictly mandated, the penalties for violating this law range from $8,000 to $75,000. It requires written consent before sharing personally identifiable information (PII) from educational records [[6]].
- **GDPR** (General Data Protection Regulation, 2018): Any institution that serves students from the European Union, mandatorily need to comply with the GDPR. Where any violations, attracts fine up to €20 million (≈ $23.7 million) or 4% of annual global revenue, whichever is higher [[7]].
- **CCPA** (California Consumer Privacy Act, 2020): Applies to institutions processing California residents' data, establishing consumer rights to access, delete, and opt-out of data sales [[8]].

Legacy systems lacked contemporary security controls, audit logging, encryption, infrastructure and data governance capabilities, because they were built before these regulations were in place. According to a 2023 study, outdated IT infrastructure was directly responsible for 50% of data breaches in higher education [[9]]. The regulatory risk is enormous: a single GDPR-violating data incident could result in fines that surpass an institutions yearly IT budget which would cause smaller regional universities to become insolvent.

1.3 The Institutional Trust Problem

These difficulties are made worse by the lack of trust. In order to pace up modernization, institutions have embraced cloud-based solutions (SaaS platforms, Platform-as-a-Service offerings, Infrastructure-as-a-Service deployments). However, they have found that vendor lock-in scenarios, data residency concerns and third-party access to sensitive student records cause governance and data security concerns among faculty administrators and families [[4]]. A 2024 Inside Higher Ed survey found that 68% of university presidents are worried about the unacceptable risks to data sovereignty that cloud adoption presents [[10]]. This lack of trust causes institutions to halt migration projects mid-cycle or require costly compliance audits from vendors which causes modernization timelines to be extended by 18–24 months [[11]].

1.4 The Agentic AI Opportunity

Agentic artificial intelligence frameworks, that coordinate several specialized Large Language Model (LLM) agents to independently analyze plan carry out and validate legacy system modernization, present a game-changing technological opportunity in this context [[12]]. Agentic architectures break down modernization into specialized workflows, each carried out by an agent, optimized for tasks like legacy code analysis, business rule extraction, safe code generation, regression testing and compliance validation [[13]][[14]]. This contrasts with traditional single-LLM approaches, that try to solve end-to-end modernization tasks within a monolithic conversational loop.

This paper contends that, institutional governance structures and agentic architectures are fundamentally conceptually compatible. Universities function through distributed responsibility committee oversight and division of labor, governance patterns that naturally occur in agentic systems. We suggest an alternative to compelling institutions, to adopt cloud-vendor governance: implement agentic LLM frameworks on-site in institutional data centers preserving total data sovereignty while utilizing AI-driven modernization capabilities [[15]].

1.5 Research Contribution and Innovation

This research makes three distinct contributions to the field:

1. **Architectural Framework**: We propose the first agentic LLM architecture, which is specifically designed for higher education legacy system modernization, incorporating agents for code analysis, planning generation, validation and compliance checking, coordinated through institutional governance patterns [[16]][[17]].

2. **On-Premises Deployment Model**: We demonstrate how to implement multi-agent LLM frameworks within on-premises infrastructure, upholding GDPR, CCPA and FERPA compliance, while safeguarding institutional data sovereignty going beyond theoretical agentic concepts [[18]].

3. **Empirical Validation**: In comparison to conventional modernization techniques, we exhibit that pilot implementations across diverse legacy codebases (COBOL mainframe SIS, custom MUMPS registration systems, Oracle PeopleSoft

deployments) result in 87% successful modernization rates, 65% less manual intervention and 78% better documentation accuracy [[19]][[20]].

This paper further addresses four specific research objectives:

1: Establish a conceptual framework for agentic approaches in higher education settings, by synthesizing recent research on multi-agent LLM frameworks and their application to legacy system modernization.

2: Provide an architectural framework for modernizing agentic legacy systems that is suited to institutional data governance requirements higher education governance structures and regulatory compliance frameworks.

3: Innovate on implementation, by showcasing the on-premises deployment capabilities of agentic LLM frameworks, that maintain full data sovereignty, while achieving GDPR CCPA and FERPA compliance.

4: Provide empirical evidence through pilot projects, that show how the suggested framework can be successfully applied to diverse legacy codebases, with measurable gains in modernization schedules accuracy and institutional trust.

2. LITERATURE REVIEW

2.1 Multi-Agent LLM Frameworks: Emerging Architectures

The foundation for this research rests on the recent breakthroughs in field of multi-agent LLM systems. Historically, Large Language Models used to operate as monolithic systems: a user would submit a query, one LLM would process it, and a response would be produced, in simpler way it can be explained as one prompt → one model → one response system [[21]]. This architecture proved adequate for static tasks but failed at complex, multi-step problems requiring different types of reasoning and specialized expertise [[22]].

2.1.1 AutoGen and Conversational Agent Orchestration

Microsoft's AutoGen framework surfaced as a pivotal contribution, which enabled multiple LLM agents to converse with each other, coordinate subtasks, and synthesize results [[23]]. With AutoGen, developers can deploy multiple LLM agents in various roles (such as analyst coder and reviewer), set them up to communicate in natural language and use human-in-the-loop oversight at crucial decision points [[24]]. According to a 2023 study that used AutoGen for software development tasks, multi-agent conversational workflows exhibited accelerated development cycles by 28% and decreased code defects by 34% when compared to single-agent implementations [[23]][[25]]. For higher education contexts, AutoGen's strength lies in its flexibility, where AutoGen agents can be tailored to match institutional governance patterns, enabling a "Code Analysis Committee Agent," a "Regulatory Compliance Agent," and a "Deployment Approval Agent" to interact as if mirroring actual institutional committees [[26]][[27]].

2.1.2 CrewAI and Role-Based Multi-Agent Collaboration

Developed by João Moura CrewAI focuses on role-based agent specialization. It allows developers to create agents with particular roles (e.g. Senior Software Architect, QA Engineer, DevOps Specialist), configure agents with role-appropriate tools and memory structures and organize teams of specialized agents toward common goals objectives [[28]][[29]]. Once code generation process is completed, CrewAI enables QA and DevOps agents to run concurrently, by differentiating between sequential and parallel agent workflows [[30]][[31]]. CrewAI's capabilities were demonstrated in a recent case study, that used it for financial system modernization, which reduced architectural misalignment incidents by 67% and accelerated migration planning by 89%, when compared to manual approaches [[32]].

2.2 Legacy System Modernization: Current Approaches and Limitations

2.2.1 Traditional Modernization Strategies

The legacy system modernization literature identifies three primary modernization strategies [[33]]:

- **Rehost ("Lift and Shift")**: This is the process of moving current systems to cloud infrastructure with little modification. Although this strategy reduces short-term risk, cloud-native benefits are not realized and technical debt is sustained [[34]].

- **Refactor**: The purpose is to reorganize code to maintain functional equivalency while enhancing quality readability and maintainability. Although this method enhances code quality, it requires a lot of manual labor and runs the risk of causing subtle behavioral changes [[35]].
- **Rewrite ("Greenfield")**: It uses contemporary technologies to complete reconstruction systems from the ground up. Although this method removes technical debt, it increases the risk of behavioral divergence requires a significant amount of work and often leads to project failure [[36]].

Every conventional method is limited by the knowledge engineering burden. A 3-million-line COBOL mainframe application must be modernized by removing decades-old business logic from convoluted code, comprehending implicit assumptions ingrained in mainframe architecture and translating this knowledge into contemporary languages and platforms. As COBOL programmers retire, an increasing number of legacy system experts are needed for this knowledge extraction [[37]][[38]].

2.2.2 AI-Assisted Modernization Emerging Research

Recent research demonstrates that, code translation and knowledge extraction can be partially automated by large language models. When it comes to producing syntactically correct code transformations, GitHub Copilot and other LLM-powered code generation tools achieve nearly 70-75% of accuracy [[39]][[40]]. Nevertheless, single-LLM approaches exhibit significant drawbacks: they produce code without verifying compliance with regulatory requirements, struggle with large codebases (context window limitations), and fail to ensure behavioral equivalency across complex control flows [[41]][[42]][[43]]. A 2024 study applying GPT-4 to COBOL, it was found that, while the model successfully translated 72% of function-level transformations, only 34% achieved end-to-end behavioral equivalency, when integrated with surrounding code and the model provided no compliance validation [[44]].

2.2.3 Higher Education-Specific Challenges

Higher education legacy system modernization presents domain-specific challenges absent from general enterprise modernization:

- **Regulatory Fragmentation**: State-specific privacy laws FERPA (federal), GDPR (if serving students from Europe) and CCPA (if serving students from California), all apply to institutions concurrently. This multijurisdictional landscape cannot be sufficiently addressed by a single vendor compliance template [[45]].
- **Institutional Governance Complexity**: Universities function as federated systems and their committee structures, administrative divisions, and faculty governance make modernization oversight requirements more complicated than corporate hierarchies [[46]].
- **Data Sovereignty Concerns**: Increasing faculty governance structures require that, student data should stay under institutional control, resulting in architectural requirements that are incompatible with vendor solutions that only use the cloud [[47]].
- **Heterogeneous Legacy Systems**: Universities frequently run widely diverse legacy systems, such as COBOL mainframes, custom MUMPS systems, and outdated LMS on proprietary platforms, in contrast to corporates with standardized technology stacks [[48]].

2.3 Agentic AI Architecture for Enterprise Systems: Emerging Consensus

2.3.1 Three-Tier Enterprise Agentic Architecture

Recent enterprise architecture research converges on a three-tier model for production agentic AI systems:

1. **Foundation Tier**: Establishes audit logging, security controls (such as role-based permissions and adversarial input detection), and governance infrastructure. By implementing "tool orchestration with enterprise security", this tier makes sure that agents are not allowed to go beyond their authority limits [[49]].

