Avaliação simétrica de detecção e *over-blocking* na geração de políticas OPA/Rego por LLMs

Eduardo Rodrigues, Altair Olivo Santin,
Eduardo K. Viegas, Fellipe Medeiros Veiga

[1]Pontifícia Universidade Católica do Paraná (PUCPR)
Curitiba – PR – Brasil

rodrigues.e@pucpr.edu.br, altair.santin@pucpr.br

eduardo.kugler@pucpr.br, fellipe.veiga@pucpr.br

Resumo. *As métricas usadas para avaliar geração de políticas* policy-as-code *por LLMs medem detecção, mas quase nunca especificidade. Propomos a* strict_pass@1*, métrica pareada construída sobre avaliação por plano dual: a política precisa disparar no plano Terraform vulnerável e ficar silenciosa no plano seguro espelhado (*minimal pair*). Em 30 cenários, 5 LLMs e 900 execuções ($K = 3$), comparamos* Zero-shot *(ZS) e* Recursive Criticism and Improvement *(RCI) com* Compiler-in-the-Loop*. O RCI aumenta a aprovação na verificação estática e a detecção, mas piora a* strict_pass@1 *em quatro dos cinco modelos. No* gpt-4o *a queda vai de 28,9% a 0,0% por execução, sem que nenhum dos 30 cenários produzisse política utilizável sob RCI ($p = 0{,}004$, teste pareado por cenário). Medir apenas detecção, portanto, não caracteriza a utilidade operacional das políticas geradas.*

Abstract. *Metrics for LLM-generated policy-as-code measure detection, but they rarely measure specificity. We propose* strict_pass@1*, a paired metric built on a dual-plane evaluation: a policy has to fire on the vulnerable Terraform plan and stay silent on its mirrored secure plan (minimal pair). Across 30 scenarios, 5 LLMs and 900 runs ($K = 3$), we compare Zero-shot (ZS) with Recursive Criticism and Improvement (RCI) under Compiler-in-the-Loop. RCI raises static-check pass rates and detection, but it degrades* strict_pass@1 *in four of five models. For* gpt-4o *the metric falls from 28.9% to 0.0% per run, with no scenario out of 30 yielding a usable policy under RCI ($p = 0.004$, scenario-level paired test). Detection metrics alone do not tell us whether a generated policy is operationally useful.*

1. Introdução

A definição de topologias de nuvem por Infraestrutura como Código (IaC) consolidou-se como padrão da indústria, com o Terraform como principal tecnologia de provisionamento *multi-cloud*. Acompanhando essa evolução, ferramentas de *policy-as-code*, como o *Open Policy Agent* (OPA) e sua linguagem Rego, emergiram como mecanismo de governança automatizada, permitindo expressar requisitos de segurança como código verificável e versionável. A autoria manual dessas políticas exige domínio de segurança e de linguagem declarativa, o que motiva o uso de *Large Language Models* (LLMs) para automatizá-la. Trabalhos recentes apontam direções: [Zhang et al. 2025] descrevem a "lacuna de implantabilidade" (*deployability gap*), na qual códigos sintaticamente corretos falham em produção; [Li et al. 2025] mostram que a complexidade das infraestruturas modernas induz LLMs a gerar configurações padrão inseguras; [Bruni et al. 2025] reportam reduções de

até 68,7% em vulnerabilidades com *prompting* iterativo do tipo *Recursive Criticism and Improvement* (RCI), e mecanismos de retroalimentação a partir de verificadores automáticos elevam a correção funcional de *templates* de IaC [Zhang et al. 2025, Tony et al. 2025].

Há, porém, uma assimetria metodológica recorrente nessa literatura: as métricas empregadas (variantes de *pass@k* e detecção CWE) mensuram a capacidade de alertar sobre artefatos vulneráveis, mas não avaliam simetricamente a especificidade, capacidade de *não* bloquear configurações legítimas. Em *policy-as-code* essa lacuna é operacionalmente relevante: uma política que bloqueia indiscriminadamente é tão inutilizável quanto uma que nada detecta, e o *over-blocking* é um dos fatores que levam organizações a flexibilizar ou desabilitar o *enforcement* automatizado.

