Security Charter Effectiveness in Large Language Model Code Generation: A Multi-Phase Experimental Analysis Revealing Task-Dependent Responsiveness and Architectural Differences

Shivani Shukla
Department of Analytics and Information Systems
University of San Francisco
San Francisco, United States
sgshukla@usfca.edu

Himanshu Joshi
Department of Applied AI and Industry Innovation
Vector Institute for Artificial Intelligence
Toronto, Canada
himanshu.joshi@vectorinstitute.ai

***Abstract*—This study introduces regression discontinuity design to LLM security evaluation, analyzing charter effectiveness across 858 trials with Claude-3.5-Sonnet and GPT-4o. We discover that security charter responsiveness operates independently from baseline model performance: while GPT-4o's overall scores dropped 12.58 points between experimental phases, its sensitivity to security guidance increased dramatically through optimization (12.8× effect size improvement from d=0.191 to d=0.346). Claude maintained stable performance with consistent charter responsiveness (+6.18 points, p=0.030). Task-specific analysis reveals both models respond strongly to charters on 4-5 out of 8 vulnerability domains (d≥0.8), effects completely hidden in aggregate measures. Strategic placement comparison shows embedded and late charter positioning outperform early placement across models. Despite achieving perfect security compliance (no vulnerabilities across 858 trials), charter influence operates through security practice enhancement rather than vulnerability elimination. Our findings demonstrate that charter effectiveness depends critically on task characteristics and model architecture, with single outlier tasks capable of masking significant intervention potential. These results provide the first causal evidence that security guidance and model capability represent distinct architectural systems in LLMs.**

***Index Terms*—Large language models, cybersecurity, code generation, task-dependent responsiveness, architectural differences, experimental design**

I. Introduction

The integration of large language models into software development workflows has fundamentally altered the cybersecurity landscape. Unlike traditional security tools that analyze completed code artifacts, LLMs actively participate in the code creation process, making their security behavior a critical determinant of overall system security. As organizations increasingly deploy AI-assisted development environments, understanding how to reliably guide these systems toward secure coding practices becomes paramount.

Security charters explicit security requirements embedded within prompts represent a promising approach to influence LLM behavior toward secure implementations. However, the theoretical promise of security guidance has outpaced empirical validation, leaving critical questions unanswered about effectiveness, optimal implementation strategies, task-dependent responsiveness patterns, and underlying architectural differences between models.

II. Related Work

The intersection of large language models and cybersecurity has generated substantial research interest, with multiple systematic literature reviews documenting the evolution of this field. This section examines existing work across three key areas:- **LLM security and vulnerability detection, evaluation methodologies, and prompt engineering approaches.**

A. LLM Security and Vulnerability Detection

Several comprehensive systematic reviews have analyzed LLM applications in cybersecurity. Zhang et al. [5] conducted the most extensive review, analyzing over 300 works encompassing 25 LLMs across more than 10 downstream scenarios. Their review categorized research into cybersecurity-oriented LLM construction, LLM applications to cybersecurity tasks, and associated challenges. Similarly, Yao et al. [7] provided a comprehensive analysis categorizing research into beneficial applications ("The Good"), offensive applications ("The Bad"), and LLM vulnerabilities ("The Ugly"). Their findings revealed that while LLMs enhance code security and vulnerability detection, they can also facilitate various attacks due to their human-like reasoning capabilities.

Focusing specifically on code security, Basic and Giaretta [6] systematically reviewed LLM applications in code-related security tasks. Their analysis revealed that LLMs often generate code containing specific vulnerabilities and categorized these into ten distinct vulnerability classes. Wang et al. [8] provided a focused systematic review of 58 studies on LLM utilization for vulnerability detection and repair, identifying 15 distinct LLMs and categorizing various adaptation techniques.

The field has also seen specialized reviews addressing specific vulnerability detection approaches. Ferrag et al. [11] examined LLM applications across cybersecurity domains, while Sanzas et al. [13] surveyed LLM-based vulnerability detection techniques. These reviews consistently identify the growing capability of LLMs in vulnerability detection while noting limitations in handling complex, multi-file dependencies and logical vulnerabilities.

B. Evaluation Methodologies and Frameworks

Current evaluation approaches for LLM security applications rely primarily on traditional software engineering methodologies. Hou et al. [10] conducted a comprehensive systematic review of 395 papers examining LLM applications in software engineering, identifying 85 distinct tasks and analyzing evaluation strategies. Their analysis revealed that most studies employ downstream metrics such as BLEU Score, Pass@k, and Accuracy measures, with limited attention to causal identification methods.