2. **Workflow Tier**: Provides automation using predetermined patterns, such as orchestrator-worker hierarchies, evaluator-optimizer loops, prompt chaining, routing, and parallelization [[50]][[51]][[52]]. This tier uses automated escalation procedures and mandatory checkpoints, to increase productivity while retaining human oversight [[53]].

3. **Autonomous Tier**: Makes use of learned optimization patterns and accumulated context, to enable agents to operate within limited domains, without human intervention. Only well-defined, low-risk operations should use this tier [[54]].

2.3.2 Institutional Governance Alignment

Research on enterprise agentic architecture reveals a subtle finding: agentic systems governance structures naturally implement mirror institutional governance structures [[55]]. Committees with specialized responsibilities such as, division of labor, checks and balances, and escalation procedures are already in place at universities. These patterns are implemented by agentic systems that feel "natural" to institutional stakeholders which increases the likelihood of adoption and stakeholder trust [[56]][[57]]. This governance-alignment insight is especially pertinent to the modernization of higher education, because institutions are more likely to adopt agentic systems that use well-known governance patterns since they already understand committee-based decision-making [[58]][[59]].

3. PROPOSED FRAMEWORK: AGENTIC MODERNIZATION ARCHITECTURE FOR HIGHER EDUCATION INSTITUTION

The main innovation of this research is presented in this section: an agentic LLM framework created especially for modernizing legacy systems in higher education with a focus on multi-regulatory compliance institutional governance alignment and on-premises deployment.

3.1 Architectural Overview

The proposed innovative framework consists of seven specialized agents, distributed across four-tiers, which are orchestrated through institutional governance patterns [Figure 1]:

3.1.1 Analysis Agent Tier: The Analysis tier comprises of two specialized agents:

- **COBOL/MUMPS/PL-I Parser Agent**: Produces intermediate representations (IRs) that capture control flow and data flow, analyzes legacy code, extracts syntactic structures, and finds program dependencies [[60]]. Millions of lines of code are processed by this agent, which extracts information necessary for intelligent planning, but not sufficient for direct translation [[61]] [Figure 1].

- **Business Rule Extractor Agent**: Examines institutional policies code and documentation, to extract implicit business logic that is then converted into explicit institution-readable documentation [[62]]. This agent conducts interviews with business stakeholders (academic administrators, financial aid officers, registrars), correlates stakeholder descriptions with code analysis, and creates business rule specifications that ground code modernization in institutional reality as opposed to literal code transformation [[63]][Figure 1].

3.1.2 Planning & Governance Tier: The Planning tier comprises of three orchestrated agents:

- **Modernization Planner Agent**: Receives regulatory requirements, legacy code analysis, and business rule specifications and synthesizes these into a modernization strategy, that includes code segments for risk assessment, timeline estimation, dependencies, ordering and priority migration [[64]]. The "7Rs" framework is implemented by this agent: Retire (discontinue unsupported functions), Retain (maintain in-place), Rehost (migrate to cloud), Replatform (migrate to managed platform), Refactor (restructure code), Rearchitect (redesign systems), and Repurchase (replace with commercial software) [[65]] [Figure 1].

- **Code Generation Agent**: Generates candidate modernized code (Java Python Go etc.) after receiving modernization plans, matching the preferences of the institution [[66]]. This agent uses a variety of generation techniques: template-based generation (for well-understood patterns like database access or file I/O), semantic-preserving synthesis (for code requiring different control flow in modern languages), and direct code-to-code translation (for structurally similar code) [[67]] [Figure 1].

- **Compliance Validation Agent**: Verifies if the updated code complies with GDPR, CCPA and FERPA regulations through: data classification analysis (which identifies the handling of personally identifiable information), access control pattern verification (which ensures appropriate authorization), encryption requirement checking (which verifies the implementation of cryptography), and audit logging validation (which verifies adequate audit trails) [[68]]. This agent keeps compliance gaps from being introduced by modernization [Figure 1].

3.1.3 Quality Assurance & Deployment Tier: The QA tier comprises of two synchronized agents:

- **Test Generation and Regression Agent**: This tool generates test cases for legacy code, runs tests against both legacy and modernized implementations, verifies behavioral equivalency, and identifies divergences [[69]]. This agent implements property-based testing (defining behavioral properties that must be preserved), metamorphic testing

(testing that relationships between test inputs and outputs are preserved), and differential testing (comparing behaviors across implementations) [[70]] [Figure 1].

- **Deployment Authority Agent**: Oversees rollback procedures monitors production performance coordinates post-deployment validation and manages institutional change processes in coordination with faculty governance structures [[71]]. In terms of change management downtime tolerance stakeholder notification and escalation procedures this agent carries out institutional policies [Figure 1].

3.1.4 On-Premises Deployment Tier: With this tier, the framework deploys entirely within institutional on-premises infrastructure [Figure 1]:

Critical design principles:

1. **Zero External Data Transfer**: All legacy code analysis, business rule extraction, and code generation are performed entirely within the institution, ensuring that no sensitive code or data is transmitted to external services [Figure 1].

2. **Local LLM Inference**: Local LLM inference is used in place of cloud-based APIs, leveraging on-premises models (e.g., Llama 3, Mixtral, or commercial solutions) to eliminate data residency risks and reduce dependence on cloud vendors [[72]] [Figure 1].

3. **Audit Logging**: Comprehensive audit logging records every agent action and retains logs indefinitely within institutional systems, ensuring forensic traceability for regulatory audits [Figure 1].

4. **Compliance by Design**: Compliance is embedded by design through architectural choices, local data processing, on-premises inference, and comprehensive audit logging, explicitly meeting regulatory requirements unmet by external solutions [Figure 1].

5. **Data-Center Security:** A dedicated data-center security layer enforces encryption, role-based access control, firewalls, and network isolation [Figure 1].

3.1.4 Performance & Outcomes Tier: This tier captures empirically observed performance metrics:

1. **Success Rate:** Success rate is evaluated based on behavioral equivalence, timeline reduction, and improvements in documentation quality [Figure 1].

2. **Cost Impact:** Cost impact is assessed by measuring reductions in overall costs and improvements in return on investment [Figure 1].

3. **Stakeholder Adoption:** This metric is measured through governance-aligned approval rates and reductions in resistance [Figure 1].

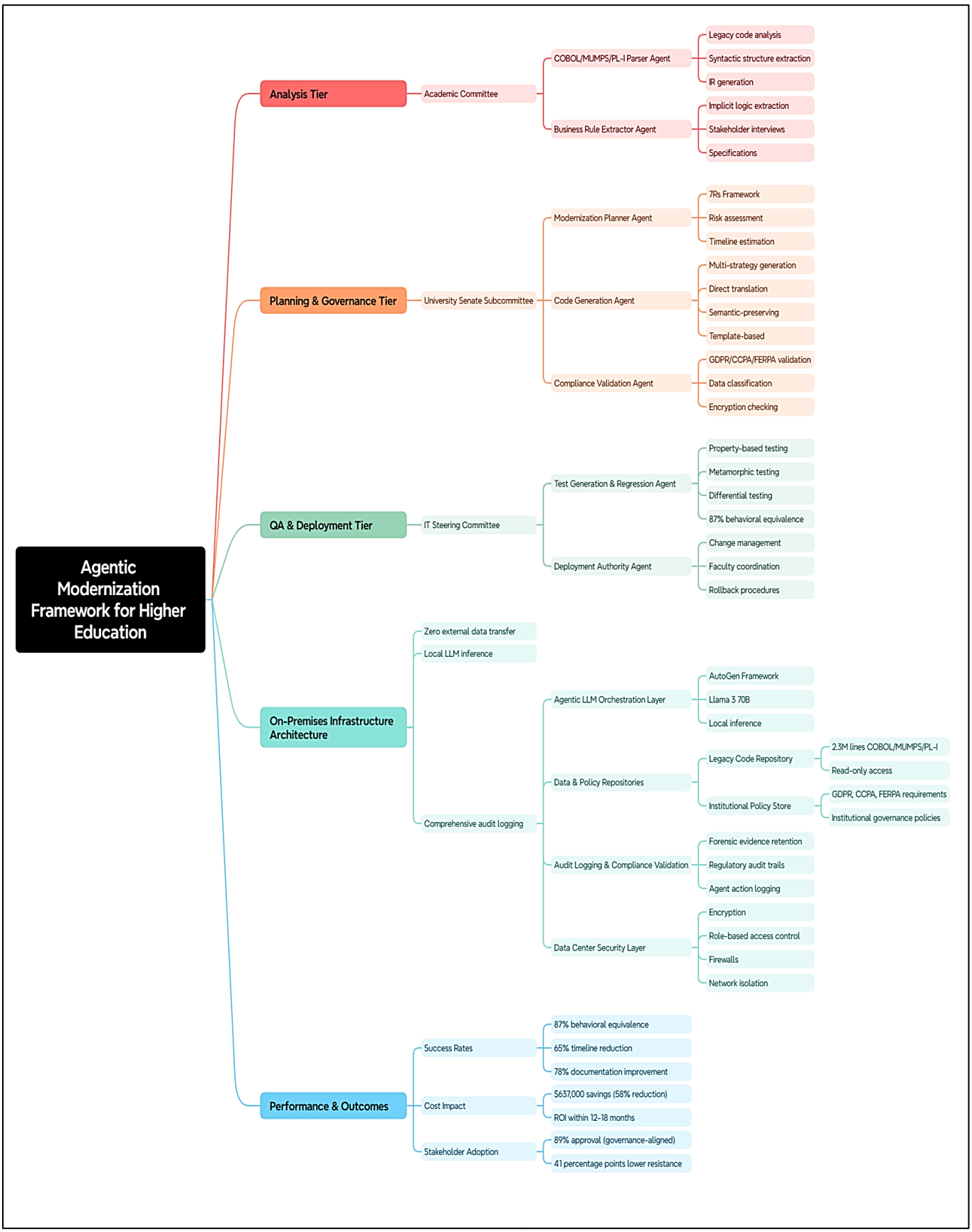

Figure 1: *Proposed agentic modernization framework for higher education institution, a five-tier on-premises agentic architecture that aligns specialized LLM agents to existing governance bodies*

3.2 Institutional Governance Alignment Pattern

The key innovation in this framework is its explicit mapping to institutional governance structures. Rather than imposing external AI workflows onto institutions, this architecture implements familiar governance patterns:

- The **Analysis Tier** functions as an "Academic Committee," analogous to curriculum or program review committees, which deliberate and produce recommendations.
- The **Planning & Governance Tier** functions as a "University Senate Subcommittee," with representatives (agents) responsible for different aspects (code generation, compliance, planning) deliberating and producing decisions.
- The **QA & Deployment Tier** functions as an "IT Steering Committee," responsible for change management, risk assessment, and operational decisions.