Técnicas de *prompting* iterativo melhoram a geração de políticas OPA/Rego *utilizáveis* em ambiente de produção? E, se melhoram, a que custo? Responder exige um instrumento de medida que a literatura ainda não oferece, e construir esse instrumento é a contribuição do trabalho.

A contribuição central deste trabalho é uma **métrica pareada, a *strict_pass@1*, ancorada em uma avaliação por plano dual**: cada política gerada é confrontada ao mesmo tempo com o plano vulnerável, que mede detecção, e com um plano seguro espelhado, que mede *over-blocking*. Só conta como aprovada a política que dispara no plano vulnerável e permanece silenciosa no plano seguro. Essa é a definição operacional de política utilizável em ambiente de *enforcement* automatizado adotada aqui. Sob essa métrica, o RCI com *Compiler-in-the-Loop* eleva a taxa de aprovação na verificação estática e a detecção, mas degrada a especificidade em quatro dos cinco modelos avaliados, e leva a *strict_pass@1* do *gpt-4o* de 28,9% a 0,0%.

A medição se apoia em dois elementos: um conjunto de 30 planos Terraform vulneráveis em três níveis de complexidade, cada um com um *minimal pair*, que corrige apenas a vulnerabilidade-alvo, e 25 deles também com uma versão *Checkov-clean*, sem achados residuais, incluindo cenários de *Zero-Trust Architecture* (ZTA) com Twingate, MariaDB e GLPI, pouco explorados na literatura; e um protocolo com cinco LLMs, duas condições de *prompting* e $K = 3$ amostras independentes por célula — 900 execuções, com telemetria estruturada e análise pareada por cenário.

2. Trabalhos Relacionados

A literatura sobre uso de LLMs para geração de código de segurança e infraestrutura organiza-se em três frentes: engenharia de *prompt* direta, *fine-tuning* de domínio e sistemas multi-agente. A Tabela 1 sintetiza o posicionamento deste trabalho frente a elas.

Na primeira frente, [Bruni et al. 2025] reportam reduções de até 68,7% em vulnerabilidades com prefixos de segurança e RCI, avaliando supressão de CWE em código de aplicação, e [Tony et al. 2025] investigam sistematicamente técnicas de *prompting* para código seguro. [Zhang et al. 2025] propõem o IaCGen e o *benchmark* DPIaC-Eval, com 153 cenários e a métrica *passItr@n*, sem mensurar falsos positivos sobre infraestrutura legítima. Ainda em avaliação de *templates*, o IaC-Eval [Kon et al. 2024] estabelece 458 cenários AWS, com *pass@1* de 19,36% (GPT-4, *zero-shot*) a 36,70% com *retrieval-augmented generation*.

A segunda frente busca especialização por *fine-tuning*: o AutoPAC [Chowdhary et al. 2025] atinge 94% de validade em políticas RBAC/ABAC com *CodeT5-small*, mas opera sobre políticas de aplicação, não sobre planos de IaC; o GenSIaC [Li et al. 2025] aplica *instruction fine-tuning* sobre *CodeLlama* e atinge F1 =

0,771. A terceira explora arquiteturas multi-agente: o MACOG [Khan et al. 2025] alcança 74,02 no IaC-Eval, e o ARPaCCino [Romeo et al. 2025] emprega *Agentic-RAG* para validar políticas OPA em Proxmox, com 5/5 de corretude sob oráculo externo e n reduzido.

Duas observações delimitam a lacuna. Primeiro, gerar *templates* de IaC e gerar políticas que regulam IaC são tarefas distintas, e suas métricas não são diretamente comparáveis. Segundo, todos os trabalhos avaliam a capacidade de identificar vulnerabilidades, mas nenhum mede a capacidade de não bloquear infraestrutura legítima. É nessa lacuna que este trabalho se posiciona.

Tabela 1. Posicionamento frente aos trabalhos relacionados. "Plano dual" indica que cada artefato gerado é avaliado contra um par vulnerável/seguro espelhado.