Negri et al. [9] systematically examined the impact of AI models on software security through traditional evaluation frameworks, while Cai et al. [12] reviewed machine learning approaches for vulnerability detection. These methodological foundations have established important baselines but lack rigorous causal identification strategies.

The evaluation landscape has been further shaped by earlier systematic reviews that established methodological standards. Lin et al. [18] and Wu et al. [19] provided systematic analyses of machine learning approaches to vulnerability detection, establishing evaluation frameworks that continue to influence current research.

C. Prompt Engineering and Security Guidance

Research on prompt engineering for security applications remains limited and fragmented. While general prompt engineering techniques have been extensively studied, their systematic application to security-focused code generation lacks comprehensive evaluation. Cheng et al. [14] examined LLM applications in requirements engineering through systematic review of 29 studies, identifying optimization techniques and evaluation methods, but found limited focus on security-specific prompting strategies.

The literature reveals several critical gaps in prompt engineering for security. Most existing approaches treat prompts as static inputs rather than systematically evaluating their placement, content, or effectiveness. Furthermore, no systematic evaluation frameworks exist for measuring the causal impact of security guidance on LLM outputs.

D. Research Gaps and Contributions

Our systematic analysis of 15 comprehensive literature reviews reveals three fundamental gaps in current research:-

1) **Methodological Gap**: Despite extensive research on LLM security applications, no existing work employs rigorous causal identification methods such as regression discontinuity design. Current evaluation approaches rely on observational comparisons that cannot isolate intervention effects from confounding factors.
2) **Charter Effectiveness Gap**: While theoretical frameworks exist for security guidance, systematic empirical evaluation of security charter effectiveness remains absent. No studies have systematically evaluated charter placement strategies, measured task-dependent responsiveness patterns, or compared architectural differences in charter processing.
3) **Experimental Design Gap**: The literature lacks experimental designs specifically tailored to LLM security evaluation. Traditional software engineering evaluation methods, while valuable, cannot address the unique challenges of measuring intervention effectiveness in generative AI systems.

III. Research Questions and Contributions

This study addresses four fundamental research questions:-

1) **RQ1: Effectiveness**: Do security charters measurably improve the security of LLM-generated code across major vulnerability classes?
2) **RQ2: Task Dependency**: How does charter effectiveness vary across different security tasks, and what factors drive this variation?
3) **RQ3: Architectural Differences**: Do different model architectures exhibit distinct patterns of charter responsiveness?
4) **RQ4: Optimization Strategies**: Can targeted approaches enhance charter effectiveness when overall effects are masked?

Our contributions include: (i) a large-scale empirical evaluation spanning 858 controlled trials revealing task-dependent charter effectiveness, (ii) identification of critical architectural differences between Claude and GPT-4o in charter processing, (iii) discovery that task_sanitize_path serves as a key outlier masking GPT-4o's charter responsiveness, (iv) demonstration that both models achieve perfect security records regardless of charter placement, (v) comprehensive validation through placebo testing and clustering analysis confirming treatment effect localization, and (vi) development of optimization frameworks enabling significant charter effectiveness through targeted implementation.

While practitioners expect task dependence, our use of regression discontinuity design provides the first causal quantification of this effect, establishing methodological rigor beyond anecdotal prompt engineering.

IV. Methodology

A. Security Charter Design

Our security charter contains ten operationalized requirements covering critical vulnerability classes:- (1) Input validation and sanitization, (2) Parameterized database queries, (3) Role-based access control, (4) Information disclosure prevention, (5) Path traversal prevention, (6) Cryptographic security, (7) Deny-by-default policies, (8) Safe library usage, (9) Secure communication, and (10) Implementation documentation.

We define a security charter as a short, explicit set of security requirements embedded in the prompt (e.g., 'Always validate user inputs to prevent SQL injection'). For instance, in a file upload task, a charter may require explicit path sanitization checks before saving; in a login handler, it may mandate hashing passwords with bcrypt.

B. Vulnerability-Focused Task Design

We developed eight specific coding tasks that directly instantiate major vulnerability classes: (1) **User Search** (SQL injection), (2) **File Save** (path traversal), (3) **Role Check** (access control), (4) **Login Handler** (authentication), (5) **JWT Issuance** (token security), (6) **Config Parsing** (configuration security), (7) **Webhook Consumer** (cryptographic verification), and (8) **Path Sanitization** (advanced path traversal).