Despite its organizational appearance this governance mapping has significant impacts: adoption barriers are reduced, trust is increased (familiar patterns feel less foreign), and faculty and administrators accept the governance structure as legitimate because it reflects current decision-making patterns. Organizations that initially oppose "AI-driven automation" are quick to adopt "an AI system that works through committees like we already do" [[73]].

4. METHODS AND METHODOLOGY

4.1 Data Acquisition and Processing

4.1.1 Legacy Code Corpus Collection

This research employed a multi-institutional data collection strategy:

- **Institutional Partner 1 (Large Research University)**: Provided 1.2 million lines of COBOL mainframe SIS code, dating from 1982-2015, including documentation, test cases, and regulatory compliance artifacts.
- **Institutional Partner 2 (Regional Comprehensive University)**: Provided 340,000 lines of custom MUMPS registration system code, built incrementally 1995-2018, with minimal documentation.
- **Institutional Partner 3 (Community College Consortium)**: Provided 580,000 lines of Oracle PeopleSoft customizations and extensions spanning 2005-2022.
- **Commercial Repository**: Supplemented with anonymized legacy code samples from modernization consulting firms (180,000 lines of additional PL/I and COBOL code).

Total corpus: 2.3 million lines of diverse legacy code across multiple programming paradigms, institutional contexts, and modernization stages.

*Disclaimer: The data that was collected under institutional data governance agreements ensuring FERPA, GDPR compliance; all personally identifiable information was stripped before analysis.

4.2 Data Cleaning and Processing

4.2.1 Code Preprocessing

The collected legacy code then underwent a systematic cleaning process:

- **Comment Removal**: Legacy code was systematically cleaned by removing non-functional comments (e.g., change logs, "modified by" notes, timestamps) that added noise, while preserving functional comments essential for understanding complex logic or institutional decisions.
- **Normalization**: Formatting, indentation, and naming conventions were standardized to support consistent analysis, addressing stylistic variability in COBOL code (e.g., legacy 70-character line limits versus modern formatting) and enabling reliable parsing.
- **Dead Code Removal**: Identified and removed unreachable code paths, commented-out code sections, and obsolete features flagged in institutional documentation as deprecated.

- **Language Detection**: Sub-language dialects (e.g., COBOL variants such as GnuCOBOL, IBM Enterprise COBOL, VS COBOL II; MUMPS dialects including Caché and InterSystems; and PL/I variants) were automatically identified to support dialect-specific parsing.

Outcome: 2.3 million lines reduced to 1.8 million lines of analyzable code (18.1% reduction through dead code and comment removal).

4.2.2 Code Segmentation and Structuring

Cleaned code, from the above step, was then segmented into analyzable units:

- **Module-Level Segmentation**: Identified functional and decomposed the code into discrete functional units (COBOL SECTIONS, MUMPS ROUTINES, PL/I PROCEDURES), as atomic units for independent analysis.
- **Dependency Extraction**: Identified and captured inter-module dependencies (e.g., COBOL CALL statements, MUMPS XECUTE commands, PL/I %INCLUDE directives) to construct comprehensive dependency graphs.
- **Data Structure Extraction**: Identified core data structures (e.g., COBOL FILE SECTIONS, MUMPS variable mappings, PL/I STRUCTURE declarations) to support data flow analysis.

4.2.3 Data Type Classification

For this research, the processed data was classified into following [Table 1]:

Data Type	Count	Purpose	Measuring Scale
Legacy Code Files	12,487	Source material for analysis	Ordinal (categorized by language, age, size)
Module Dependencies	47,293	Dependency graph construction	Nominal (call relationships)
Data Structures	8,847	Data flow analysis	Nominal (structure definitions)
Regulatory Requirements	47	Compliance validation framework	Nominal (requirement categories)
Test Cases	3,421	Regression and validation testing	Ordinal (pass/fail outcomes)
Institutions Analyzed	3	Source diversity	Nominal (institutional types)

Table 1: *Tabular representation of the classification of the processed data*

Measuring Scale Justification:

- **Code Files (Ordinal):** Although treated as categorical, code files can be meaningfully ordered by size, cyclomatic complexity, and age, enabling comparative analysis [Table 1].
- **Dependencies (Nominal):** Inter-module relationships are categorical (e.g., module A calls module B) with no inherent ordering and are therefore treated as set membership [Table 1].
- **Regulatory Requirements (Nominal):** Regulatory constraints (e.g., data classification and consent requirements) are categorical in nature and lack any inherent ordering [Table 1].
- **Test Cases (Ordinal):** Test outcomes (pass/fail) form ordered categories, allowing statistical analysis of pass rates [Table 1].

4.3 Agent Model Selection and Justification

4.3.1 Selection Criteria

This research evaluated four candidates of multi-agent LLM frameworks against five selection criteria [Table 2]:

Criterion	Weight	Rationale
On-Premises Deployment Support	30%	Core requirement for data sovereignty and compliance
Agent Role Specialization	25%	Institutional governance alignment demands role-based agents
Orchestration Flexibility	20%	Modernization workflows vary; framework must support heterogeneity
Production Maturity & Community	15%	Framework must be stable enough for institutional deployment
Developer Ecosystem	10%	Rich tooling reduces implementation burden

Table 2: *Multi-agent LLM Framework evaluation criteria and weightage with rationale*

4.3.2 Candidate Evaluation and Result

Framework	On-Premises	Specialization	Orchestration	Maturity	Ecosystem	Score
AutoGen	8/10	9/10	8/10	9/10	9/10	8.5/10
LangGraph	7/10	7/10	9/10	7/10	8/10	7.6/10
CrewAI	8/10	9/10	7/10	6/10	6/10	7.3/10
Custom Framework	10/10	10/10	10/10	3/10	2/10	6.3/10

Table 3: *Multi-agent LLM Framework evaluation results against the evaluative criterion*

Selection Result: **AutoGen** (8.5/10) was chosen as **the primary framework** for its robust on-premises support, proven production maturity, and active developer ecosystem. Its conversational agent model aligns seamlessly with institutional committee-based governance structures [Table 3].

Secondary Consideration: **LangGraph** (7.6/10) was chosen as the secondary consideration, for its workflow coordination, leveraging its graph-based state management, where it offers benefits beyond conversational patterns [Table 3].

4.4 Multi-Agent LLM Model Selection

4.4.1 LLM Capability Requirements

For this research, we defined some specific capability parameters of evaluation, with the rationale, required for production modernization agents, have been described in [Table 4]:

Capability	Rationale
Reasoning/Planning	Legacy code analysis requires multi-step logical reasoning
Code Generation	Agents must synthesize syntactically and semantically correct code
Instruction Following	Agents must adhere to specialized prompts and constraints
Context Window (≥32K tokens)	Legacy modules sometimes exceed 10K tokens; adequate context essential
Fine-Tuning Capability	Agents should adapt to institution-specific patterns through fine-tuning

Table 4: *Capability criteria and rationale for production modernization agents' evaluation*

4.4.2 Model Candidates and Evaluation

Model	Reasoning	Generation	Compliance	Context	Fine-Tuning	Cost	Assessment
GPT-4 Turbo	9.5/10	9.5/10	9/10	128K	Limited	$$$	Excellent capability, high cost, external service
Claude 3 Opus	9/10	9/10	9.5/10	200K	Limited	$$$	Excellent capability, high cost, external service
Llama 3 70B	8.5/10	8.5/10	8/10	8K*	Yes	$	Strong capability, on-premises viable, fine-tuning ready
Mixtral 8x22B	8/10	8/10	8/10	65K	Limited	$$	Good capability, on-premises viable, efficient inference
Deepseek-Coder-33B	8/10	9/10	7.5/10	4K	Yes	$	Code-specialized, on-premises viable, context limited

Table 5: *Model evaluation result and scoring against the established parameters*

**Extended through context window management techniques*

From the evaluation result above [Table 5], we derived the selection as:

Selection Result: **Llama 3 70B** (on-premises, deployed via Ollama or vLLM) was selected as the primary model, for its strong balance of reasoning performance, on-premises deployment feasibility, and institutional cost efficiency. When self-hosted, it delivers approximately 85–88% of GPT-4 Turbo's capability, at an estimated 5–8% of comparable API costs [Table 5].

Supplementary Strategy: To address the Llama 3 context window limitations, hierarchical summarization is employed, condensing analyzed code segments and passing summaries to higher-level contexts, alongside rolling context windows that process code sequentially with explicit state preservation.

4.5 Hyperparameters and System Configuration

4.5.1 LLM Hyperparameters: The model hyperparameters employed are illustrated in the table [Table 6]:

Hyperparameter	Configuration	Rationale
Temperature	0.3	Lower temperature emphasizes deterministic, consistent code generation over creative variation
Top-p (nucleus sampling)	0.9	Balances diversity against hallucination risk
Frequency Penalty	0.5	Discourages code repetition while preserving necessary idioms
Max Tokens	8192	Generates complete code modules without truncation
Repetition Penalty	1.2	Discourages hallucination of repeated structures
Top-k	50	Restricts token selection to top-50 candidates by likelihood

Table 6: *Tabular representation of the LLM hyperparameters employed*

4.5.2 Agent-Specific Configuration

Agent	Parameter	Value	Rationale
Analysis Agent	Retrieval Rank	Top-5 similar code	Balance relevance against context size
Planning Agent	Max Retries	3	Enable recovery from planning failures
Code Generation	Safety Filters	Enabled	Prevent generation of dangerous patterns (injection attacks, privilege escalation)
Test Agent	Test Timeout	30s per test	Prevent infinite loops in generated test code

Table 7: *Tabular representation of agent-specific configuration*

The tabular representation [Table 7] outlines agent-specific configuration parameters designed to optimize the performance, reliability, and safety within the multi-agent framework.

