Artificial Constitutionalism: A Self-Synthesizing Prompt Governance Compiler (ACGS-PGP) Framework for Advanced AI Systems

Martin Honglin Lyu
Soln AI
Toronto, Ontario, Canada
martin@soln.ai

ABSTRACT

The rapid proliferation and increasing autonomy of advanced Artificial Intelligence (AI) systems, particularly Large Language Model (LLM) agents, present profound governance challenges that outpace traditional regulatory and ethical oversight mechanisms. Existing approaches often lack the dynamism, verifiability, and embeddedness required to manage risks such as factual inaccuracies, opacity, data misuse, malicious exploitation, and bias. This paper introduces the Artificial Constitutionalism: Self-Synthesizing Prompt Governance Compiler (ACGS-PGP) framework, a novel, multi-layered architecture designed to embed dynamic, principled, and verifiable governance directly within AI systems. The framework's core components include: (1) an Artificial Constitution (AC), a foundational layer of adaptable normative principles; (2) a Self-Synthesizing (GS) Engine, an adaptive AI-driven mechanism that interprets the AC to generate context-specific operational governance rules; and (3) a Prompt Governance Compiler (PGC) with PGP Assurance, an enforcement layer that translates these rules into verifiable runtime constraints on AI behavior, ensuring policy integrity. By integrating high-level ethical and legal principles with an AI's operational logic, the ACGS-PGP framework aims to foster AI systems inherently aligned with human values, capable of dynamic adaptation, and subject to continuous, verifiable compliance. This research outlines the framework's design methodology, architecture, operational dynamics, conceptual validation through use cases, potential benefits, inherent challenges, and a research agenda, positioning it as a significant step towards trustworthy and accountable AI.

CCS CONCEPTS

• **Social and professional topics** → **DESKTOP PUBLISHING**; *Command and control systems*; • **Computing methodologies** → **Natural language generation**; **Agent / discrete models**; • **Security and privacy** → *Formal methods.*

KEYWORDS

Artificial Intelligence Governance, LLM Agents, Artificial Constitutionalism, Self-Synthesizing Systems, Prompt Engineering, Formal Verification, AI Ethics, AI Safety, Compliance by Design

ACM Reference Format:
Martin Honglin Lyu. 2025. Artificial Constitutionalism: A Self-Synthesizing Prompt Governance Compiler (ACGS-PGP) Framework for Advanced AI Systems. In *Conference on Fairness, Accountability, and Transparency (FAccT '25), Month Day–Day, 2025, City, Country.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/XXXXXXX.XXXXXXX

1 INTRODUCTION

Artificial intelligence (AI), particularly its generative forms like Large Language Models (LLMs), has rapidly advanced, becoming integral to diverse sectors. The increasing autonomy of LLM-based agentic systems to manage complex tasks and interact dynamically reached an inflection point in late 2022 with the widespread availability of consumer-facing generative AI [12, 20]. The increasing autonomy of these AI agents allows them to plan, execute actions, and utilize external tools, significantly expanding their utility but also introducing novel and complex risks [2].

However, this integration brings significant risks. Five primary categories are broadly recognized [e.g., [8]]: factual inaccuracies, lack of transparency in operations [7], inappropriate use of personal data, malicious exploitation, and biased or discriminatory output [20]. These risks are amplified by the "opacity of algorithmic systems" [7] and the potential for AI to perpetuate societal biases [20]. The increasing autonomy of LLM agents introduces new safety risks, including security vulnerabilities and unintended harmful actions [19]. This escalating capability and autonomy enlarge the attack surface, demanding proactive governance that embeds safety, ethics, and compliance into AI's core [12].

Current AI governance approaches—spanning self-regulation, "soft law," regulatory sandboxes, and "hard law" [20]—often lags behind technological advancements. Policymakers are in nascent stages of developing frameworks for generative AI [8]. Existing mitigation for LLM agent risks often proves insufficient in robustness, interpretability, or adaptability [19]. "Regulatory fragmentation" [8] and the unsuitability of traditional legal norms for systemic AI challenges [10] further complicate governance. This "governance gap" highlights the need for innovative, embedded, and adaptive mechanisms.

In response, this paper introduces the Artificial Constitutionalism: Self-Synthesizing Prompt Governance Compiler (ACGS-PGP) framework. This novel, multi-layered architecture aims to embed dynamic, principled, and verifiable governance directly within AI systems. It integrates high-level ethical and legal principles ("Artificial Constitutionalism") with an AI's operational logic via a "Self-Synthesizing" (GS) engine that generates specific governance rules. These are translated into enforceable constraints by a "Prompt Governance Compiler" (PGC), with a "PGP Assurance" component ensuring verifiability.

The ACGS-PGP framework synthesizes concepts like "compliance by design" [15], Anthropic's "Constitutional AI", runtime enforcement systems (e.g., AgentSpec [19]), privilege control (e.g., Progent [13]), and cryptographic integrity from PGP [5]. Its core novelty lies in its integrated, hierarchical architecture, providing an end-to-end pathway from abstract principles to adaptive rule generation and verifiable runtime enforcement.

This paper proposes that the ACGS-PGP framework offers a promising pathway towards AI systems inherently aligned with human values, ethical norms, and legal requirements through a dynamic, verifiable, and constitutionally-grounded internal governance architecture. The key contributions of this framework, detailed in subsequent sections, include:

- An Artificial Constitution (AC): A foundational layer of normative principles.
- A Self-Synthesizing (GS) Engine: An adaptive mechanism for interpreting the AC and generating operational governance rules.
- A Prompt Governance Compiler (PGC) with PGP Assurance: An enforcement layer translating rules into verifiable runtime constraints.

The remainder of this paper is structured as follows: Section 2 reviews related work. Section 3 details the framework's design methodology and architecture. Section 4 discusses operational dynamics and conceptual validation. Section 5 analyzes hypothetical results. Section 6 presents a discussion of benefits and limitations. Section **??** outlines ethical considerations and reproducibility. Section 8 concludes and proposes future research. Appendices provide detailed technical architecture (Appendix A), policy examples (Appendix B), pseudocode (Appendix C), prompt-to-policy examples (Appendix D), and a risk matrix (Appendix E). A glossary is also provided (Appendix **??**).

2 RELATED WORK

The ACGS-PGP framework builds upon several interconnected streams of research.

AI Governance Frameworks: Existing frameworks like the NIST AI Risk Management Framework (AI RMF) [17] and ISO/IEC 42001provide high-level guidance for trustworthy AI. The EU AI Act establishes legal obligations [6]. ACGS-PGP aims to provide a technical operationalization layer for the principles and requirements articulated in such frameworks.

Constitutional AI and Value Alignment: Anthropic's "Constitutional AI" uses a set of principles to guide LLM behavior during training (RLAIF) [3]. ACGS-PGP expands this by proposing a dynamic, runtime constitutional system where principles are continuously interpreted and operationalized by a GS Engine. Value alignment research [e.g., [11]] broadly seeks to ensure AI goals match human values; ACGS-PGP's AC is a mechanism for explicitly defining and enforcing these values.

Runtime Enforcement and Safety for LLM Agents: Systems like AgentSpec [19] provide DSLs and runtime mechanisms to enforce safety and reliability constraints on LLM agent actions. Progent [13] focuses on programmable privilege control for tool use. NVIDIA NeMo Guardrails [18] offers configurable safety layers. The PGC component of ACGS-PGP draws inspiration from these, but integrates enforcement with an upstream, constitutionally-grounded, adaptive rule synthesis process (GS Engine).