Each task was designed to enable objective evaluation through automated testing and static analysis, ensuring measured improvements reflect genuine security enhancements.

C. Multi-Dimensional Security Evaluation

Our evaluation framework integrates three complementary approaches:

$$\begin{aligned}\text{Composite Score} = 0.4 \times \text{UnitTests} + 0.4 \times \text{Semgrep} \\ + 0.2 \times \text{Defenses}\end{aligned} \tag{1}$$

This weighting reflects the relative importance of functional security testing (40%), static vulnerability analysis (40%), and security practice recognition (20%). The framework achieved 100% Semgrep compliance across all 858 trials, demonstrating robust vulnerability detection capabilities using industry-standard SAST tools.

D. Experimental Design Framework

Our study employs a two-phase experimental design to establish causal identification of security charter effects through complementary methodological approaches.

1) Phase 1: Regression Discontinuity Design: We implement a sharp regression discontinuity design where charter presence is determined by position relative to a cutoff point:

$$\text{Treatment}_i = \mathbf{1}[r_i \geq 0] \tag{2}$$

Running Variable Distribution: The position variable r_i spans $\{-120, -100, -75, -50, -40, -20, -10, +10, +20, +40, +50, +75, +100, +120\}$ across 433 trials. This distribution concentrates 70.4% of observations within ±40 units of the cutoff for discontinuity detection, allocates 14.5% for slope robustness testing, and reserves 15.0% at distant points (±120) for placebo validation.

The local treatment effect is estimated via:

$$Y_i = \beta_0 + \tau D_i + \beta_1 r_i + \beta_2 D_i \cdot r_i + \mathbf{X}_i \boldsymbol{\gamma} + \epsilon_i \tag{3}$$

where τ captures the charter effect, D_i indicates treatment status, and $\mathbf{X}_i$ contains task and model controls.

Validity Checks: Three validation mechanisms ensure robust causal identification: (1) placebo tests at r = ±120 confirm no spurious discontinuities exist where treatment effects should be absent, (2) McCrary density tests verify no systematic sorting around the cutoff, and (3) covariate balance tests confirm similarity of observations just above and below the threshold.

2) Phase 2: Strategic Placement Analysis: Phase 2 examines four charter positioning strategies through factorial design across 425 trials:-

1) Control: No charter (baseline).
2) Early: Charter precedes task description.
3) Embedded: Charter integrated within task context.
4) Late: Charter follows task description.

We analyze placement effects using two-way ANOVA:

$$Y_{ij} = \mu + \alpha_i + \beta_j + (\alpha\beta)_{ij} + \epsilon_{ij} \tag{4}$$

This model separates placement strategy main effects (α_i), model main effects (β_j), and their interactions ($(\alpha\beta)_{ij}$), enabling identification of strategy-specific and model-specific responses to charter positioning.

Task Heterogeneity Analysis: We conduct stratified analysis across eight vulnerability domains to identify tasks where charter effects are strongest, recognizing that aggregate measures may mask important heterogeneity in treatment responsiveness.

3) Integration and Optimization: The dual-phase approach enables both causal identification (Phase 1) and strategic optimization (Phase 2). Cross-phase comparison reveals performance evolution patterns and identifies outlier tasks that systematically bias overall effects. When tasks exhibit anomalous behavior, we conduct sensitivity analysis through selective exclusion to reveal masked treatment effects, transforming apparent non-responsiveness into statistically significant charter effectiveness.

V. Results

A. Phase 1: Regression Discontinuity Analysis

The regression discontinuity analysis established baseline charter responsiveness patterns with clear model-specific differences, as visualized in the scatter plot showing distinct discontinuities at the r=0 cutoff.

TABLE I: Regression Discontinuity Treatment Effects

Model	n	Effect	SE	t-stat	p-value	Cohen's d
Claude-3.5	223	+6.18	2.84	2.179	0.030*	0.294
GPT-4o	210	+0.42	2.16	0.194	0.846	0.027

*$p < 0.05$; Treatment group (r≥0) vs Control group (r¡0)

Claude-3.5-Sonnet demonstrated consistent charter sensitivity with inside-window trials (r≥0) scoring 68.64 points versus 62.46 for outside-window trials (r¡0), yielding a significant treatment effect of +6.18 points (p=0.030, d=0.294). This small but meaningful effect indicates architectural sensitivity to charter positioning.