Trabalho	Domínio	Abordagem	Plano dual	*Over-block.*	Métrica
[Bruni et al. 2025]	Cód. de aplicação	*Prompting* (RCI)	Não	Não	Redução de CWE
[Tony et al. 2025]	Cód. de aplicação	*Prompting* (revisão)	Não	Não	Vulns./CWE
[Zhang et al. 2025]	*Templates* IaC	Refino iterativo	Não	Não	*passItr@n*
[Kon et al. 2024]	*Templates* IaC	*Benchmark* + RAG	Não	Não	*pass@1*
[Chowdhary et al. 2025]	Políticas RBAC/ABAC	*Fine-tuning*	Não	Não	Validade (%)
[Li et al. 2025]	*Scripts* IaC	*Instr. tuning*	Não	Parcial[a]	F1
[Khan et al. 2025]	*Templates* IaC	Multi-agente	Não	Não	Escore IaC-Eval
[Romeo et al. 2025]	Políticas OPA	*Agentic-RAG*	Não	Não[b]	Corretude (5/5)
Este trab.	**Políticas OPA p/ IaC**	***Prompting* (ZS/RCI)**	**Sim**	**Sim**	***strict_pass@1***

[a] O F1 penaliza falsos positivos na classificação de trechos, e não o bloqueio de infraestrutura legítima por uma política. [b] Conformidade verificada por oráculo externo, sem medição sistemática de falsos positivos ($n = 5$).

3. Metodologia

Este trabalho apresenta um experimento controlado para comparar duas estratégias de engenharia de *prompt*, Zero-shot (ZS) e RCI, na geração automática de políticas OPA na sintaxe Rego v1 a partir de planos de execução Terraform. O desenho do experimento considera cinco modelos de linguagem, duas condições de *prompting*, trinta cenários de infraestrutura e três amostras independentes por célula, totalizando 900 execuções de telemetria.

3.1. Construção do conjunto de cenários

Foram definidos 30 cenários Terraform vulneráveis (T01–T30) em três níveis de complexidade: L1 (T01–T10), falhas *cloud-native* simples sobre S3 e EC2 com verificação de atributos isolados; L2 (T11–T20), interdependências entre *Security Groups*, políticas IAM com JSON aninhado e VPC; e L3 (T21–T30), cenários de ZTA sobre Twingate, MariaDB e GLPI, complementando *benchmarks* existentes, predominantemente focados em recursos AWS isolados.

Para avaliar os falsos positivos (*over-blocking*), cada cenário vulnerável possui um plano Terraform seguro correspondente e, em 25 dos 30, também um segundo; ambos re-

presentam a mesma infraestrutura em configuração segura e foram revisados manualmente. O primeiro, o *minimal pair*, corrige a vulnerabilidade-alvo do cenário e os atributos que dependem dela, sem tocar nos demais alertas do Checkov. Na maioria dos cenários isso é um único atributo alterado. Em três deles (T20, T27 e T28) a correção exige recursos auxiliares: habilitar o registro de fluxo de uma VPC, por exemplo, requer o grupo de *logs* e o papel IAM correspondentes. Nesses casos o par difere em até oito atributos. O segundo, o *Checkov-clean*, é uma versão em que o Checkov não encontra nenhum alerta remanescente; ele não foi construído para cinco cenários (T01, T21, T22, T24 e T26). A divisão permite medir o *over-blocking* de duas formas: contra correções pontuais (*minimal pair*) e contra códigos integralmente limpos (*Checkov-clean*). Dois cenários (T22 e T28) foram marcados como limitações semânticas, pois a vulnerabilidade não é detectável apenas pelo código Terraform; esses cenários são excluídos apenas do denominador da *fn_rate*, e não das demais métricas, para não penalizar o modelo por uma limitação da ferramenta.

3.2. Modelos de linguagem avaliados

Foram avaliados cinco modelos: *gpt-4o* e *o3-mini* (API OpenAI), *deepseek-reasoner* (API DeepSeek), *qwen3.6-plus* (API DashScope) e *qwen2.5-coder:7b*, executado localmente via Ollama — APIs comerciais, um modelo aberto de alto nível e um modelo local menor, cobrindo cenários de uso distintos. As versões exatas foram registradas nos dados da pesquisa. Para os modelos não-*reasoning* fixa-se `temperature=0`; o *o3-mini* e o *deepseek-reasoner* não admitem esse parâmetro, o que constitui um *confound* declarado, já que a variância entre suas amostras tem origem qualitativamente distinta. O *o3-mini* foi ainda executado com `reasoning_effort="high"`, sem equivalente no *deepseek-reasoner*, o que se soma a esse *confound* na leitura dos resultados desses dois modelos. Não se usa o parâmetro `seed`, cuja implementação é *best-effort*; a mitigação adotada é a coleta de $K = 3$ amostras independentes.