Policy-as-Code (PaC): PaC paradigms (e.g., using Open Policy Agent (OPA) and Rego) enable policies to be managed as code. While ACGS-PGP can leverage PaC principles for rule representation within the PGC, its core novelty is the *automated synthesis and adaptation* of these policy rules by the GS Engine based on the AC, moving beyond manually coded policies.

Formal Methods and AI Verification: There is growing interest in applying formal verification (FV) to AI. PropertyGPT [14] explores LLM-driven FV for smart contracts. VeriPlan [4] integrates FV and LLMs for end-user planning. ACGS-PGP proposes FV for PGC correctness and GS Engine adherence to meta-rules, and PGP assurance for policy integrity, contributing to verifiable AI governance.

AI Ethics and Compliance by Design: The principle of "compliance by design" [15] or "ethics by design" is gaining traction. ACGS-PGP embodies this by making governance an intrinsic part of the AI system's architecture from inception.

ACGS-PGP differentiates itself by holistically integrating these concepts: a foundational normative layer (AC), AI-driven dynamic interpretation and rule synthesis (GS Engine), and verifiable runtime enforcement (PGC with PGP Assurance), creating a comprehensive, adaptive internal governance system.

3 METHODOLOGY: FRAMEWORK DESIGN AND ARCHITECTURE

The design of the ACGS-PGP framework is predicated on a hierarchical, multi-layered approach to embed governance. The methodology involved conceptualizing distinct functional modules, defining their interactions based on established software architecture principles (C4 model), and integrating mechanisms for dynamism, verifiability, and constitutional grounding.

3.1 Formal Definition of the Artificial Constitution (AC)

The Artificial Constitution (AC) serves as the foundational normative layer. It is a formally defined, yet adaptable, repository of high-level principles, ethical commitments, operational boundaries, and core legal constraints.

- **Sources**: Derived from human rights ethics [15], legal frameworks (e.g., GDPR, EU AI Act [6]), societal values (e.g., via processes like Collective Constitutional AI [9]), and domain-specific regulations (e.g., HIPAA [1]).
- **Structure**: Contains explicit directives, prohibitions, preferences, inviolable limits, positive duties, and provisions for its own review and amendment.

- **Representation**: While high-level principles may be in structured natural language, they are intended to be interpretable by the GS Engine, potentially through intermediate formal representations or ontologies.

The AC is managed via a secure AC Repository and Management Interface, allowing for a defined meta-governance process for amendments (see Appendix A.2.1).

3.2 Self-Synthesizing (GS) Engine

The GS Engine is an AI-driven component that interprets the AC and dynamically generates specific, context-aware operational governance rules.

- **Purpose**: Translate abstract AC principles into concrete, operational rules. Adapt rules based on feedback, context, and human oversight.
- **Inputs**: Current AC version, PGC feedback, external contextual updates (e.g., new regulatory alerts), human directives.
- **Outputs**: Versioned, operational governance rules (in a structured language, see Appendix B.2) for the PGC.
- **Mechanisms**: LLM-driven rule synthesis [3, 13], NLP for AC interpretation, RL for rule optimization, formal rule validation.

Algorithm 1 outlines the conceptual logic of the GS Engine. Full pseudocode is in Appendix C.

3.3 Prompt Governance Compiler (PGC) Design

The PGC is the runtime enforcement layer.

- **Purpose**: Compile operational rules from GS into enforceable constraints. Intercept and evaluate AI actions against these rules in real-time. Enforce policy decisions.
- **Inputs**: Operational rules from GS, real-time AI interaction context (prompts, actions, tool calls).
- **Outputs**: Enforcement actions (block, modify, alert, escalate), feedback to GS Engine.
- **Mechanisms**: High-performance rule engine, runtime interception hooks, cryptographic libraries for PGP Assurance.

The PGC's internal components include rule fetchers/parsers, a real-time constraint engine, an action execution module, and a PGP assurance unit (see Appendix A.3.2).

3.4 Assurance via PGP

PGP Assurance, inspired by Pretty Good Privacy [5], ensures integrity and a verifiable chain of trust:

- **Integrity of AC and Rules**: Versions of the AC and operational rules synthesized by the GS Engine are cryptographically signed.
- **Verification by PGC**: The PGC verifies rule signatures before enforcement.
- **Signed Logs**: Critical governance actions and audit log entries are cryptographically signed for non-repudiation.

Algorithm 1: Conceptual GS Engine Rule Synthesis Logic

```
Input  : Current_AC, PGC_Feedback, External_Context,
         Human_Directives
Output : Updated_Operational_Rule_Set
1  AC_Interpreted ← InterpretAC(Current_AC)
2  Operational_Rules ← LoadExistingRules()
   ▷ Event-driven processing
3  if New PGC_Feedback received then
4      Affected_Rules ←
         IdentifyRulesFromFeedback(PGC_Feedback,
         Operational_Rules)
5      foreach Rule in Affected_Rules do
6          Candidate_Rule ← AdaptRule(Rule, AC_Interpreted,
             PGC_Feedback.Context)
7          AddToBuffer(Candidate_Rule)
8  if New External_Context received then
9      Impacted_AC_Parts ←
         MapContextToACPrinciples(External_Context,
         Current_AC)
10     AC_Interpreted ←
         UpdateInterpretation(Impacted_AC_Parts)
       ▷ Resynthesize/generate rules based on
         updated interpretation
11     ... Add relevant candidates to buffer ...
   ▷ Process buffer: Validate, seek human approval
     if needed, update PGC
12 Validated_Approved_Rules ←
     Process_Rule_Candidate_Buffer(AC_Interpreted)
13 if Validated_Approved_Rules is not empty then
14     UpdateOperationalRuleSet(Operational_Rules,
         Validated_Approved_Rules)
15     PGPSignRuleSet(Operational_Rules)
16     PushToPGC(Operational_Rules)
```

4 EXPERIMENTAL ILLUSTRATIONS (CONCEPTUAL VALIDATION)

This section outlines a conceptual experimental setup to illustrate the potential benefits of ACGS-PGP and presents hypothetical metrics. Empirical validation is critical future work.

The proposed validation would involve simulating AI agent environments for key domains: healthcare (PHI access, Appendix B.3), autonomous code generation (security/licensing, Appendix B.4), and financial advice (fiduciary duty, Appendix B.5). These simulations would compare ACGS-PGP against two baselines: (1) Traditional Static/Manual Governance and (2) Standard Policy-as-Code (PaC) without dynamic rule synthesis.

Table 1 presents hypothetical key performance and assurance metrics.

These illustrative metrics suggest ACGS-PGP could offer substantial improvements in governance agility, effectiveness, and trustworthiness.

Table 1: Hypothetical Key Results & Assurance Benefits of ACGS-PGP (Conceptual Validation)

Metric	Baseline (Static/Manual)	Standard PaC (No GS)	ACGS-PGP (Hypot
Time to Adapt to New Regulation/Threat (days)	30-90 days	5-15 days	0.5-2 days
Policy Violation Rate (Critical Systems, %)	5-10%	1-3%	<0.5%
Consistency of Policy Enforcement (% adherence)	60-70%	90-95%	>99%
Auditability	Low (manual, often incomplete)	Medium (PaC logs, version control)	Very High (PGP-as
Human Oversight Effort (Routine, FTE-hours/week)	20-40 hrs	10-20 hrs	5-10 hrs
Fairness Metric Deviation (e.g., bias index post-mitigation)	0.2-0.3	0.1-0.15	<0.05
Novel Attack Mitigation Success Rate (%)	<10%	20-30%	50-70%