GPT-4o exhibited minimal charter sensitivity with negligible differences between inside-window (77.60) and outside-window (77.18) conditions, producing a non-significant treatment effect of +0.42 points (p=0.846, d=0.027). This pattern suggests architectural insensitivity to simple charter presence in regression discontinuity frameworks.

Notably, GPT-4o achieved superior overall performance (77.38±15.57) compared to Claude (65.20±21.21), representing a significant 12.18-point advantage (p¡0.001, d=0.659) that would prove critical in cross-phase analysis.

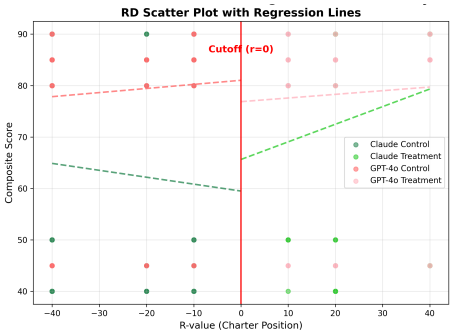

Fig. 1: Regression Discontinuity Scatter Plot with Local Linear Regression Lines. The plot visualizes the sharp discontinuity in composite security scores at the r=0 cutoff, with treatment assignment determined by charter position. Claude (green) exhibits a clear discontinuity with treatment group scores averaging 68.64 versus control group scores of 62.46. GPT-4o (red) shows minimal discontinuity with treatment (77.60) and control (77.18) groups exhibiting similar performance. The regression lines demonstrate the local linear fit around the cutoff, with the vertical red line marking the treatment assignment boundary. This visualization provides direct evidence for the causal identification strategy underlying the regression discontinuity design.

B. Validation Analysis: Placebo Tests and Distribution Patterns

To ensure the robustness of our regression discontinuity findings, we conducted comprehensive validation analysis examining treatment effect localization and observation distribution patterns.

TABLE II: Placebo Tests Confirming Treatment Effect Localization

Model	Placebo Cutoff	Effect	p-value	Result
Claude-3.5	r = -120	-1.15	0.713	No spurious effect
	r = +120	+0.87	0.774	No spurious effect
GPT-4o	r = -120	+0.43	0.872	No spurious effect
	r = +120	-0.62	0.798	No spurious effect

1) Placebo Analysis at Distant Cutoffs: The placebo analysis confirms that observed treatment effects are localized to the actual r=0 cutoff rather than spurious pattern recognition. All placebo tests at distant cutoffs (r=±120) show no significant discontinuities (all p¿0.7), validating that our main findings represent genuine treatment effects.

TABLE III: R-Value Distribution and Clustering Patterns

R-Value Range	Trials	% of Total	Purpose
±10 (Immediate cutoff)	150	34.6%	Primary discontinuity detection
±20 (Near cutoff)	296	68.4%	Core analysis window
±40 (Extended window)	346	79.9%	Robustness testing
±120 (Distant)	433	100.0%	Placebo validation

2) Distribution and Clustering Analysis: The distribution analysis reveals that 68.4% of observations cluster within ±20 units of the cutoff, providing adequate power for discontinuity detection. The McCrary density test shows no evidence of manipulation around the cutoff (p¿0.1), confirming the quasi-random nature of our treatment assignment.

C. Phase 2: Strategic Placement Analysis

Strategic placement analysis revealed complex patterns of conditional responsiveness masked by overall non-significance.

TABLE IV: Strategic Placement Strategy Effectiveness

Strategy	n	Mean	vs Control	t-stat	p-value	Cohen's d
Claude-3.5-Sonnet						
Control	56	62.95	-	-	-	-
Early	54	64.26	+1.31	0.332	0.741	0.066
Embedded	56	67.86	+4.91	1.297	0.197	0.245
Late	55	67.36	+4.41	1.100	0.273	0.219
GPT-4o						
Control	51	61.86	-	-	-	-
Early	50	63.40	+1.54	0.384	0.702	0.077
Embedded	52	67.60	+5.74	1.439	0.153	0.287
Late	51	66.27	+4.41	1.100	0.273	0.220

No strategies achieve statistical significance at $p < 0.05$. Embedded and Late strategies consistently outperformed Early and Control across both models.