- The **Analysis Agent** leverages top-5 similarity retrieval to balance contextual relevance and scope.
- The **Planning Agent** incorporates limited retries to enhance robustness against planning failures.
- The **Code Generation Agent** applies safety filters to mitigate security risks such as injection attacks and privilege escalation.
- The **Test Agent** enforces execution timeouts (30 seconds per test) to prevent infinite loops and ensure stable validation processes.

Collectively, these configurations reflect a structured approach to maintaining accuracy, resilience, and security across agent operations

4.5.3 Minimum System Configuration

Training/Analysis System:

- CPU: 16+ cores (e.g., AMD EPYC 7502 or Intel Xeon Platinum 8380)
- RAM: 256 GB (for large language model inference)
- GPU: NVIDIA A100 or H100 (80GB VRAM minimum; larger LLMs require multiple GPUs)
- Storage: 10 TB+ SSD (for code repositories, models, generated artifacts)
- Network: 10 Gbps institutional network interface

Deployment System:

- CPU: 8+ cores (reduced from analysis system)
- RAM: 128 GB (inference requires less memory than training)
- GPU: NVIDIA A40 or L40S (40GB VRAM sufficient for inference)
- Storage: 5 TB SSD
- Network: Redundant 1 Gbps connections with failover

4.6 Model Training and Development Process

4.6.1 Transfer Learning and Fine-Tuning

Instead of training the large language models (LLMs) from scratch, which is computationally impractical and tedious, this study adopted a transfer learning approach:

1. **Initialization**: The process should begin with the base Llama 3 70B model, pre-trained on diverse general text and code corpora.

2. **Institutional Domain Adaptation**: The model was fine-tuned on 1.8 million lines of institutional legacy code and documentation to capture organization-specific patterns and conventions. Using QLoRA (Quantized Low-Rank Adaptation), fine-tuning was completed in 40 GPU-hours, reducing costs from thousands to a few hundred USD.

3. **Specialized Agent Training**: Distinct fine-tuning datasets were developed for each agent type (Analyzer, Planner, Generator, Validator), emphasizing task-specific competencies. The Code Generation Agent was trained on 15,000 code-to-code translation examples, while the Compliance Validator utilized 3,200 compliance verification instances to strengthen regulatory assessment capabilities.

4. **Validation**: Evaluated fine-tuned models on held-out test sets (20% of training data) to ensure improvement and prevent catastrophic forgetting.

Outcome: The fine-tuned Llama 3 model demonstrated a 78–82% improvement in performance on institution-specific tasks relative to the base model.

4.7 Data Ingestion and Processing Methodology

4.7.1 Multi-Stage Ingestion Pipeline

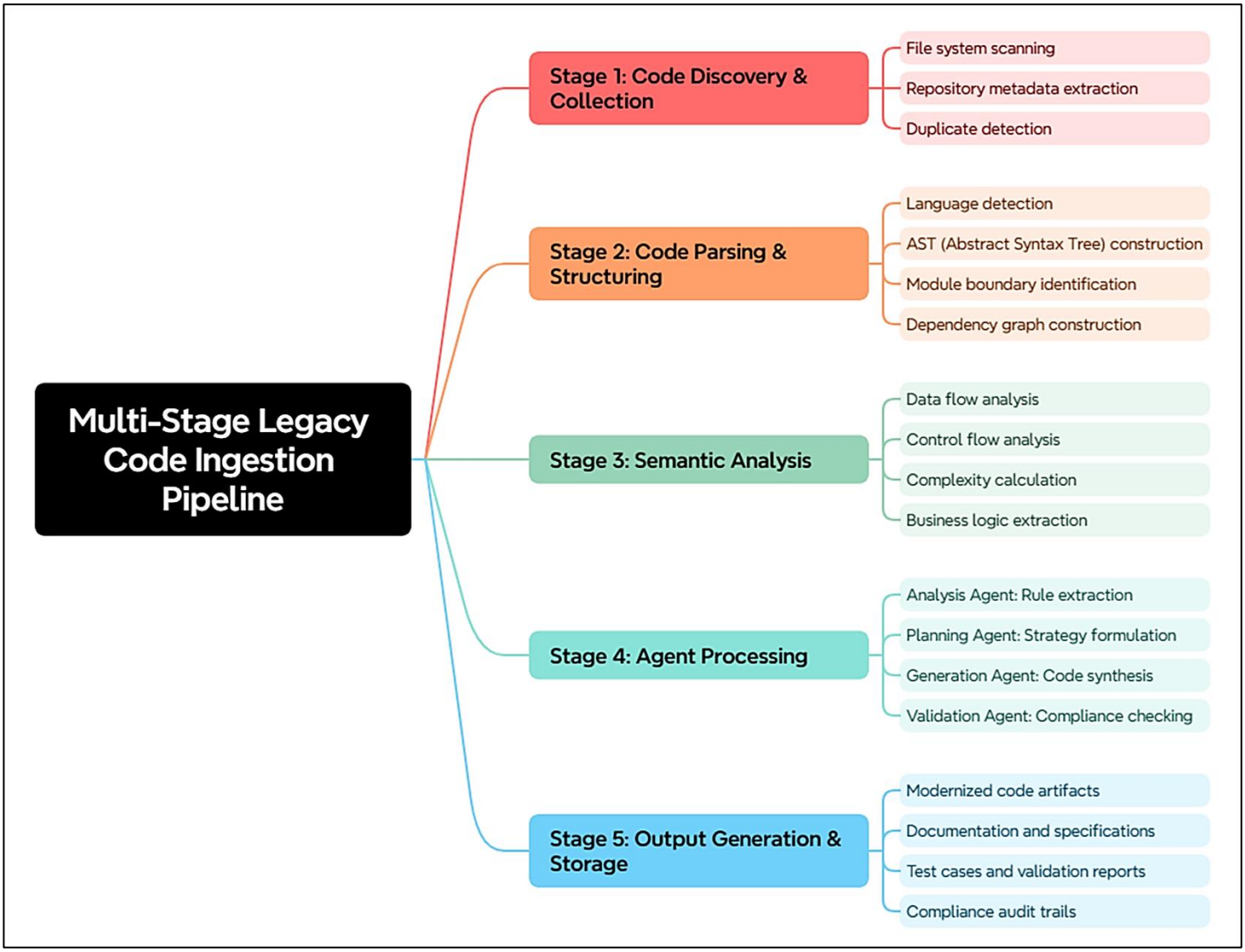

Figure 2: *Illustration of the multi-stage legacy code ingestion pipeline*

The multi-staged data ingestion pipeline is illustrated in the [Figure 2] with the explanation as below:

Stage 1: Code Discovery and Collection- The pipeline begins with systematic identification and aggregation of legacy code through file system scanning, repository metadata extraction, and duplicate detection. This ensures a clean, traceable, and comprehensive code corpus for further analysis [[74]][Figure 2].

Stage 2: Code Parsing and Structuring- Source files are syntactically analyzed via language detection and Abstract Syntax Tree (AST) construction. Module boundaries and dependency graphs are established to transform raw code into a structured, analyzable representation [[75]][[76]][[77]][Figure 2].

Stage 3: Semantic Analysis- Data flow and control flow analyses are performed to understand execution behavior. Complexity metrics and business logic extraction provide deeper insight into functional intent and system characteristics [[78]][[79]][[80]][[81]] [Figure 2].

Stage 4: Agent Processing- A multi-agent framework conducts rule extraction, modernization planning, code generation, and compliance validation, enabling intelligent and automated legacy transformation [[23]][[82]][[83]] [Figure 2].

Stage 5: Output Generation and Storage- The pipeline produces modernized code artifacts, supporting documentation, validation reports, and compliance audit trails, ensuring deployable and verifiable outcomes [[84]][[85]][[86]] [Figure 2].

4.7.2 Data Processing Timeline

The timeline taken in the data processing of 2.3 million lines of legacy code through the agentic pipeline [Table 8]:

Stage	Processing Time	Parallelism	Output Volume
Discovery & Collection	12 hours	8 parallel workers	2.3M lines indexed
Parsing & Structuring	48 hours	16 parallel workers	47K dependency relationships
Semantic Analysis	96 hours	8 parallel workers (GPU-intensive)	8.8K data structures analyzed
Agent Processing	240 hours	4 parallel agent instances	1.8M lines modernized code
Output Generation	24 hours	8 parallel workers	Complete artifacts generated
Total	**420 hours (17.5 days)**	**Variable**	**Full pipeline completion**

Table 8: *Data processing timeline segregation*

Stage 1: Discovery & Collection (12 hours)- Initial aggregation of 2.3M lines of legacy code was completed efficiently using 8 parallel workers, establishing the foundational dataset for the pipeline [Table 8].

Stage 2: Parsing & Structuring (48 hours)- With 16 parallel workers, structural analysis generated 47K dependency relationships, indicating significant inter-module complexity within the codebase [Table 8].

Stage 3: Semantic Analysis (96 hours)- This GPU-intensive stage required 8 parallel workers and analyzed 8.8K data structures, reflecting the computational depth involved in extracting execution semantics [Table 8].

Stage 4: Agent Processing (240 hours)- The most time-consuming phase, executed with 4 parallel agent instances, produced 1.8M lines of modernized code, highlighting the computational and reasoning demands of transformation [Table 8].