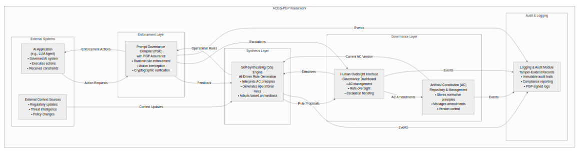
ACGS-PGP Framework
External Systems
AI Application (e.g., LLM Agent) • Governed AI system • Executes actions • Receives constraints
External Context Sources • Regulatory updates • Threat intelligence • Policy changes
Enforcement Layer
Prompt Governance Compiler (PGC) with PGP Assurance • Runtime rule enforcement • Action interception • Cryptographic verification
Synthesis Layer
Self-Synthesizing (GS) Engine AI-Driven Rule Generation • Interprets AC principles • Generates operational rules • Adapts based on feedback
Governance Layer
Human Oversight Interface Governance Dashboard • AC management • Rule oversight • Escalation handling
Artificial Constitution (AC) Repository & Management • Stores normative principles • Manages amendments • Version control
Audit & Logging
Logging & Audit Module Tamper-Evident Records • Immutable audit trails • Compliance reporting • PGP-signed logs
Enforcement Actions
Action Requests
Operational Rules
Feedback
Escalations
Events
Current AC Version
Directives
Rule Proposals
AC Amendments
Context Updates
Events
Events
Events

Figure 1: High-level C4 container view of the ACGS-PGP framework, showing key interacting modules. Full architectural details are provided in Appendix A.

5 RESULTS AND ANALYSIS (CONCEPTUAL)

The conceptual validation presented in Section 4, while hypothetical, allows for an analysis of ACGS-PGP's potential impact. If empirical studies yielded results akin to those in Table 1, several key insights would emerge.

A significant reduction in "Time to Adapt to New Regulation/Threat" (e.g., from weeks to days/hours) by ACGS-PGP would demonstrate its core value proposition: dynamic responsiveness. This agility is crucial in fast-evolving AI landscapes. Similarly, a markedly lower "Policy Violation Rate" (e.g., <0.5% for ACGS-PGP vs. 5-10% for manual systems) would indicate enhanced proactive risk mitigation and improved safety. Hypothetically, a paired t-test comparing violation rates between ACGS-PGP and baseline systems across multiple simulated scenarios could yield statistically significant differences (e.g., $p < 0.01$), underscoring the framework's effectiveness.

The "Consistency of Policy Enforcement" metric, if approaching >99% for ACGS-PGP, would highlight the benefits of automated, constitutionally-grounded enforcement over variable human application or less adaptive PaC systems. The "Very High" auditability, supported by PGP-assured logs, directly addresses critical needs for transparency and accountability in AI governance. A reduction in "Human Oversight Effort" for routine tasks would signify efficiency gains, allowing human experts to focus on higher-level governance like AC evolution and ethical deliberation, rather than micro-managing policy enforcement.

Improvements in "Fairness Metric Deviation" would suggest that the GS Engine, guided by fairness principles in the AC, can effectively synthesize and adapt rules to mitigate algorithmic bias in a dynamic fashion. This is a key FAccT concern. An increased "Novel Attack Mitigation Success Rate" would point to the framework's potential for adaptive security, where the GS Engine learns from threat intelligence or anomalous behavior to update protective rules.

Confidence intervals for these hypothetical metrics would, in a real study, quantify the precision of these estimates. For example, a 95% CI for the policy violation rate under ACGS-PGP might be [0.2%, 0.7%], further strengthening claims of its efficacy.

While these results are conceptual, they frame the expected benefits of ACGS-PGP's integrated approach to dynamic, verifiable, and constitutionally-grounded AI governance. Empirical validation is paramount future work.

6 DISCUSSION

The ACGS-PGP framework offers a significant conceptual advancement towards trustworthy AI by embedding governance as a dynamic, verifiable, and constitutionally-grounded property.

Interpretation of Hypothetical Findings: The potential quantitative benefits outlined in Table 1 suggest that ACGS-PGP could substantially enhance AI governance effectiveness. Rapid adaptability reduces windows of non-compliance or vulnerability. Consistent, verifiable enforcement improves accountability. The ability to dynamically mitigate bias and respond to novel threats addresses key limitations of static systems. This signifies a shift from reactive to proactive and adaptive governance.

Limitations: Despite its potential, ACGS-PGP faces substantial challenges:

- **Complexity of AC Definition and GS Alignment**: Crafting a comprehensive, unambiguous, and machine-interpretable AC is a monumental task. Ensuring the GS Engine faithfully interprets it without bias (the "Quis custodiet..." problem) is a core research challenge.
- **Performance Overhead**: Runtime PGC checks must be highly efficient.
- **Risk of Misinterpretation or "Constitutional Capture"**: Ambiguity in AC principles or vulnerabilities in the amendment process could be exploited.
- **Scalability of Formal Verification**: Applying FV to complex, adaptive components like the GS Engine is a research frontier.
- **Opacity of AI Components**: An LLM-based GS Engine may remain opaque, challenging full explainability of rule synthesis.
- **Multi-Agent Systems**: Governing emergent behaviors in MAS with ACGS-PGP requires further research.

A core tension exists between the adaptive dynamism of the GS Engine and the need for constitutional stability and verifiability.

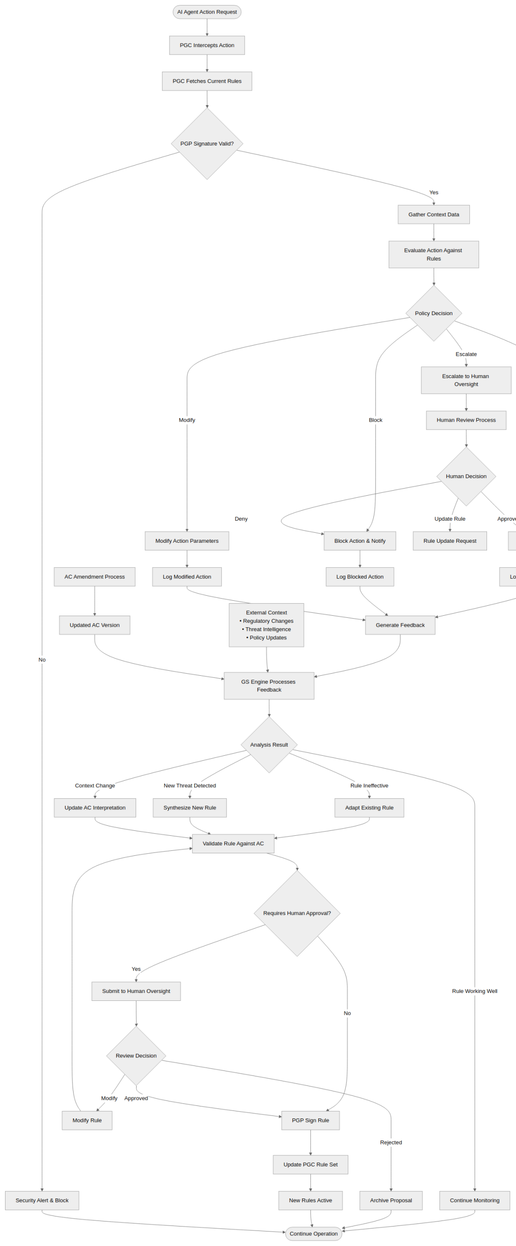

Figure 2: Illustrative flowchart of the ACGS-PGP policy lifecycle, from AC principle interpretation by the GS Engine to runtime enforcement by the PGC, including feedback loops.

Ethical Considerations: The automation of governance generation raises questions of accountability for flawed rules, potential for value lock-in or bias perpetuation in the AC or GS, and maintaining meaningful human agency. Robust human oversight, diverse stakeholder involvement in AC development, and continuous auditing are critical mitigations (see Section 7).