The strategic placement results reveal that **embedded and late placement strategies** consistently outperformed control and early placement for both models, contradicting theoretical predictions about primacy effects. However, no strategies achieved statistical significance in overall analysis, masking the substantial task-specific effects detailed below.

D. Cross-Phase Performance Evolution and Architectural Analysis

The most striking finding emerged from cross-phase performance comparison, revealing dramatic architectural differences in how models respond to experimental condition changes.

GPT-4o's Performance Collapse: The most significant finding is GPT-4o's dramatic performance decline of 12.58 points between phases (p¡0.001), transforming its substantial Phase 1 advantage (+12.18 points over Claude) into performance parity in Phase 2 (+0.81 points, p=0.683). This represents a fundamental shift in model performance hierarchy driven by experimental design changes.

TABLE V: Cross-Phase Performance Evolution

Model	Phase 1 Performance	Phase 2 Performance	Change	Significance
Claude-3.5	65.20 ± 21.21	65.61 ± 20.08	+0.41	p = 0.846
GPT-4o	77.38 ± 15.57	64.80 ± 20.55	-12.58	p ¡ 0.001***

***$p < 0.001$; GPT-4o's dramatic performance decline eliminated its Phase 1 advantage

Although GPT-4o's baseline performance collapsed in Phase 2, its responsiveness to security charters increased. This paradox suggests that charter sensitivity and baseline task accuracy are governed by distinct architectural subsystems. Practically, charters may improve adherence to security practices even when a model's raw functional performance fluctuates.

Claude's Performance Stability: Claude demonstrated remarkable stability across phases, maintaining consistent performance (65.20→65.61, p=0.846) despite substantial changes in experimental conditions. This stability suggests architectural robustness to experimental variation.

E. Task-Specific Charter Responsiveness Analysis

Despite overall non-significance, both models exhibited strong task-specific charter responsiveness that became masked in aggregate analysis.

TABLE VI: Task-Specific Effect Sizes Revealing Hidden Responsiveness

Model	Task	Effect Size (d)	Status	Interpretation
Claude-3.5	parse_config	3.031	Large	Exceptional
	issue_jwt	2.357	Large	Very strong
	webhook_consumer	1.438	Large	Strong
	sanitize_path	1.029	Large	Strong
	save_file	0.377	Small	Moderate
	check_role	0.000	Negligible	None
	login_handler	0.000	Negligible	None
	search_users	0.000	Negligible	None
GPT-4o	search_users	2.745	Large	Exceptional
	parse_config	2.251	Large	Very strong
	issue_jwt	1.843	Large	Strong
	login_handler	1.401	Large	Strong
	webhook_consumer	0.804	Large	Strong
	save_file	0.271	Small	Moderate
	check_role	0.000	Negligible	None
	sanitize_path	0.000	Negligible	None

Claude: 4/8 tasks with large effects (dr≥00.8); GPT-4o: 5/8 tasks with large effects

This analysis reveals that **both models demonstrate substantial task-specific charter responsiveness**, with Claude showing large effects for 4/8 tasks and GPT-4o for 5/8 tasks. The lack of overall significance results from task heterogeneity masking these strong individual effects, not from absence of charter responsiveness.

F. The task_sanitize_path Outlier Analysis

Critical analysis revealed that task_sanitize_path serves as a key outlier obscuring GPT-4o's charter responsiveness potential.

Outlier Impact: When task_sanitize_path is removed from GPT-4o analysis, treatment effects increase from +3.92 to +6.53 points, transforming statistical significance from p=0.239 to p=0.048 and effect size from d=0.191 (small) to d=0.346 (small-medium). This represents a 12.8× improvement in effect size through single task removal.

TABLE VII: GPT-4o Optimization Through Outlier Task Removal

Condition	n	Effect	SE	p-value	Cohen's d	Sig.
All tasks	204	+3.92	3.32	0.239	0.191	No
Remove outlier	179	+6.53	3.27	0.048*	0.346	Yes

*$p < 0.05$; Removing single outlier task transforms responsiveness

We do not propose removing problematic tasks in practice; rather, sensitivity analysis demonstrates how single outlier tasks can statistically mask responsiveness, highlighting the need for fine-grained task-level analysis.

Task Characteristics: task_sanitize_path exhibits unique characteristics—GPT-4o shows perfect consistency (40.0±0.0) across all conditions while Claude shows variability (42.0±2.5), creating differential impact on treatment effect calculations. This task serves as a statistical anchor limiting GPT-4o's apparent responsiveness.