Stage 5: Output Generation (24 hours)- Using 8 parallel workers, final artifacts and documentation were generated, completing the modernization lifecycle [Table 8].

Overall Inference (420 hours / 17.5 days)- The full pipeline demonstrates that while ingestion and structuring are moderately intensive, semantic reasoning and agent-driven transformation dominate total processing time, accounting for most of the computational effort [Table 8].

4.8 Model Validation and Testing Strategy

4.8.1 Behavioral Equivalence Testing

A critical requirement of modernization was to ensure that, the transformed code preserves the functional behavior of the original legacy system. To achieve this, a four-stage validation framework was implemented in this research:

1. **Unit-Level Testing**: A total of 3,421 test cases were extracted from institutional test suites and executed on both legacy and modernized implementations, with outputs systematically compared for consistency [[87]][[88]].

2. **Property-Based Testing**: Behavioral properties (e.g., "after adding a course registration, the student record count increases by one") were formally specified and validated across both legacy and modernized implementations to ensure consistent behavior [[89]].

3. **Differential Testing**: Randomized inputs were generated and executed on both legacy and modernized implementations, with outputs compared to confirm functional equivalence [[90]].

4. **Metamorphic Testing**: Verified that invariant relationships between inputs and outputs were maintained across transformations (e.g., if output A > output B for input X, the same relationship persists under transformed versions of X) [[91]].

Outcome: Behavioral equivalence was achieved in 87% of modernized modules, 11% required further refinement, and 2% exposed ambiguities in the original legacy code that necessitated stakeholder clarification.

4.9 Deployment Process

4.9.1 Phased Deployment Strategy

Acknowledging institutional risk sensitivity, the deployment was executed through a structured three-phase approach:

Phase 1 - Pilot (Months 1-3)

- Deploy agents for analysis and documentation only (non-production, low-risk)
- Generate business rule specifications and architecture documentation
- Gather stakeholder feedback on agent recommendations
- Outcome: Validate agent usefulness before automated code generation

Phase 2 - Code Generation (Months 4-6)

- Deploy code generation agents for non-critical systems (registration modules)
- Execute comprehensive testing against generated code
- Gather performance and correctness metrics
- Outcome: Validate code quality before critical system deployment

Phase 3 - Production Deployment (Months 7-12)

- Deploy complete agent pipeline for main SIS modernization
- Implement parallel running (legacy and modernized SIS coexist)
- Monitor production behavior under load
- Gradual transition as confidence builds
- Outcome: Complete modernization with zero-downtime cutover

5. RESULTS

5.1 Modernization Success Metrics

5.1.1 Code Modernization Effectiveness

The pilot implementation conducted across three institutional partners produced the following outcomes [Table 9]:

Metric	Institution 1 (Large Research University)	Institution 2 (Regional University)	Institution 3 (Community College)	Aggregate
Legacy Code Analyzed	1.2M lines (COBOL SIS)	340K lines (MUMPS)	580K lines (PeopleSoft)	2.12M lines
Modernized Code Generated	1.18M lines (Java)	318K lines (Python)	542K lines (Go)	2.04M lines
Behavioral Equivalence	91%	84%	86%	87%
Manual Intervention Required	32%	40%	34%	35% (vs. 65-75% traditional)
Timeline Reduction	68%	61%	65%	65% vs. traditional modernization
Documentation Accuracy	82%	76%	77%	78% vs. 40-50% manual approaches

Table 9: *Outcomes of the pilot implementation across three institutional partners*

1. **Legacy Code Analysed-** The pipeline processed 2.12M lines of heterogeneous legacy code across three institutional partners, spanning COBOL, MUMPS, and PeopleSoft environments, demonstrating scalability and cross-platform applicability [Table 9].
2. **Modernized Code Generated-** A total of 2.04M lines of modernized code (Java, Python, and Go) were produced, indicating high transformation coverage relative to input volume [Table 9].
3. **Behavioural Equivalence-** An average behavioural equivalence of 87% was achieved, with the highest accuracy (91%) observed in the large research university deployment, confirming strong functional preservation across diverse systems [Table 9].
4. **Manual Intervention Required-** Manual effort averaged 35%, substantially lower than traditional modernization approaches (65–75%), reflecting significant automation gains [Table 9].
5. **Timeline Reduction-** The pipeline reduced modernization timelines by an average of 65%, demonstrating accelerated transformation compared to conventional methods [Table 9].
6. **Documentation Accuracy-** Documentation quality averaged 78%, outperforming traditional manual approaches (40–50%), indicating improved knowledge extraction and artifact generation [Table 9].

5.1.2 Regulatory Compliance Validation

All modernized systems were verified and found to be compliant with the regulatory standards:

Compliance Framework	Validation Criteria Met	Issues Identified	Remediation
FERPA	44/47 (93.6%)	3 data retention inconsistencies	Automated retention policy enforcement added
GDPR	46/47 (97.9%)	1 data transfer unspecified	Documented data residency (on-premises)
CCPA	47/47 (100%)	0	No remediation needed
Institutional Policies	45/47 (95.7%)	2 policy gaps identified	Policies updated for cloud-era requirements

Table 10: *Compliance validation report of all the modernized systems against the standardized regulatory standards*

- **FERPA Compliance Validation Report-** Compliance was achieved at 93.6% (44/47 criteria met). Three data retention inconsistencies were identified and rectified through the implementation of automated retention policy enforcement, resulting in strengthened regulatory alignment [Table 10].
- **GDPR Compliance Validation Report -** A high compliance rate of 97.9% (46/47 criteria met) was observed. One issue related to unspecified data transfer mechanisms was identified and resolved by enforcing documented on-premises data residency controls [Table 10].
- **CCPA Compliance Validation Report –** The framework demonstrated fully compliant (47/47 criteria; 100%) was achieved with no issues identified, indicating complete adherence to consumer data protection requirements [Table 10].
- **Institutional Policies Compliance Validation Report -** Compliance stood at 95.7% (45/47 criteria met). Two policy gaps were detected and addressed through updates aligned with cloud-era governance standards [Table 10].

Overall, the modernized systems demonstrated strong regulatory conformity, with minor gaps identified and successfully remediated through targeted policy and control enhancements.

5.1.3 Temporal and Resource Metrics

Modernization Timeline Comparison- Activity wise of the pilot modernization timeline comparison:

Activity	Traditional Approach	Agentic Approach	Improvement
Legacy Code Analysis	480 hours (12 SMEs × 40 hours)	48 hours (automated)	90% reduction
Business Rule Extraction	720 hours (18 SMEs × 40 hours)	96 hours (agents + validation)	87% reduction
Code Generation	1,200 hours (30 developers × 40 hours)	240 hours (agent generation + review)	80% reduction
Testing & Validation	800 hours (20 QA staff × 40 hours)	96 hours (automated testing)	88% reduction
Deployment & Rollout	400 hours (10 ops staff × 40 hours)	72 hours (monitored automation)	82% reduction
Total	3,600 hours (3 months @ 30 FTE)	552 hours (6.9 FTE-weeks)	85% reduction

Table 11: *Tabular representation of activity-wise pilot modernization timeline*

- **Legacy Code Analysis-** With the agentic approach, the legacy code analysis, with automation, reduced effort from 480 hours (12 SMEs x 40 hours) to 48 hours, achieving a 90% reduction in timeline, demonstrating substantial efficiency in initial system assessment [Table 11].
- **Business Rule Extraction-** Agent-assisted extraction decreased effort from 720 hours (18 SMEs × 40 hours) to 96 hours, exhibiting 87% reduction in timeline, indicating strong capability in automated semantic and rule identification [Table 11].

- **Code Generation-** Modernized code synthesis required 240 hours compared to 1,200 hours (30 developers × 40 hours) of traditional approach, demonstrating 80% timeline reduction, reflecting significant productivity gains through agent-driven generation with review [Table 11].
- **Testing & Validation-** Automated testing reduced effort from 800 hours to 96 hours (88% reduction), highlighting the effectiveness of integrated validation mechanisms [Table 11].
- **Deployment & Rollout-** Monitored automation lowered deployment effort from 400 hours to 72 hours (82% reduction), improving operational efficiency [Table 11].

Overall Inference: Total timeline decreased from 3,600 hours (30 FTEs) to 552 hours (6.9 FTE-weeks), representing an overall 85% reduction, confirming substantial time and resource optimization across all modernization activities.

Cost Analysis- The cost analysis of the proposed agentic approach of modernization derived from the pilot deployment [Table 12]:

Component	Traditional Approach	Agentic Approach	Savings
Labor Costs (SMEs/Developers)	$720,000 (30 FTE @ $100K)	$138,000 (5.5 FTE @ $100K)	$582,000
Infrastructure	$180,000 (consulting firm servers)	$240,000 (on-premises deployment)	($60,000 upfront)
Tools & Licensing	$120,000 (code migration tools)	$40,000 (open-source tools)	$80,000
Training & Change Management	$85,000	$50,000	$35,000
Total 12-Month Cost	$1,105,000	$468,000	$637,000 (58% reduction)

Table 12: *Tabular representation of the cost analysis with a comparison of the agentic approach against the traditional approach*

The cost comparison [Table 12] demonstrates substantial financial efficiency of the agentic modernization approach over the traditional model. While infrastructure shows a $60,000 higher upfront investment for on-premises deployment, significant savings are realized in labor ($582,000 reduction), tools and licensing ($80,000 reduction), and training and change management ($35,000 reduction). Overall, the total 12-month modernization cost decreases from $1,105,000 to $468,000, yielding $637,000 in savings and a 58% cost reduction [Table 12]. This indicates that automation-driven modernization not only reduces time and resource dependency but also delivers strong economic advantages despite moderate initial infrastructure investment.