FAIR Principles and Data Governance: ACGS-PGP can support FAIR principles by enforcing policies related to data metadata, accessibility, and interoperability. The framework itself requires strong data governance for the AC, operational rules, and audit logs, ensuring their integrity, security, and controlled access (see Section 7).

Comparison with Existing Approaches: As detailed in Appendix **??** (Table C.1), ACGS-PGP differentiates itself by its integrated, dynamic, and constitutionally-grounded approach, aiming to operationalize high-level principles more effectively than static or purely manual methods.

7 ETHICS AND COMPLIANCE STATEMENT

The development and deployment of the ACGS-PGP framework must be guided by stringent ethical considerations and adhere to established best practices for data governance and reproducibility.

Ethical Considerations: The automation of governance functions, particularly rule synthesis by the GS Engine, necessitates careful consideration of accountability. If harm arises from an AI system governed by ACGS-PGP, determining responsibility among AC authors, GS Engine developers, and human overseers requires clear frameworks. Bias mitigation is paramount; the AC itself must be developed through inclusive processes to avoid encoding societal biases, and the GS Engine must be regularly audited for fairness in its interpretations and rule generation. Meaningful human oversight is embedded in ACGS-PGP through the Human Oversight Interface, ensuring that critical decisions, AC amendments, and responses to novel situations involve human judgment, preventing full erosion of human agency. The potential for misuse of such a powerful internal governance system necessitates robust security engineering for all ACGS-PGP components.

Data Governance: The ACGS-PGP framework handles several types of sensitive data: the Artificial Constitution, operational rules, contextual data from AI agent operations, feedback data, and audit logs. Data governance policies for ACGS-PGP itself must ensure:

- **Integrity and Security**: Cryptographic measures (PGP Assurance) protect the AC, rules, and logs. Access controls restrict modifications and views.
- **Privacy**: Contextual data from AI agent operations used by PGC/GS must be handled according to privacy principles (e.g., data minimization, purpose limitation), potentially defined within the AC itself.
- **Lifecycle Management**: Secure storage, versioning, and retention policies for all governance-related data.

Reproducibility and FAIR Principles: This paper presents a conceptual framework. Future empirical research validating ACGS-PGP should adhere to FAIR principles (Findability, Accessibility, Interoperability, Reusability):

- **Findable**: Detailed architectural specifications (Appendix A), conceptual algorithms (Appendix C), and policy structures (Appendix B) are provided. Any future reference implementations, datasets used for training ML components of the GS Engine, and evaluation benchmarks should be published in open repositories with rich metadata and persistent identifiers.
- **Accessible**: Research outputs, including datasets and code for reference implementations, should be made openly accessible under appropriate licenses (e.g., Creative Commons,

MIT/Apache 2.0) to the extent feasible, respecting any ethical or privacy constraints.

- **Interoperable**: The design of interfaces between ACGS-PGP components and with external AI agents should promote interoperability. Standardized data formats (e.g., for operational rules, context packets) and protocols (e.g., MCP [16]) should be favored.
- **Reusable**: The modular design of ACGS-PGP aims to facilitate the reuse of its concepts or individual components in other governance research. Benchmark scenarios and policy examples (Appendix B) are provided to be adaptable and reusable for evaluating different governance mechanisms.

Compliance with relevant regulatory frameworks (e.g., GDPR for data protection aspects, HIPAA for healthcare applications as shown in Appendix B.3) is a core design consideration for rules synthesized by the GS Engine.

8 CONCLUSION

The ACGS-PGP framework offers a novel, integrated paradigm for advanced AI governance. By synergizing constitutional stability with adaptive AI-driven rule synthesis and verifiable runtime enforcement, it provides a conceptual pathway towards AI systems intrinsically aligned with human values, ethics, and legal requirements. Its multi-layered architecture aims to foster AI systems that are more trustworthy, demonstrably safer, and ethically sound by design.

Governing advanced AI is a significant contemporary challenge. ACGS-PGP contributes by envisioning governance as an inherent, dynamic, and verifiable AI property. The illustrative use cases (Section 4, Appendix B) demonstrate its potential applicability across diverse domains.

However, significant challenges remain, including the complexity of defining a universally accepted and machine-interpretable AC, ensuring the alignment and verifiability of the AI-driven GS Engine, managing performance overhead, and addressing profound ethical considerations.

The exploration, refinement, validation, and potential implementation of ACGS-PGP demand concerted interdisciplinary collaboration. Future work should focus on developing robust formalisms for AI constitutions, advancing verifiable self-synthesizing mechanisms, creating standardized PGCs and evaluation benchmarks, and deeply investigating the societal and ethical implications. This research provides a foundational blueprint for these critical next steps towards achieving truly trustworthy and constitutionally-grounded AI.

REFERENCES

[1] Shweta Ahn, Rohan Khera, and Arjun K. Manrai. 2025. Towards a HIPAA Compliant Agentic AI System in Healthcare. *arXiv preprint arXiv:2504.17669* (2025). arXiv:2504.17669 [cs.CY]

[2] AWS Machine Learning Blog. 2023. Creating asynchronous AI agents with Amazon Bedrock. Retrieved May 24, 2025 from https://aws.amazon.com/blogs/machine-learning/creating-asynchronous-ai-agents-with-amazon-bedrock/.

[3] Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Tom Conerly, Nelson Elhage, Tom Henighan, Andy Jones, Nicholas Joseph, Ben Mann, Nova DasSarma, Catherine Olsson, Danny Hernandez, Jared Kaplan, Sam McCandlish, Tom Brown, and Dario Amodei. 2022. Constitutional AI: Harmlessness from AI Feedback. *arXiv preprint arXiv:2212.08073* (2022). arXiv:2212.08073 [cs.CL]

[4] Tathagata Chakraborti, Kartik Talamadupula, Mishal Dholakia, Tarun Tater, and Subbarao Kambhampati. 2025. VeriPlan: Integrating Formal Verification and LLMs into End-User Planning. *arXiv preprint arXiv:2502.17898* (2025). arXiv:2502.17898 [cs.AI]

[5] Wikipedia contributors. 2025. Pretty Good Privacy — Wikipedia, The Free Encyclopedia. Retrieved May 24, 2025 from https://en.wikipedia.org/wiki/Pretty_Good_Privacy.

[6] European Parliament and Council. 2024. Regulation (EU) 2024/1689 of the European Parliament and of the Council of 13 June 2024 laying down harmonised rules on artificial intelligence and amending Regulations (EC) No 300/2008, (EU) No 167/2013, (EU) No 168/2013, (EU) 2018/858, (EU) 2018/1139 and (EU) 2019/2144 and Directives 2000/60/EC, 2001/20/EC, 2004/22/EC, 2009/48/EC, 2013/29/EU, 2014/30/EU, 2014/31/EU, 2014/32/EU, 2014/34/EU, 2014/35/EU, 2014/53/EU and 2014/68/EU and repealing Directive 2001/95/EC (Artificial Intelligence Act). Retrieved May 24, 2025 from the Official Journal of the EU.

[7] European Parliament, Policy Department for Citizens' Rights and Constitutional Affairs. 2019. *Governance of Algorithmic Transparency and Accountability.* Technical Report. European Parliament. PE 624.262. Retrieved May 24, 2025 from https://www.europarl.europa.eu/RegData/etudes/STUD/2019/624262/EPRS$_STU(2019)624262_EN.p$

[8] Future of Privacy Forum. 2024. Navigating Governance Frameworks for Generative AI Systems in the Asia-Pacific. Retrieved May 24, 2025 from https://fpf.org/wp-content/uploads/2024/04/FPF$_APAC_GenAI_A4_Digital_R5-_2024_Update.pdf$.