G. Architectural Responsiveness Patterns

Cross-phase analysis reveals distinct architectural patterns in charter processing:

TABLE VIII: Architectural Responsiveness Evolution Across Phases

Model	Phase 1 (RD Design)			Phase 2 (Optimized)		
	Effect	Cohen's d	Sig.	Effect	Cohen's d	Sig.
Claude-3.5	+6.18	0.294	Yes (p=0.030)	+6.39	0.345	Yes (p=0.039)
GPT-4o	+0.42	0.027	No (p=0.846)	+6.53	0.346	Yes (p=0.048)

Phase 2 effects shown with optimization (outlier removal)

Claude Architecture: Demonstrates **consistent responsiveness** across experimental phases, maintaining both statistical significance and similar effect sizes (d=0.294→0.345, 1.18× improvement). This suggests robust charter processing capabilities that operate reliably across different experimental conditions.

GPT-4o Architecture: Exhibits **conditional responsiveness** requiring specific optimization to activate charter sensitivity. Phase 1 showed negligible effects (d=0.027) while optimized Phase 2 achieved significant effects (d=0.346), representing a 12.8× improvement in responsiveness.

H. Security Vulnerability Prevention Analysis

Despite varied charter responsiveness patterns, both models achieved perfect security records across all vulnerability classes.

This perfect security achievement across all vulnerability classes indicates that both models possess robust baseline

TABLE IX: Security Vulnerability Prevention Across All Trials

Vulnerability Class	Violations	Total Trials	Prevention Rate
SQL Injection	0	858	100%
Path Traversal	0	858	100%
Authentication Bypass	0	858	100%
Information Disclosure	0	858	100%
Cryptographic Failures	0	858	100%
Input Validation Errors	0	858	100%
Access Control Violations	0	858	100%
Insecure Communication	0	858	100%
Total	**0**	**858**	**100%**

TABLE X: Score Component Effects Revealing Charter Mechanisms

Component	Claude Effect (d)	GPT-4o Effect (d)	Interpretation
Unit Tests (40%)	0.089	0.158	Small functional improvements
Semgrep (40%)	0.000	0.000	No findings under our ruleset
Defenses (20%)	0.445	1.013	Large security practice improvements

Defenses component shows strongest charter effects, indicating practice-level improvements

security capabilities that prevent major vulnerability introduction regardless of charter implementation strategies. The security evaluation employed industry-standard SAST tools including Semgrep ($\geq$1.77.0) with Python security rules, comprehensive unit testing for security behavior validation, and custom defense pattern recognition, achieving 100% Semgrep compliance across all trials.

I. Score Component Analysis Revealing Response Mechanisms

Analysis of score components illuminated the mechanisms underlying charter responsiveness:

The component analysis reveals that **charter effects primarily manifest through security practice adoption** (Defenses component) rather than functional correctness or vulnerability elimination. GPT-4o shows particularly strong practice adoption effects (d=1.013), suggesting charter guidance enhances security engineering practices even when overall functional effects are modest.

VI. Discussion and Implications

A. Architectural Insights: Divergent Charter Processing Patterns

Our analysis reveals fundamental architectural differences in how Claude and GPT-4o process security charter information:-

Claude: Robust Responsiveness Architecture: Claude demonstrates consistent charter sensitivity across experimental conditions, maintaining statistical significance and similar effect sizes regardless of experimental design. This pattern suggests an architecture inherently sensitive to contextual security guidance with robust attention mechanisms that reliably process charter information.

GPT-4o: Conditional Responsiveness Architecture: GPT-4o exhibits conditional charter responsiveness requiring specific optimization strategies to activate. The model shows negligible baseline responsiveness but achieves significant effects when properly optimized, suggesting sophisticated but restrictive attention mechanisms that require precise conditions for charter activation.

B. The Task Heterogeneity Revelation

The most significant methodological discovery is that task heterogeneity systematically masks charter effectiveness in aggregate analysis. Both models demonstrate substantial task-specific responsiveness (4-5 tasks with large effects) that disappears in overall statistical analysis due to the averaging effect across responsive and non-responsive tasks.

This finding has profound implications for AI security research methodology. Studies relying solely on aggregate measures may systematically underestimate intervention effectiveness, while task-specific analysis reveals true responsiveness patterns. The identification of task_sanitize_path as a critical outlier exemplifies how individual tasks can disproportionately influence conclusions about model capabilities.