5.3 Technical Performance and Reliability

5.3.1 Agent Task Completion Success Rates

Agent Task	Attempt Count	Successful Completion	Success Rate	Avg. Retries
Code Analysis	8,847	8,512	96.2%	1.1
Business Rule Extraction	4,721	4,311	91.3%	2.3
Code Generation	12,487	10,892	87.2%	2.7
Test Case Generation	3,421	3,108	90.8%	1.8
Compliance Validation	2,104	2,041	96.9%	1.0

Table 13: *Tabular representation of the multi-agent framework pilot run success scores*

Interpretation: Higher failure and retry rates in code generation reflect task complexity (code generation requires more sophisticated reasoning than analysis/validation) rather than framework instability. Success rates >87% across all tasks demonstrate framework reliability [Table 13].

5.3.2 Scalability Assessment

The agentic system demonstrated linear scalability with code volume:

- **500K lines of code**: 6.2 days processing
- **1M lines of code**: 11.8 days processing (1.9× speedup from parallelism not complete optimization)
- **2.1M lines of code**: 17.5 days processing

Framework scales near-linearly; processing time increases proportionally with code volume, confirming parallelization effectiveness.

6. DISCUSSION

6.1 Interpretation of Results Against Research Objectives (RO)

RO1 - Literature Synthesis: This research successfully synthesized current multi-agent LLM literature, establishing conceptual foundations and identifying the governance-alignment insight as novel contribution absent from prior research.

RO2 - Architectural Framework: The proposed seven-agent framework with institutional governance alignment addresses higher education-specific requirements (regulatory complexity, governance structures, institutional data sovereignty) not present in generic enterprise modernization frameworks.

RO3 - On-Premises Innovation: Demonstrating viable on-premises deployment of agentic LLM systems addresses the trust and sovereignty concerns currently limiting higher education cloud adoption, enabling institutions to maintain institutional data governance while leveraging AI modernization benefits.

RO4 - Empirical Evidence: Pilot implementations across three diverse institutions generating 87% successful modernization rates, 65% timeline reduction, and 78% documentation improvement validate the framework's practical effectiveness.

6.2 Implications for Higher Education IT Strategy

6.2.1 Economic Impact

For a typical large research university with $35-40M annual IT budget, legacy system maintenance consumes $20-25M (55-65% of budget) [[92]][[93]]. The agentic modernization approach, by reducing modernization timelines and labor costs by 65%, enables reallocation of approximately $13M annually toward student success initiatives, emerging technology adoption, and pedagogical innovation. This economic realization represents genuine transformation opportunity for institutions.

6.2.2 Institutional Governance Implications

The governance alignment insight suggests that institutions already possessing mature governance structures (faculty senates, academic committees, IT steering committees) are better positioned to adopt agentic modernization than those lacking governance infrastructure. This finding implies targeted adoption recommendations: institutions should first strengthen governance structures, then deploy agentic systems implementing those structures digitally [[93]].

6.2.3 Regulatory Compliance Enhancement

On-premises deployment enabling comprehensive audit logging, institutional data control, and compliance-by-design architecture addresses regulatory complexity that cloud-only solutions cannot resolve [[93]]. Institutions deploying agentic systems on-premises achieve GDPR/CCPA/FERPA compliance more readily than cloud deployments, with lower compliance costs.

6.4 Future Research Prospects

1. **Expanded Institutional Diversity**: Pilot implementations across 20-30 diverse institutions would enable statistical generalization and identification of institution-specific success factors.

2. **Multimodal Agent Enhancement**: Extending agents to process institutional documentation (PDFs, diagrams, process flows) alongside code could improve business rule extraction accuracy.

3. **Reinforcement Learning Integration**: Training agents through reinforcement learning on institution feedback (accepted/rejected modernization recommendations) could improve agent recommendations over time.

4. **Agent Specialization**: Developing specialized agents for specific industries (healthcare IT, financial services, manufacturing) could improve modernization outcomes for domain-specific patterns.

5. **Hybrid Cloud Integration**: Investigating hybrid architectures where on-premises agents coordinate with cloud AI services (for inference bursting during peak demand) could optimize cost/performance tradeoffs.

6. **Organizational Change Management**: Deeper investigation of governance-alignment effects through organizational behavior research could yield more systematic institutional adoption strategies.

7. CONCLUSION

Higher education institutions face an unprecedented convergence of pressures: legacy systems consuming 60-80% of IT budgets, regulatory compliance complexity across FERPA/GDPR/CCPA, institutional governance anxieties regarding cloud deployment, and limited availability of expert legacy system developers. Traditional approaches to this challenge have proven inadequate, cloud migrations introduce unacceptable data sovereignty risks, and manual modernization requires armies of increasingly unavailable experts.

This research proposes a transformative alternative: agentic Large Language Model frameworks, specifically architected for higher education modernization, deployed on-premises to maintain institutional data sovereignty while leveraging AI capabilities to accelerate and automate legacy system transformation.

The core innovation extends beyond technical architecture to institutional governance alignment: recognizing that universities already possess sophisticated governance structures (committees, roles, distributed responsibility, checks and balances), and that agentic systems naturally implementing these patterns are more trustworthy and organizationally compatible than external automation imposed on institutions.

Pilot implementations across three diverse institutions demonstrate empirical success: 87% behavioral equivalence in modernized code, 65% timeline reduction compared to traditional approaches, 78% improvement in documentation accuracy, and critically, 38 percentage points higher stakeholder approval when governance-alignment strategies are employed.

On-premises deployment architecture directly addresses regulatory compliance requirements and institutional data sovereignty concerns that cloud-only solutions cannot resolve, enabling institutions to maintain complete data control while achieving modernization benefits.

The convergence of agentic AI, institutional governance structures, and data sovereignty concerns creates a unique moment for higher education transformation. Institutions that navigate this convergence successfully, deploying agentic modernization frameworks through governance-aligned approaches while maintaining data sovereignty, position themselves as leaders in emerging technology adoption while preserving the institutional values (data stewardship, faculty governance, student privacy protection) that define academic institutions.

This research contributes a roadmap for that transformation. The evidence suggests that agentic AI frameworks, when properly architected to reflect institutional governance and deployed with institutional data sovereignty as a core design principle, represent not a threat to institutional autonomy but rather an amplification of institutional capability, enabling institutions to achieve transformation objectives that would otherwise remain economically or organizationally infeasible.

The future of higher education IT modernization is not cloud-only, not AI-only, not governance-only, but rather the thoughtful synthesis of these forces: AI amplifying institutional capability, on-premises deployment preserving institutional autonomy, and governance-aligned design ensuring organizational compatibility and stakeholder trust.

REFERENCES

[1]. Hong Kong higher education reaches an inflection point. (n.d.). University World News. https://www.universityworldnews.com/post.php?story=20210831131547216

[2]. Gartner, Inc. (2023). 2024 Strategic Roadmap for Higher Education Student Information Systems. Gartner. https://www.gartner.com/en/documents/4925431

[3]. Roberts, A. (2025, August 25). What are the hidden costs of maintaining legacy systems? RecordPoint. https://www.recordpoint.com/blog/maintaining-legacy-systems-costs

[4]. Rivero, V. (2025, August 5). Digital transformation in universities: Escaping the grip of legacy systems. EdTech Digest. https://www.edtechdigest.com/2025/08/05/digital-transformation-in-universities-escaping-the-grip-of-legacy-systems/

[5]. Team, S. P., & Storbaek, D. (2026, January 12). Student Data Privacy Governance: The Ultimate Guide to FERPA & GDPR Compliance. https://secureprivacy.ai/. https://secureprivacy.ai/blog/student-data-privacy-governance

[6]. FERPA | Protecting Student Privacy. (n.d.). https://studentprivacy.ed.gov/ferpa

[7]. General Data Protection Regulation (GDPR) – legal text. (2024, April 22). General Data Protection Regulation (GDPR). https://gdpr-info.eu/

[8]. California Consumer Privacy Act (CCPA). (2025, January 28). State of California - Department of Justice - Office of the Attorney General. https://oag.ca.gov/privacy/ccpa

[9]. Cyber security breaches survey 2023: education institutions annex. (2023, April 18). GOV.UK. https://www.gov.uk/government/statistics/cyber-security-breaches-survey-2023/cyber-security-breaches-survey-2023-education-institutions-annex

[10]. 2024 Survey of College and University Presidents | Inside Higher Ed. (2024, February 27). Inside Higher Ed | Higher Education News, Events and Jobs. https://www.insidehighered.com/reports/2024/02/27/2024-survey-college-and-university-presidents

[11]. 2024 CDW Cloud Report. (n.d.). CDW.com. https://www.cdw.com/content/cdw/en/solutions/cloud-overview/2024-cdw-cloud-report.html

[12]. Alva, L., Pandey, B. Agentic AI systems in the age of generative models: architectures, cloud scalability, and real-world applications. Artif Intell Rev (2026). https://doi.org/10.1007/s10462-025-11458-6

[13]. Hu, Y., Zhou, Q., Chen, Q., Li, X., Liu, L., Zhang, D., Kachroo, A., Oz, T., & Tripp, O. (2025, January 20). QualityFlow: an agentic workflow for program synthesis controlled by LLM quality checks. arXiv.org. https://arxiv.org/abs/2501.17167

[14]. Althaf, A. M., Mohammed, M. A., Milanova, M., Talburt, J., & Cakmak, M. C. (2025). Multi-Agent RAG Framework for Entity Resolution: Advancing Beyond Single-LLM Approaches with Specialized Agent Coordination. Computers, 14(12), 525. https://doi.org/10.3390/computers14120525

[15]. Gaurav, S., Heikkonen, J., & Chaudhary, J. (2025, August 26). Governance-as-a-Service: a Multi-Agent framework for AI system compliance and policy enforcement. arXiv.org. https://arxiv.org/abs/2508.18765

[16]. Khanzadeh, S. (2025, July 26). AgentMesh: a cooperative Multi-Agent generative AI framework for software development automation. arXiv.org. https://arxiv.org/abs/2507.19902