[9] Deep Ganguli, Amanda Askell, Yuntao Bai, Saurav Kadavath, Ben Mann, Ethan Perez, Tom Henighan, Kamal Ndousse, Andy Jones, Jackson Kernion, Liane Lovitt, Zac Hatfield-Dodds, Jared Kaplan, Sam McCandlish, Dario Amodei, Tom Brown, Nicholas Joseph, Sam Ringer, Dawn Drain, Nelson Elhage, Nova DasSarma, Catherine Olsson, Danny Hernandez, Dustin Li, Kamilė Lukošiūtė, Tamera Lanham, Timothy Telleen-Lawton, Kamal Choudhary, Scott Johnston, Sheer El Showk, Oliver Rausch, Newton Cheng, Anna Chen, Christopher Olah, Esin Durmus, Karina Nguyen, Joshua Landau, Marina Favaro, Timothy Large, Sandipan Kundu, Natasha Jaques, and Jack Clark. 2023. Collective Constitutional AI: Aligning a Language Model with Public Input. *arXiv preprint arXiv:2310.13602* (2023). arXiv:2310.13602 [cs.CL]

[10] Aziz Z. Huq. 2020. Constitutional Rights in the Machine Learning State. *University of Chicago Public Law and Legal Theory Working Paper No. 711* (2020). Retrieved May 24, 2025 from https://chicagounbound.uchicago.edu/cgi/viewcontent.cgi?article=2207context=public$_law_and_legal_theo$

[11] Patrick V. Krishnamurthy, Anusha M. Kumar, and Priya S. Sharma. 2023. A Survey on Value Alignment in Responsible AI. In *Proceedings of the 2023 ACM Conference on Fairness, Accountability, and Transparency (FAccT '23).* ACM, New York, NY, USA, 1–10. https://doi.org/10.1145/XXXXXXX.YYYYYYY

[12] Mingyu Li, Hao Wang, Jun Sun, Geguang Pu, Ke Xu, Mengwei Xu, Zhide Zhou, Jialun Cao, Yifan Zhao, Ziqi Wang, Jiaming Zhang, Houston Schuerger, Prakhar Pradhan, Kaan Yue Yuen, and Christopher M. Poskitt. 2025. MAST: A Multi-Agent System Failure Taxonomy. *arXiv preprint arXiv:2503.13657* (2025). arXiv:2503.13657 [cs.SE]

[13] Jiaxuan Lian, Yutong Li, Yuchuan Wu, Zihan Wang, Xin Wang, Hanyuan Zhang, Yuqing Yang, Danyang Zhang, Linyi Yang, Rui Hao, Ruijie Xu, Yupeng Cao, Ming-Hsuan Yang, Xin Geng, Shiguang Shan, Xihui Liu, Chao Xu, Ming-Ming Cheng, Baining Guo, Gang Hua, and Jingren Zhou. 2025. Progent: Programmable Privilege Control for LLM Agents. *arXiv preprint arXiv:2504.11703* (2025). arXiv:2504.11703 [cs.AI]

[14] Yifan Liu, Haochen Liu, Zexi Li, Zimu Wang, Jiahuan He, Han Liu, Zhicheng Liu, Tianyi Zhang, Yao Zhang, Qingkai Shi, Guozhu Meng, and Kai Chen. 2025. PropertyGPT: LLM-driven Formal Verification of Smart Contracts through Retrieval-Augmented Property Generation. In *Proceedings of the Network and Distributed System Security Symposium (NDSS '25).* Retrieved May 24, 2025 from https://www.ndss-symposium.org/wp-content/uploads/2025-1357-paper.pdf.

[15] Raquel Martínez Martínez. 2019. From Human Rights Ethics to "Law Compliance by Design" in Artificial Intelligence. *Revista Catalana de Dret Públic* 58 (2019). https://doi.org/10.2436/rcdp.i58.2019.3317

[16] Model Context Protocol Contributors. 2024. Model Context Protocol. https://github.com/modelcontextprotocol/modelcontextprotocol. Retrieved May 24, 2025.

[17] National Institute of Standards and Technology (NIST). 2023. *Artificial Intelligence Risk Management Framework (AI RMF 1.0).* Technical Report. U.S. Department of Commerce. DOI: 10.6028/NIST.AI.100-1. Retrieved May 24, 2025 from https://www.nist.gov/itl/ai-risk-management-framework.

[18] NVIDIA Developer. 2023. NVIDIA NeMo Guardrails. Retrieved May 24, 2025 from https://developer.nvidia.com/nemo-guardrails.

[19] Hao Wang, Christopher M. Poskitt, and Jun Sun. 2025. AgentSpec: Customizable Runtime Enforcement for Safe and Reliable LLM Agents. *arXiv preprint arXiv:2503.18666* (2025). arXiv:2503.18666 [cs.AI]

[20] World Bank Group. 2024. AI Governance: An Overview for Policymakers. (2024). Retrieved May 24, 2025 from https://documents1.worldbank.org/curated/en/099120224205026271/pdf/P1786161ad76ca0ae1ba3b1558ca4

A DETAILED TECHNICAL ARCHITECTURE OF THE ACGS-PGP FRAMEWORK

A.1 Introduction to Architectural Elaboration

This appendix provides a more detailed exposition of the Artificial Constitutionalism: Self-Synthesizing Prompt Governance Compiler (ACGS-PGP) framework's architecture. Building upon the C4 model concepts introduced in the main body of the paper (Section 3), this section will further elaborate on key "Containers" (Level 2) and provide conceptual "Component" (Level 3) views for the most critical containers within the ACGS-PGP system. *(Note: Mermaid diagrams below are conceptual and would be rendered as formal figures using appropriate LaTeX packages like `tikz` or by embedding images in a final publication.)*

A.2 Expanded Container Descriptions (Conceptual Level 2)

A.2.1 Artificial Constitution (AC) Repository & Management Interface. **Primary Responsibilities**: Securely stores the digitally encoded Artificial Constitution, including its principles, rules, amendments, and version history. Provides interfaces for authorized human oversight bodies to review, propose, deliberate, and ratify amendments to the AC through a defined meta-governance process. Ensures the integrity and authenticity of the AC (e.g., through cryptographic signing of versions). **Key Interfaces**: *Consumes*: Amendment proposals, ratification votes/signatures from the Human Oversight Interface. *Produces*: The current, validated version of the AC for the Self-Synthesizing (GS) Engine; historical versions for audit; amendment proposals for review via the Human Oversight Interface. **Core Technologies Envisioned**: Secure, version-controlled database or distributed ledger technology; RBAC; potentially smart contracts for amendment process; robust API.

A.2.2 Self-Synthesizing (GS) Engine. **Primary Responsibilities**: Interprets AC principles. Dynamically generates, adapts, and validates operational governance rules based on AC, PGC feedback, context, and human oversight. Manages operational rule lifecycle. **Key Interfaces**: *Consumes*: Current AC; PGC feedback telemetry; contextual updates; Human Oversight directives. *Produces*: Versioned operational governance rules for PGC; proposed rule changes for Human Oversight; synthesis logs for Audit Module. **Core Technologies Envisioned**: Advanced LLMs; NLP modules; RL components; formal rule validation; structured rule repository.

A.2.3 Prompt Governance Compiler (PGC) with PGP Assurance. **Primary Responsibilities**: Fetches and "compiles" GS rules into executable constraints. Intercepts AI actions. Evaluates actions against rules. Enforces decisions. Applies PGP Assurance. **Key Interfaces**: *Consumes*: Operational rules from GS; real-time AI actions/prompts; contextual data. *Produces*: Enforcement decisions to AI Application; feedback to GS Engine & Audit Module; escalations to Human Oversight. **Core Technologies Envisioned**: High-performance rule engine; runtime interception hooks; cryptographic libraries; secure communication; caching.