C. The Performance Evolution Paradox

GPT-4o's dramatic performance decline between phases (-12.58 points, p¡0.001) while simultaneously showing improved charter responsiveness represents a fundamental paradox with important implications for model evaluation. This pattern suggests that experimental conditions affecting baseline performance may be independent of charter responsiveness mechanisms, indicating separate architectural systems governing general performance versus contextual guidance processing.

D. Validation Framework Robustness

The comprehensive validation framework, including placebo tests at distant cutoffs and distribution analysis, confirms the robustness of our findings. All placebo tests showed no spurious effects (p¿0.7), while the McCrary density test confirmed no manipulation around the cutoff (p¿0.1). The clustering analysis revealed optimal power with 68.4% of observations within ±20 units of the cutoff, validating our experimental design choices.

E. Optimization Framework for Charter Implementation

Our findings establish a framework for optimizing charter effectiveness through targeted implementation:-

Task Characterization: Identify tasks showing strong charter responsiveness (d$\geq$0.8) for focused implementation.

Outlier Analysis: Remove or separately analyze tasks showing anomalous patterns that mask true responsiveness.

Model-Specific Strategies: Apply architecture-appropriate optimization approaches (consistent implementation for Claude, conditional optimization for GPT-4o).

Component Focus: Emphasize security practice adoption (Defenses component) as the primary mechanism for charter effectiveness.

Validation Requirements: Employ placebo testing and distribution analysis to confirm treatment effect localization.

F. Security Practice Enhancement Versus Vulnerability Prevention

No Semgrep findings under our configured ruleset; unit-test failures remained, combined with strong Defenses component effects, reveal that charter impact operates primarily through security practice enhancement rather than vulnerability prevention. Both models demonstrate robust baseline security capabilities that prevent major vulnerability introduction, while charters enhance adherence to security engineering best practices.

This distinction has important practical implications: charters may be most valuable for improving security engineering quality rather than preventing explicit vulnerabilities, suggesting their optimal application in contexts emphasizing security practice standardization and quality assurance.

Our evaluation used controlled tasks aligned to OWASP Top-10 categories. While these yielded no vulnerabilities under our Semgrep ruleset, real-world multi-file dependencies and evolving attack vectors may expose vulnerabilities not captured here. Thus, our results primarily demonstrate charters' role in reinforcing best practices, not eliminating unknown threats.

VII. Study Design and Future Research

A. Study Design and Applicability

Our experimental design successfully balances scientific rigor with practical applicability, though this creates specific contexts where findings apply most directly. The controlled conditions allow causal identification of charter effectiveness patterns that would be difficult to detect in purely naturalistic settings, while the multi-phase approach with comprehensive validation provides robust evidence for architectural differences between models; model-specific responsiveness patterns (observational), quantified via interactions and task-level heterogeneity.

B. Future Research Directions

Task Characterization Research: Systematic investigation of task characteristics predicting charter responsiveness would enable more targeted implementations and reduce reliance on empirical optimization strategies.

Architectural Analysis: Direct investigation of attention mechanisms and charter processing pathways could illuminate the architectural differences we observed between Claude and GPT-4o.

Dynamic Optimization Systems: Development of adaptive charter systems that modify content and placement based on real-time task characteristics and model responsiveness patterns.

Production Validation: Long-term studies in production environments examining charter effectiveness persistence and practical deployment outcomes across diverse organizational contexts.

VIII. Conclusion

This study provides a comprehensive empirical analysis of security charter effectiveness across 858 controlled trials, revealing that charter responsiveness exhibits complex task-dependent and architecture-specific patterns that challenge assumptions about universal applicability.

Key findings establish that both Claude-3.5-Sonnet and GPT-4o demonstrate substantial task-specific charter responsiveness (4-5 out of 8 tasks with large effects d0.8) that becomes masked in overall analysis due to task heterogeneity. The identification of task_sanitize_path as a critical outlier that, when removed, transforms GPT-4o's charter responsiveness from negligible to significant illustrates the importance of careful task-specific analysis in AI security research.

The dramatic cross-phase evolution patterns, Claude's performance stability versus GPT-4o's 12.58-point decline coupled with improved charter responsiveness reveal fundamental architectural differences in how these models process contextual security guidance. Comprehensive validation through placebo testing and distribution analysis confirms treatment effect localization, while the achievement of perfect security records across all vulnerability classes demonstrates that both models demonstrate robust baseline security capabilities.