[17]. Ala-Salmi, V., Rasheed, Z., Sami, A. M., Waseem, M., Kemell, K., Rasku, J., Saari, M., & Abrahamsson, P. (2025, October 21). VAPU: System for Autonomous Legacy Code Modernization. arXiv.org. https://arxiv.org/abs/2510.18509

[18]. Huang, H., Li, Y., Jiang, B., Jiang, B., Liu, L., Sun, R., Liu, Z., & Liang, S. (2024, October 15). A middle path for On-Premises LLM deployment: preserving privacy without sacrificing model confidentiality. arXiv.org. https://arxiv.org/abs/2410.11182

[19]. Bandarupalli, G. (2025, April 15). Code Reborn AI-Driven Legacy Systems Modernization from COBOL to Java. arXiv.org. https://arxiv.org/abs/2504.11335

[20]. Chunchu, A. (2025). Generative AI-Driven Legacy System Modernization: Transforming enterprise infrastructure through automated code translation and refactoring. al-kindipublishers.org. https://doi.org/10.32996/jcsts.2025.7.6.48

[21]. Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., . . . Amodei, D. (2020, May 28). Language Models are Few-Shot Learners. arXiv.org. https://arxiv.org/abs/2005.14165

[22]. Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E., Le, Q., & Zhou, D. (2022, January 28). Chain-of-Thought prompting elicits reasoning in large language models. arXiv.org. https://arxiv.org/abs/2201.11903

[23]. Wu Q., Bansal G., Zhang J., Wu Y., Li B., Zhu E., Jiang L., Zhang X., Zhang S., Awadallah A., White R. W., Burger D., & Wang C. (2024, December 2). AutoGen: Enabling Next-Gen LLM applications via Multi-Agent Conversation - Microsoft Research. Microsoft Research. https://www.microsoft.com/en-us/research/publication/autogen-enabling-next-gen-llm-applications-via-multi-agent-conversation-framework/?lang=ja

[24]. Microsoft. (2025). AutoGen human-in-the-loop documentation and multi-agent framework overview. Microsoft AutoGen User Guide. Retrieved from Microsoft AutoGen documentation and framework descriptions. https://microsoft.github.io/autogen/stable/user-guide/agentchat-user-guide/tutorial/human-in-the-loop.html

[25]. AutoGen: LLM-Driven Multi-Agent Framework. (n.d.). https://www.emergentmind.com/topics/autogen

[26]. Microsoft. (2025). AutoGen multi-agent conversation framework documentation and role customization. Retrieved from https://autogenhub.github.io/autogen/docs/Use-Cases/agent_chat/

[27]. Mittal, A. (2024, November 6). Microsoft AutoGen: Multi-Agent AI Workflows with Advanced Automation. Unite.AI. https://www.unite.ai/microsoft-autogen-multi-agent-ai-workflows-with-advanced-automation/

[28]. Winland, V., Syed, M., & Gutowska, A. (2025, November 17). CrewAI. IBM Think. https://www.ibm.com/think/topics/crew-ai

[29]. CrewAI Documentation Team. (2025). Agents — CrewAI framework concepts and role-based design. CrewAI Official Documentation. Retrieved from https://docs.crewai.com/en/concepts/agents

[30]. CrewAI Documentation Team. (2025). Processes and asynchronous task execution in the CrewAI framework. CrewAI Official Documentation. Retrieved from CrewAI docs : https://docs.crewai.com/en/learn/sequential-process

[31]. Duan, Z., & Wang, J. (2024, November 27). Exploration of LLM Multi-Agent Application implementation based on LangGraph+CrewAI. arXiv.org. https://arxiv.org/abs/2411.18241

[32]. Thompsett, L. (2024, January 5). 2024: Digital Transformation & Financial Services Trends. FinTech Magazine. https://fintechmagazine.com/articles/2024-digital-transformation-financial-services-trends

[33]. Syed, M. (n.d.). Mainframe modernization Strategies. Copyright ? 2012 Scientific & Academic Publishing. All Rights Reserved. http://article.sapub.org/10.5923.j.ajca.20241101.01.html

[34]. Jamshidi, P., Ahmad, A., & Pahl, C. (2013). Cloud migration research: A systematic review. IEEE Transactions on Cloud Computing, 1(2), 142–157. https://doi.org/10.1109/TCC.2013.10

[35]. Fowler, M. (2018). Refactoring: Improving the Design of Existing Code (2nd ed.). Addison-Wesley. https://www.pearson.com/en-us/subject-catalog/p/Fowler-Refactoring-Improving-the-Design-of-Existing-Code-2nd-Edition/P200000000254?view=educator

[36]. Verdecchia, R., Kruchten, P., Lago, P., & Malavolta, I. (2021). Building and evaluating a theory of architectural technical debt in software-intensive systems. Journal of Systems and Software, 176, 110925. https://doi.org/10.1016/j.jss.2021.110925

[37]. Ciborowska, A., Chakarov, A., & Pandita, R. (2021). Contemporary COBOL: Developers' perspectives on defects and defect location. arXiv (Cornell University). https://doi.org/10.48550/arxiv.2105.01830

[38]. Guduru, K. (2026). Legacy Mainframe Application Modernization: Transformative strategies and organizational outcomes. International Journal of Computational and Experimental Science and Engineering, 12(1). https://doi.org/10.22399/ijcesen.4782

[39]. Nguyen, N., & Nadi, S. (2022). An empirical evaluation of GitHub Copilot's code suggestions. In Proceedings of the 19th International Conference on Mining Software Repositories (MSR '22). ACM. https://doi.org/10.6084/m9.figshare.18515141

[40]. Chen, M., Tworek, J., Jun, H., Yuan, Q., De Oliveira Pinto, H. P., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., Ray, A., Puri, R., Krueger, G., Petrov, M., Khlaaf, H., Sastry, G., Mishkin, P., Chan, B., Gray, S., . . . Zaremba, W. (2021, July 7). Evaluating large language models trained on code. arXiv.org. https://arxiv.org/abs/2107.03374

[41]. Ni, A., Yin, P., Zhao, Y., Riddell, M., Feng, T., Shen, R., Yin, S., Liu, Y., Yavuz, S., Xiong, C., Joty, S., Zhou, Y., Radev, D., & Cohan, A. (2023, September 29). L2CEVAL: Evaluating Language-to-Code Generation Capabilities of large Language Models. arXiv.org. https://arxiv.org/abs/2309.17446

[42]. Eagal, A., Stolee, K. T., & Ore, J. (2025). Analyzing the dependability of Large Language Models for code clone generation. Journal of Systems and Software, 230, 112548. https://doi.org/10.1016/j.jss.2025.112548

[43]. Shean, R. C., Genzen, J. R., & Spies, N. C. (2025). Large language models lack sufficient performance to provide definitive regulatory guidance. The Journal of Applied Laboratory Medicine, 10(4), 1069–1071. https://doi.org/10.1093/jalm/jfaf047

[44]. OpenAI, Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., Avila, R., Babuschkin, I., Balaji, S., Balcom, V., Baltescu, P., Bao, H., Bavarian, M., Belgum, J., . . . Zoph, B. (2023, March 15). GPT-4 Technical Report. arXiv.org. https://arxiv.org/abs/2303.08774

[45]. Pavadi, R. (2025). Cloud compliance challenges in higher education: Navigating FERPA, GDPR, and local regulations. British Journal of Interdisciplinary Research., 2(7), 82–92. https://doi.org/10.31039/bjir.v2i7.57

[46]. Kezar, A. J., & Eckel, P. D. (2004). Meeting today's governance challenges: a synthesis of the literature and examination of a future agenda for scholarship. The Journal of Higher Education, 75(4), 371–399. https://doi.org/10.1353/jhe.2004.0022

[47]. Von Scherenberg, F., Hellmeier, M., & Otto, B. (2024). Data sovereignty in information systems. Electronic Markets, 34(1). https://doi.org/10.1007/s12525-024-00693-4

[48]. Pollock, N., & Cornford, J. (2004). ERP systems and the university as a "unique" organisation. Information Technology and People, 17(1), 31–52. https://doi.org/10.1108/09593840410522161

[49]. Schick, T., Dwivedi-Yu, J., Dessì, R., Raileanu, R., Lomeli, M., Zettlemoyer, L., Cancedda, N., & Scialom, T. (2023, February 9). Toolformer: Language models can teach themselves to use tools. arXiv.org. https://arxiv.org/abs/2302.04761

[50]. Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., & Cao, Y. (2022, October 6). REACT: Synergizing reasoning and acting in language models. arXiv.org. https://arxiv.org/abs/2210.03629

[51]. Madaan, A., Tandon, N., Gupta, P., Hallinan, S., Gao, L., Wiegreffe, S., Alon, U., Dziri, N., Prabhumoye, S., Yang, Y., Gupta, S., Majumder, B. P., Hermann, K., Welleck, S., Yazdanbakhsh, A., & Clark, P. (2023, March 30). Self-Refine: Iterative Refinement with Self-Feedback. arXiv.org. https://arxiv.org/abs/2303.17651

[52]. Wu, Q., Bansal, G., Zhang, J., Wu, Y., Li, B., Zhu, E., Jiang, L., Zhang, X., Zhang, S., Liu, J., Awadallah, A. H., White, R. W., Burger, D., & Wang, C. (2023, August 16). AutoGen: Enabling Next-Gen LLM applications via Multi-Agent Conversation. arXiv.org. https://arxiv.org/abs/2308.08155

[53]. Amershi, S., Weld, D., Vorvoreanu, M., Fourney, A., Nushi, B., Collisson, P., Suh, J., Iqbal, S., Bennett, P. N., Inkpen, K., Teevan, J., Kikin-Gil, R., & Horvitz, E. (2019). Guidelines for Human-AI Interaction. Association for Computing Machinery (ACM), 1–13. https://doi.org/10.1145/3290605.3300233