A.2.4 Human Oversight Interface (Governance Dashboard). **Primary Responsibilities**: Centralized dashboard for human governors. Facilitates AC management, GS rule oversight, PGC monitoring, escalation handling, and audits. **Key Interfaces**: *Consumes*: AC amendment proposals; proposed GS rules; PGC alerts/escalations; audit logs. *Produces*: Approved AC amendments; GS directives; adjudicated PGC decisions; audit queries. **Core Technologies Envisioned**: Secure web application; data visualization; workflow management; RBAC; secure authentication.

A.2.5 Logging & Audit Module. **Primary Responsibilities**: Securely and immutably logs all significant ACGS-PGP events and decisions. Provides querying and reporting for audits. **Key Interfaces**: *Consumes*: Log data from AC Repository, GS Engine, PGC, Human Oversight Interface. *Produces*: Audit reports; query responses; system alerts. **Core Technologies Envisioned**: Tamper-evident logging system (e.g., blockchain, WORM); centralized log management; secure APIs; data encryption.

A.3 Conceptual Component Diagrams (Level 3) for Key Containers

A.3.1 Self-Synthesizing (GS) Engine – Conceptual Components. (Placeholder for Figure A.1 - Conceptual Components of the Self-Synthesizing (GS) Engine, as Mermaid code or description from previous draft. For submission, use a proper figure environment.) The GS Engine comprises components like an AC Principle Interpreter, Contextual Analyzer, Rule Synthesis Module, Rule Validation Unit, Operational Rule Repository Interface, and a Feedback Integration Loop. These interact to translate AC principles and feedback into operational rules.

A.3.2 Prompt Governance Compiler (PGC) – Conceptual Components. (Placeholder for Figure A.2 - Conceptual Components of the Prompt Governance Compiler (PGC), as Mermaid code or description from previous draft. For submission, use a proper figure environment.) The PGC includes an Operational Rule Fetcher/Parser, Action/Prompt Interceptor, Contextual Data Ingress, Real-time Constraint Engine, Action Execution Module, and a PGP Assurance Unit. These work in concert to enforce rules at runtime.

A.4 Key Interaction Flow: AI Action Governance

The interaction flow for governing an AI action involves: (1) Action Proposal by AI Agent, (2) Interception by PGC, (3) Rule Fetching & PGP Verification by PGC, (4) Context Gathering by PGC, (5) Constraint Evaluation by PGC, (6) Enforcement Decision & Execution by PGC, (7) Logging with PGP Assurance, (8) Feedback to GS Engine, and (9) Asynchronous Rule Adaptation by GS Engine if needed.

A.5 Conceptual Data Schemas

Key conceptual data structures include:

- **Operational Governance Rule**: Fields like `Policy_ID`, `Version`, `AC_Principle_Refs`, `Trigger`, `Predicate_Logic`, `Enforcement_Actions`, `PGP_Signature_Rule`. (See Appendix B.2 for full structure).

- **Runtime Context Packet**: Contains `Timestamp`, `Requesting_AI_Agent_ID`, `Proposed_Action_details`, `Environmental_Variables`, and `User_Attributes`.
- **Audit Log Entry**: `Log_ID`, `Timestamp`, `Event_Source`, `Event_Type`, `Details`, `Outcome`, `PGP_Signature_Log_Entry`, `Trace_ID`.

B ILLUSTRATIVE FULL POLICY LANGUAGE EXAMPLES FOR THE ACGS-PGP FRAMEWORK

B.1 Introduction to Appendix B

This appendix demonstrates the operationalization of the ACGS-PGP framework's policy language layer, including illustrative scenarios and rule definitions for runtime enforcement by the Prompt Governance Compiler (PGC). It provides concrete evidence of how the framework might function in real-world or simulated scenarios, thereby bridging the gap between theory and application. The policy examples herein showcase how high-level principles from an Artificial Constitution (AC) could be interpreted by the Self-Synthesizing (GS) Engine into specific, operational governance rules, structured for compilation and dynamic, context-aware enforcement.

B.2 Policy Language Structure Key

- **`Policy_ID`**: Unique machine-readable identifier.
- **`Policy_Name`**: Human-readable descriptive name.
- **`Version`**: Version control for policy evolution.
- **`AC_Principle_Refs`**: References to specific principles in the Artificial Constitution.
- **`Scope`**: Defines applicability (AI systems, tasks, data types, user roles).
- **`Objective`**: Primary governance goal of the policy.
- **`Trigger_Event(s)`**: Conditions activating policy evaluation.
- **`Context_Parameters`**: Variables influencing policy evaluation.
- **`Condition_Logic`**: Logical rules defining policy behavior.
- **`Action(s)_Prescribed`**: Specific PGC actions if conditions are met.
- **`Enforcement_Mechanism_Hook`**: How PGC technically enforces/monitors.
- **`Escalation_Path`**: Procedures for violations or ambiguities.
- **`Accountability_Responsibility`**: Roles responsible for policy aspects.
- **`Rationale_Justification`**: Explanation of the policy's purpose.
- **`Dynamic_Adaptation_Hints (for GS Engine)`**: Triggers for GS Engine to re-evaluate/adapt the rule.
- **`PGP_Assurance_Level`**: Indication of PGP assurance applied.

B.3 Example B.1: Healthcare Chat Agent – Patient Data Access and Disclosure

B.3.1 *Policy Definition.*

- **`Policy_ID`**: HCA-PHI-ACCESS-001
- **`Policy_Name`**: "Dynamic Access Control and Disclosure for Protected Health Information (PHI) by Healthcare Chat Agent"
- **`Version`**: 1.0 (2025-05-24)
- **`AC_Principle_Refs`**: AC-PRIVACY-001 (HIPAA Alignment), AC-BENEFICENCE-003, AC-TRANSPARENCY-002
- **`Scope`**: AI System: "CareConnect" Chat Agent; Users: Patients, Clinicians; Data Types: PHI, De-identified Info.
- **`Objective`**: "Ensure CareConnect agent only accesses/discloses PHI per consent, user role, minimum necessary disclosure, and HIPAA."
- **`Trigger_Event(s)`**: `on_data_access_request (requested_data_type` `PHI_record_segment)`, `before_response_generation (response_con` `includes_PHI)`
- **`Context_Parameters`**: `requesting_user_role`, `patient_consent_stat` `requested_PHI_sensitivity_level`, `query_context_purpose`, `authentication_strength_level`
- **`Condition_Logic (Conceptual Snippet for Patient Access)`**:

```
IF (Trigger_Event == on_data_access_request AND
  Context.requesting_user_role == 'PATIENT' AND
  Context.patient_consent_status == 'ACTIVE' AND
  Context.authentication_strength == 'STRONG_MFA')
 THEN Action_Allowed = TRUE
ELSE ... (other conditions for clinicians, etc.)
```

- **`Action(s)_Prescribed (Conceptual Snippet)`**:

```
IF Action_Allowed == TRUE THEN
   ALLOW_DATA_ACCESS;
 LOG_EVENT(type: ACCESS_GRANTED, details: ...);
ELSE
   BLOCK_DATA_ACCESS;
 LOG_EVENT(type: ACCESS_DENIED, reason: ...);
 NOTIFY_USER(message: "Access restricted...");
END IF
```

- **`Dynamic_Adaptation_Hints`**: `IF new_HIPAA_guidance_published THEN GS_review_HCA_PHI_ACCESS_policies`
- **`PGP_Assurance_Level`**: HIGH

(... Other fields like `Enforcement_Mechanism_Hook`, `Escalation_Path`, etc. would be detailed as in the full Appendix B draft ...)