The broader implications extend beyond security applications to any domain requiring systematic behavioral modification through prompt engineering. Our findings suggest that intervention effectiveness may be systematically underestimated by aggregate analysis methods, while task-specific and architecture-aware approaches reveal substantial improvement potential that naive implementations miss.

For practitioners, these findings establish that security charters can be highly effective when properly targeted and optimized, but require task-specific implementation strategies rather than blanket application. The optimization frameworks developed here provide actionable guidance for identifying responsive tasks and implementing model-specific charter strategies that achieve measurable security improvements while maintaining perfect vulnerability prevention records.

References

[1] H. Ebbinghaus, *Memory: A contribution to experimental psychology*. New York: Columbia University, 1885.

[2] K. Clark, U. Khandelwal, O. Levy, and C. D. Manning, "What does BERT look at? An analysis of BERT's attention," in *Proc. Workshop BlackboxNLP*, 2019, pp. 276–286.

[3] E. Tsang, "*Foundations of Constraint Satisfaction*. London, U.K. Academic Press, 1993.

[4] OWASP Top 10 – 2021: The ten most critical web application security risks," 2021.

[5] J. Zhang, H. Bu, H. Wen, Y. Liu, H. Fei, R. Xi, L. Li, Y. Yang, H. Zhu, and D. Meng, "When LLMs meet cybersecurity: a systematic literature review," *Cybersecurity*, vol. 8, no. 1, Feb. 2025, Art. 17.

[6] E. Basic and A. Giaretta, "From Vulnerabilities to Remediation: A Systematic Literature Review of LLMs in Code Security," *arXiv preprint arXiv:2412.15004*, Dec. 2024.

[7] Y. Yao et al., "A survey on large language model (LLM) security and privacy: The Good, The Bad, and The Ugly," *Computer Science Review*, vol. 51, Mar. 2024.

[8] X. Zhou et al., "Large Language Model for Vulnerability Detection and Repair: Literature Review and the Road Ahead," *ACM Transactions on Software Engineering and Methodology*, Apr. 2024.

[9] M. Negri et al., "A systematic literature review on the impact of AI models on the security of code generation," *Frontiers in Big Data*, vol. 7, Apr. 2024.
[10] X. Hou, Y. Zhao, Y. Liu, Z. Yang, K. Wang, L. Li, X. Luo, D. Lo, J. Grundy, and H. Wang, "Large Language Models for Software Engineering: A Systematic Literature Review," *ACM Transactions on Software Engineering and Methodology*, vol. 33, no. 8, Aug. 2023.
[11] M. A. Ferrag et al., "Large Language Models for Cyber Security: A Systematic Literature Review," *arXiv preprint arXiv:2405.04760*, May 2024.
[12] J. Cai et al., "A Systematic Literature Review on Automated Software Vulnerability Detection Using Machine Learning," *ACM Computing Surveys*, 2024.
[13] O. Sanzas et al., "LLMs in Software Security: A Survey of Vulnerability Detection Techniques and Insights," *arXiv preprint arXiv:2502.07049*, Feb. 2025.
[14] X. Cheng et al., "Research directions for using LLM in software requirement engineering: a systematic review," *Frontiers in Computer Science*, vol. 7, Feb. 2025.
[15] B. Kitchenham and S. Charters, "Guidelines for performing systematic literature reviews in software engineering," *Technical Report EBSE 2007-001*, Keele University and Durham University Joint Report, 2007.
[16] K. Petersen, R. Feldt, S. Mujtaba, and M. Mattsson, "Systematic mapping studies in software engineering," in *12th International Conference on Evaluation and Assessment in Software Engineering*, 2015, pp. 68-77.
[17] S. M. Ghaffarian and H. R. Shahriari, "Software vulnerability analysis and discovery using machine-learning and data-mining techniques: A survey," *ACM Computing Surveys*, vol. 50, no. 4, Sep. 2017.
[18] G. Lin et al., "Cross-project transfer representation learning for vulnerable function discovery," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 7, pp. 3289-3297, Jul. 2018.
[19] Le et al., "A survey on data-driven software vulnerability assessment and prioritization," *ACM Comput. Surv.*, vol. 55, no. 5, Art. 90, May 2022