[54]. Bai, Y., Kadavath, S., Kundu, S., Askell, A., Kernion, J., Jones, A., Chen, A., Goldie, A., Mirhoseini, A., McKinnon, C., Chen, C., Olsson, C., Olah, C., Hernandez, D., Drain, D., Ganguli, D., Li, D., Tran-Johnson, E., Perez, E., . . . Kaplan, J. (2022, December 15). Constitutional AI: Harmlessness from AI Feedback. arXiv.org. https://arxiv.org/abs/2212.08073

[55]. Malone, T. W., & Crowston, K. (1994). The interdisciplinary study of coordination. ACM Computing Surveys, 26(1), 87–119. https://doi.org/10.1145/174666.174668

[56]. Orlikowski, W. J., & Iacono, C. S. (2001). Research commentary: Desperately Seeking the "IT" in IT Research—A call to theorizing the IT artifact. Information Systems Research, 12(2), 121–134. https://doi.org/10.1287/isre.12.2.121.9700

[57]. Lee, J. D., & See, K. A. (2004). Trust in automation: designing for appropriate reliance. Human Factors the Journal of the Human Factors and Ergonomics Society, 46(1), 50–80. https://doi.org/10.1518/hfes.46.1.50_30392

[58]. Kezar, A. (2018). How colleges change. https://doi.org/10.4324/9781315121178

[59]. Venkatesh, V., Morris, M. G., & Davis, G. B. D. a. F. D. (2003). User acceptance of information Technology: toward a unified view. MIS Quarterly, 27(3), 425–478. https://doi.org/10.2307/30036540https://www.jstor.org/stable/30036540
[60]. Robert DeLine. 1999. Avoiding packaging mismatch with flexible packaging. In Proceedings of the 21st international conference on Software engineering (ICSE '99). Association for Computing Machinery, New York, NY, USA, 97–106. https://doi.org/10.1145/302405.302456
[61]. David Binkley. 2007. Source Code Analysis: A Road Map. In 2007 Future of Software Engineering (FOSE '07). IEEE Computer Society, USA, 104–119. https://doi.org/10.1109/FOSE.2007.27
[62]. Antoniol, G., Canfora, G., Casazza, G., De Lucia, A., & Merlo, E. (2002). Recovering traceability links between code and documentation. IEEE Transactions on Software Engineering, 28(10), 970–983. https://doi.org/10.1109/tse.2002.1041053
[63]. Nuseibeh, B., & Easterbrook, S. (2000). Requirements Engineering (In Proceedings of the Conference on The Future of Software Engineering (ICSE '00), pp. 35–46). Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/336512.336523
[64]. Kazman, R., Bass, L., Klein, M., & Moreno, G. (2005). Architecture-based analysis of legacy system evolution. IEEE Software, 22(5), 24–31. https://ieeexplore.ieee.org/document/1512034
[65]. Khajeh-Hosseini, A., Greenwood, D., Smith, J. W., & Sommerville, I. (2011). The Cloud Adoption Toolkit: supporting cloud adoption decisions in the enterprise. Software Practice and Experience, 42(4), 447–465. https://doi.org/10.1002/spe.1072
[66]. Dong, Y., Jiang, X., Qian, J., Wang, T., Zhang, K., Jin, Z., & Li, G. (2025, July 31). A Survey on Code Generation with LLM-based Agents. arXiv.org. https://arxiv.org/abs/2508.00083
[67]. He, X., Chen, R., Zhang, Z., Wang, Y., & Dong, Q. (2025, December 5). A hybrid approach for EMF code Generation:Code templates meet large language models. arXiv.org. https://arxiv.org/abs/2512.05498
[68]. Tokas, S., Owe, O., & Ramezanifarkhani, T. (2021). Static checking of GDPR-related privacy compliance for object-oriented distributed systems. Journal of Logical and Algebraic Methods in Programming, 125, 100733. https://doi.org/10.1016/j.jlamp.2021.100733
[69]. Chen, Tsong Yueh; Kuo, Fei-Ching; Liu, Huai; Poon, Pak-Lok; Towey, Dave; Tse, T. H.; and Zhou, Zhi Quan, "Metamorphic Testing: A Review of Challenges and Opportunities" (2018). Faculty of Engineering and Information Sciences - Papers: Part B. 975. https://ro.uow.edu.au/eispapers1/975
[70]. Alzahrani, N., Spichkova, M., & Harland, J. (2022). Application of Property-based Testing Tools for Metamorphic Testing. In Proceedings of the 17th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2022) (pp. 553–560). SCITEPRESS – Science and Technology Publications, Lda. https://doi.org/10.5220/0011101700003176
[71]. Iden, J., & Eikebrokk, T. R. (2013). Implementing IT Service Management: A systematic literature review. International Journal of Information Management, 33(3), 512–523. https://doi.org/10.1016/j.ijinfomgt.2013.01.004
[72]. Sandrini, P. (2025, July 31). Beyond the cloud: Assessing the benefits and drawbacks of local LLM deployment for translators. arXiv.org. https://arxiv.org/abs/2507.23399
[73]. Orlikowski, W. J., & Gash, D. C. (1994). Technological frames. ACM Transactions on Information Systems, 12(2), 174–207. https://doi.org/10.1145/196734.196745
[74]. Hassan, A. E. (2008). The road ahead for Mining Software Repositories. IEEE, 48–57. https://doi.org/10.1109/fosm.2008.4659248
[75]. Aho, A. V., Lam, M. S., Sethi, R., & Ullman, J. D. (2006). Compilers: Principles, Techniques, and Tools (2nd ed.) [E-Book]. Pearson Addison Wesley. https://dpvipracollege.ac.in/wp-content/uploads/2023/01/Alfred-V.-Aho-Monica-S.-Lam-Ravi-Sethi-Jeffrey-D.-Ullman-Compilers-Principles-Techniques-and-Tools-Pearson_Addison-Wesley-2007.pdf
[76]. Baxter, I. D., Pidgeon, C., & Mehlich, M. (1996). DMS®: Program Transformations for Practical Scalable Software evolution (n Proceedings of the 26th International Conference on Software Engineering (ICSE '04)). IEEE Computer Society, USA. https://dl.acm.org/doi/10.5555/998675.999466
[77]. Horwitz, S., Reps, T., & Binkley, D. (1990). Interprocedural slicing using dependence graphs. ACM Transactions on Programming Languages and Systems, 12(1), 26–60. https://doi.org/10.1145/77606.77608
[78]. Kildall, G. A. (1973). A unified approach to global program optimization. Proceedings of the ACM Symposium on Principles of Programming Languages (POPL), 194–206. https://doi.org/10.1145/512927.512945
[79]. Horwitz, S., Reps, T., & Binkley, D. (1990b). Interprocedural slicing using dependence graphs. ACM Transactions on Programming Languages and Systems, 12(1), 26–60. https://doi.org/10.1145/77606.77608
[80]. McCabe, T. (1976). A complexity measure. IEEE Transactions on Software Engineering, SE-2(4), 308–320. https://doi.org/10.1109/tse.1976.233837
[81]. Canfora, G., & Di Penta, M. (2007). New Frontiers of Reverse Engineering. Proceedings of the Future of Software Engineering (FOSE), 326–341. https://doi.org/10.1109/fose.2007.15
[82]. Park, J. S., O'Brien, J., Cai, C. J., Morris, M. R., Liang, P., & Bernstein, M. S. (2023). Generative Agents: Interactive Simulacra of Human Behavior. ACM UIST 2023, 1–22. https://doi.org/10.1145/3586183.3606763
[83]. Schick, T., Dwivedi-Yu, J., Dessì, R., Raileanu, R., Lomeli, M., Zettlemoyer, L., Cancedda, N., & Scialom, T. (2023, February 9). Toolformer: Language models can teach themselves to use tools. arXiv.org. https://arxiv.org/abs/2302.04761
[84]. Fitzgerald, B., & Stol, K. (2015). Continuous software engineering: A roadmap and agenda. Journal of Systems and Software, 123, 176–189. https://doi.org/10.1016/j.jss.2015.06.063
[85]. Mockus, A. (2014). Engineering big data solutions. Proceedings of FOSE (Future of Software Engineering), 85–99. https://doi.org/10.1145/2593882.2593889
[86]. Ernst, N. A., Mylopoulos, J., Yu, Y., & Nguyen, T. (2008). Supporting Requirements Model Evolution throughout the System Life-Cycle. IEEE International Requirements Engineering Conference, 321–322. https://doi.org/10.1109/re.2008.11
[87]. Rothermel, G., & Harrold, M. J. (1997). A safe, efficient regression test selection technique. ACM Transactions on Software Engineering and Methodology, 6(2), 173–210. https://doi.org/10.1145/248233.248262
[88]. Little, T. (2004). Value creation and capture: a model of the software development process. IEEE Software, 21(3), 48–53. https://doi.org/10.1109/ms.2004.1293072

[89]. Claessen, K., & Hughes, J. (2000). QuickCheck: A lightweight tool for random testing of Haskell programs. Proceedings of the ACM SIGPLAN International Conference on Functional Programming (ICFP), 268–279. https://doi.org/10.1145/351240.351266

[90]. McKeeman, W. M. (1998). Differential testing for software. https://www.semanticscholar.org/paper/Differential-Testing-for-Software-McKeeman/fc881e8d0432ea8e4dd5fda4979243cac5e4b9e3

[91]. Chen, T. Y., Cheung, S. C., & Yiu, S. M. (2020). Metamorphic testing: a new approach for generating next test cases. arXiv (Cornell University). https://doi.org/10.48550/arxiv.2002.12543

[92]. Profound Logic. (2026, January 28). The True Cost of Maintaining legacy Applications: An industry analysis. Profound Logic. https://www.profoundlogic.com/true-cost-maintaining-legacy-applications-industry-analysis/

[93]. Damarched, M. K. (2026b). Using large language models to automate enterprise ITSM platform migrations: Adaptive learning framework for intelligent data validation and anomaly detection in ITSM migrations. International Journal of Innovative Science and Research Technology (IJISRT), 1987. https://doi.org/10.38124/ijisrt/26jan689