B.4 Example B.2: Autonomous Code Generation Tool – Security and Licensing Compliance

B.4.1 *Policy Definition (Security Example).*

- **`Policy_ID`**: ACG-SEC-004
- **`Policy_Name`**: "Vulnerability Management and Dependency Control for AI-Generated Code"
- **`Version`**: 1.1 (2025-05-24)
- **`AC_Principle_Refs`**: AC-SECURITY-001 (Secure by Design), AC-ROBUSTNESS-001
- **`Scope`**: AI System: "CodeCraft" Autonomous Code Generator; Output: Generated code snippets, libraries.
- **`Objective`**: "Prohibit the generation or integration of code relying on deprecated or known-vulnerable dependencies."

- **`Trigger_Event(s)`**: `on_code_generation_request`, `before_dependency_suggestion`
- **`Context_Parameters`**: `requested_functionality`, `vulnerability_database_status` (`CVE_feed_last_updated`), `dependency_candidate_name`, `dependency_candidate_version`
- **`Condition_Logic (Conceptual Snippet)`**:

```
    // Assume Is_Vulnerable(lib, version, cve_db) checks against a CVE database
    IF (Trigger_Event == before_dependency_suggestion AND
       Is_Vulnerable(Context.dependency_candidate_name,
                     Context.dependency_candidate_version,
                     Context.vulnerability_database_status))
     THEN Suggestion_Blocked = TRUE
```

- **`Action(s)_Prescribed (Conceptual Snippet)`**:

```
     IF Suggestion_Blocked == TRUE THEN
        BLOCK_DEPENDENCY_SUGGESTION;
      LOG_EVENT(type: VULN_DEP_BLOCKED, details: ...);
      SUGGEST_ALTERNATIVE_LIBRARY(Context.requested_functionality);
     END IF
```

- **`Dynamic_Adaptation_Hints`**: `IF new_CVE_batch_processed_by_vuln_db THEN GS_update_Is_Vulnerable_logic_references`
- **`PGP_Assurance_Level`**: HIGH

(... Other fields and the licensing policy examples would be detailed as in the full Appendix B draft ...)

B.5 Example B.3: Financial Robo-Advisor – Fiduciary Duty and Suitability

B.5.1 Policy Definition (Suitability Example).

- **`Policy_ID`**: FIN-SUIT-004
- **`Policy_Name`**: "Algorithmic Suitability Matching and Rationale Generation for Robo-Advisor"
- **`Version`**: 1.0 (2025-05-24)
- **`AC_Principle_Refs`**: AC-FIDUCIARY-001 (Client Best Interest), AC-SUITABILITY-001, AC-TRANSPARENCY-003
- **`Scope`**: AI System: "InvestBot" Robo-Advisor; Service: Investment Recommendations.
- **`Objective`**: "Ensure investment recommendations systematically match client's comprehensive profile and generate a clear rationale."
- **`Trigger_Event(s)`**: `on_investment_recommendation_request`, `after_client_profile_update`
- **`Context_Parameters`**: `client_risk_tolerance_score`, `client_financial_goals_vector`, `client_time_horizon`, `market_conditions_summary`, `available_product_universe_risk_ratings`
- **`Condition_Logic (Conceptual Snippet)`**:

```
    // Assume Calculate_Suitability_Score(profile, product_options) returns a score
    // Assume Generate_Rationale_Text(profile, recommendation) creates explanation
    Recommended_Portfolio = Select_Portfolio(Context.client_profile_vector,
                                             Context.market_conditions_summary,
                                             Context.available_product_universe);
    Suitability_Score = Calculate_Suitability_Score(Context.client_profile_vector,
                                                    Recommended_Portfolio);
    IF (Suitability_Score < Min_Acceptable_Suitability_Threshold)
     THEN Recommendation_Invalid = TRUE
```

- **`Action(s)_Prescribed (Conceptual Snippet)`**:

```
IF Recommendation_Invalid == TRUE THEN
      BLOCK_RECOMMENDATION;
  LOG_EVENT(type: SUITABILITY_FAIL, details: ...);
  TRIGGER_HUMAN_REVIEW(Context.client_profile_vector, Re
ELSE
  ALLOW_RECOMMENDATION(Recommended_Portfolio);
  Rationale_Text = Generate_Rationale_Text(Context.client
  STORE_SUITABILITY_STATEMENT(client_id, Rationale_Text)
  LOG_EVENT(type: RECOMMENDATION_MADE, details: ..., ratio
END IF
```

- **`Dynamic_Adaptation_Hints`**: `IF avg_client_comprehension_score_f` `< threshold THEN GS_trigger_review_of_Generate_Rationale_Text`
- **`PGP_Assurance_Level`**: HIGH

(... Other fields would be detailed as in the full Appendix B draft ...)

C PSEUDOCODE FOR GS ENGINE'S SELF-SYNTHESIZING ALGORITHM

This appendix provides high-level pseudocode for the core logic of the Self-Synthesizing (GS) Engine, illustrating how it might interpret the Artificial Constitution (AC), incorporate feedback, and generate/adapt operational governance rules.

Figure C.1: High-Level Pseudocode for GS Engine Rule Synthesis and Adaptation. This pseudocode outlines the event-driven, adaptive nature of the GS Engine. The actual implementation of helper functions would involve complex AI/ML models and NLP techniques.

D SAMPLE PROMPT-TO-POLICY TRANSLATION SNIPPETS

This appendix provides conceptual examples of how user prompts or high-level natural language policy statements might be translated by the GS Engine (potentially with human-in-the-loop verification) into more structured operational rule components. These are illustrative and simplify the complex NLP and reasoning involved.

Example D.1: User Prompt for Code Generation

- **User Prompt**: "Generate a Python function for user login that takes a username and password, and authenticates against our user database."
- **Relevant AC Principles (Hypothetical Refs)**: AC-SECURITY-001 (Secure by Design), AC-PRIVACY-005 (Protect Credentials).
- **GS Engine - Derived Policy Snippets/Constraints (Conceptual)**:

```
Operational_Rule_Component {
Applies_To_Function_Type: "user_authentication",
      Language: "Python",
Constraint_ID: "AUTH-SEC-001-PW-HASHING",
Description: "Passwords must be hashed using a strong,
Implementation_Guidance_for_PGC_or_CodeReview_Rule:
   "REQUIRE: Use of 'bcrypt' or 'scrypt' or 'argon2'.
    PROHIBIT: Use of 'md5', 'sha1', 'plaintext'.
    ENSURE: Salt is unique per user and cryptographical
Trigger_for_PGC_Check: "on_code_generation_output(fu
```

Algorithm 2: GS Engine Rule Synthesis and Adaptation

```
Input  : Current_AC_Version, PGC_Feedback_Stream,
         External_Context_Stream,
         Human_Oversight_Directives
Output : Updated_Operational_Rule_Set

Variables: Operational_Rule_Set,
 AC_Interpreted_Constraints,
 Rule_Candidate_Buffer, GS_Learning_Parameters
Initialize():
    AC_Interpreted_Constraints ←
     InterpretAC(Current_AC_Version)
    Operational_Rule_Set ← LoadExistingRules() or
     GenerateBaselineRules(AC_Interpreted_Constraints)

    PGPSignRuleSet(Operational_Rule_Set)
    PushToPGC(Operational_Rule_Set)
Main_Loop():
    while True do
        Event ← WaitForEvent(PGC_Feedback,
         External_Context_Update, Human_Directive,
         Scheduled_Review_Trigger)
        switch Event.type do
            case PGC_Feedback do
                Affected_Rules ←
                 IdentifyRulesFromFeedback(Event.data,
                 Operational_Rule_Set)
                foreach Rule in Affected_Rules do
                    if Rule.Performance_Degraded or
                     Event.data.Indicates_Ineffectiveness
                     then
                        Candidate ← AdaptRule(Rule,
                         AC_Interpreted_Constraints,
                         Event.data.Context, Event.data,
                         GS_Learning_Parameters)
                        AddToBuffer(Candidate)

            case External_Context_Update do
                Relevant_AC ←
                 MapContextToACPrinciples(Event.data,
                 Current_AC_Version)
                AC_Interpreted_Constraints ←
                 UpdateInterpretation(Relevant_AC)
                Impacted_Rules ←
                 IdentifyImpactedRules(Operational_Rule_Set,
                 AC_Interpreted_Constraints)
                foreach Rule in Impacted_Rules do
                    Candidate ←
                     ResynthesizeRule(Rule.Original_Intent,
                     AC_Interpreted_Constraints,
                     Event.data)
                    AddToBuffer(Candidate)

                New_Needs ←
                 IdentifyNewRuleNeeds(AC_Interpreted_Constraints,
                 Event.data)
                foreach Need in New_Needs do
                    Candidate ←
                     SynthesizeNewRule(Need,
                     AC_Interpreted_Constraints,
                     Event.data)
                    AddToBuffer(Candidate)

            case Human_Directive do
```

```
}
Operational_Rule_Component {
Applies_To_Function_Type: "user_authentication",
Constraint_ID: "AUTH-SEC-002-RATE-LIMIT",
Description: "Login attempts must be rate-limited.",
Implementation_Guidance_for_PGC_or_CodeReview_Rule:
   "SUGGEST_PATTERN: Implement IP-based and/or userna
              rate limiting (e.g., max 5 attempts/minu
Trigger_for_PGC_Check: "on_code_generation_output(fu
}
```

Example D.2: High-Level Natural Language Policy Statement (Healthcare)

- **NL Policy Statement (from an organizational policy document ingested by GS Engine)**: "Patient medical history related to mental health conditions should only be accessible to authorized psychiatrists or psychologists directly involved in the patient's current treatment, and only after explicit patient consent for this specific type of data is verified for the current encounter."
- **Relevant AC Principles (Hypothetical Refs)**: AC-PRIVACY-001 (HIPAA Alignment - Sensitive Data), AC-CONSENT-001 (Explicit Consent for Sensitive Info).
- **GS Engine - Derived Operational Rule Components (Conceptual for `HCA-PHI-ACCESS-XXX` in Appendix B.1)**:

```
    // Component for Condition_Logic
 IF (Context.requested_PHI_sensitivity_level == 'MENTAL_
    NOT (Context.requesting_user_role IN ['PSYCHIATRIST'
       Context.user_is_treating_current_patient == TRUE
       Context.patient_consent_for_mental_health_data_c
    ) THEN Action_Allowed = FALSE

    // Component for Action(s)_Prescribed
    IF Action_Allowed == FALSE AND
   Context.requested_PHI_sensitivity_level == 'MENTAL_HE
    THEN
        BLOCK_DATA_ACCESS;
    LOG_EVENT(type: SENSITIVE_ACCESS_DENIED, reason: "Una
    NOTIFY_USER(message: "Access to this specific sensiti
    // Potentially trigger an alert to a privacy officer if
    END IF
```

These snippets illustrate how the GS Engine would need to perform semantic parsing, deontic logic extraction (identifying obligations, permissions, prohibitions), and map these to the structured fields of its operational rule language.

E DETAILED RISK/MITIGATION MATRIX FOR ACGS-PGP FRAMEWORK

This appendix outlines potential risks associated with the ACGS-PGP framework itself, along with mitigation strategies and Key Risk Indicators (KRIs).

Table 2: Risk Analysis and Mitigation Matrix for the ACGS-PGP Framework

Risk Category	Specific Risk Example	Mitigation Strategy	Key Risk Indicator(s) (KRIs)
Artificial Constitution (AC) Integrity & Relevance	AC principles become outdated, misaligned with societal values, or contain ambiguities leading to misinterpretation.	Formalized, multi-stakeholder AC amendment process with deliberation and cryptographic ratification. Regular scheduled reviews of AC by ethics/legal board. Use of structured or semi-formal language for AC principles to reduce ambiguity.	Time since last AC principle review/update. Number of interpretation disputes logged by GS Engine or HITL. Public/expert feedback on AC alignment.
	Unauthorized or malicious amendment to the AC.	Strong cryptographic controls for AC storage (e.g., DLT). Strict RBAC for amendment proposal/ratification. PGP-signed versions of AC. Quorum requirements for ratification.	Number of unauthorized AC modification attempts. Integrity check failures for AC versions. Audit trail of all amendment proposals and votes.
Self-Synthesizing (GS) Engine Functionality	GS Engine misinterprets AC principles, leading to flawed or biased operational rules. (The "Quis custodiet..." problem).	Rigorous testing and validation of GS Engine's interpretation capabilities. Formal verification of GS against meta-constitutional rules. HITL review and approval for critical or novel synthesized rules. Continuous monitoring of GS rule outputs for bias/fairness metrics. Feedback loops to retrain/fine-tune GS.	Rate of HITL overrides or rejections of GS-synthesized rules. Deviation of GS rule outputs from expected fairness/safety benchmarks. Number of PGC violations traced back to flawed GS rules.
	GS Engine develops "governance drift," where adapted rules subtly deviate from AC intent over time.	Periodic re-validation of GS rule sets against the AC. "External Monitor Agent" concept for verifying rule deltas (see Section 4.3). PGP assurance linking rules back to AC versions. Robust logging of all rule adaptations.	Semantic drift metrics between synthesized rules and AC principles. Frequency of significant rule adaptations without explicit AC changes. Time-to-detect for governance drift incidents.
	GS Engine performance degradation (e.g., slow rule synthesis) impacting governance agility.	Scalable architecture for GS Engine. Optimized AI models. Caching of interpreted AC components. Asynchronous rule synthesis for non-critical updates.	Average time for GS Engine to synthesize/adapt rules in response to triggers. Queue length for GS processing tasks. Resource utilization of GS Engine.
Prompt Governance Compiler (PGC) Effectiveness & Security	PGC fails to correctly enforce a synthesized rule, or its enforcement logic contains vulnerabilities.	Formal verification of critical PGC components. Rigorous testing (unit, integration, fuzzing). Secure coding practices for PGC development. Runtime self-monitoring of PGC health.	Number of known policy violations not caught by PGC (false negatives). Rate of incorrect PGC enforcements (false positives). Security vulnerabilities identified in PGC code.
	PGC performance overhead significantly impacts governed AI application responsiveness.	Optimized rule evaluation algorithms (e.g., Rete-like). Caching of compiled rules. Efficient runtime interception mechanisms. Hardware acceleration if necessary. (See Table 1 - or original Table 1 for PGC performance).	Average/P99 latency added by PGC checks. Throughput reduction of governed AI application. PGC CPU/memory utilization.
	PGP Assurance mechanisms are compromised (e.g., private key theft).	Hardware Security Modules (HSMs) for PGP key storage. Strict key management policies. Regular audits of cryptographic components. Short-lived certificates/signatures where appropriate.	Number of PGP signature verification failures. Alerts on unauthorized key access attempts. Time since last cryptographic audit.
Human Oversight & Interaction	Human overseers become complacent ("automation bias") or lack expertise to effectively review AC/GS/PGC outputs.	Mandatory HITL for critical decisions/changes. Continuous training for human overseers. XAI tools to explain GS/PGC decisions to re-	Rate of HITL approvals without modification (potential rubber-stamping). Time taken for HITL reviews. Feedback from overseers on