3.3. Condições experimentais

A condição **ZS** consiste em uma única chamada: uma mensagem *system* com as regras obrigatórias de sintaxe Rego v1 e uma mensagem *user* com o plano Terraform. Para L2 e L3, anexa-se um *template* estrutural canônico de regra *deny* — referência sintática não-resolutiva, sem exemplo *input-output* completo — para mitigar erros básicos de sintaxe. Os cenários L1 não recebem esse *template*. A ressalva importa para a leitura dos resultados desagregados por nível (Seção 4.5).

A condição **RCI** tem três fases sequenciais com manutenção do histórico conversacional. A Fase 1 (*Critique*) submete o plano vulnerável ao modelo sob um *system prompt* de auditor de segurança, requisitando análise textual numerada das vulnerabilidades, sem código. A Fase 2 (*Generation*) reincorpora a crítica como contexto e solicita a política Rego inicial, retornando ao *system prompt* base. A Fase 3 (*Compiler-in-the-Loop*) — designação mantida por consistência com a literatura de retroalimentação iterativa a partir de verificadores automáticos [Zhang et al. 2025] — executa `opa check` sobre a política gerada, verificação estática de sintaxe e tipos do Rego, e não compilação no sentido tradicional; em caso de falha, o erro é reenviado ao modelo em laço limitado a cinco tentativas. O laço conhece exclusivamente erros de verificação estática: não avalia detecção semântica nem *over-blocking*.

3.4. Métricas e análise estatística

Além da *strict_pass@1*, definida a seguir, cinco métricas são registradas por execução. A *compile_pass@1* mede a fração de amostras cuja política é aprovada na verificação estática

(`opa check`) sob OPA v1.11; o identificador é preservado por compatibilidade com a telemetria. A *eval_pass@1* mede a fração que é aprovada e detecta a vulnerabilidade no plano vulnerável. As taxas *adj-block* e *clean-block* medem o bloqueio dos planos *minimal pair* e *Checkov-clean*, calculadas sobre as execuções em que o plano era avaliável, e a *fn_rate* mede a falha em detectar a vulnerabilidade. Todas as proporções são acompanhadas de intervalos de confiança Wilson de 95%.

A *strict_pass@1* mede a fração que é aprovada na verificação, detecta a vulnerabilidade e, ao mesmo tempo, não dispara contra o *minimal pair*, isto é, contra o plano seguro que corrige apenas a vulnerabilidade-alvo e mantém o resto da estrutura do cenário. A escolha do *minimal pair* como referência é deliberada e decorre de sua propriedade discriminativa: como difere do plano vulnerável em um único atributo na maioria dos cenários, ele isola a capacidade da política de distinguir o desvio pretendido de configurações estruturalmente quase idênticas. O predicado exige que o atributo de bloqueio seja explicitamente falso, e não apenas ausente; execuções em que o par seguro não pôde ser avaliado não contam como aprovadas, o que é uma leitura conservadora. O bloqueio sobre o plano *Checkov-clean* é registrado em paralelo, como diagnóstico independente, e fica fora do predicado da métrica.[1]

A comparação entre ZS e RCI nas métricas *eval_pass@1* e *strict_pass@1* usa o teste exato de McNemar, com o g de Cohen como medida de tamanho de efeito. **A unidade de análise dos testes pareados é o cenário**: as $K = 3$ amostras de cada célula são colapsadas por maioria (≥ 2 de 3) antes do pareamento, o que elimina a dependência intra-cenário e deixa $n = 30$ pares por modelo. A regra de colapso foi fixada antes da execução dos testes, e regras alternativas (*ao menos uma* e *todas as três*) aparecem como análise de sensibilidade. Dentro de cada família de cinco comparações aplica-se ainda a correção de Holm. As taxas descritivas da Tabela 2 permanecem por execução ($n = 90$).

3.5. Reprodutibilidade

O experimento foi desenvolvido em Python, com módulos para execução, validação via OPA e integração com as APIs, e telemetria por execução em JSONL. Para avaliar os falsos positivos sem consumir créditos adicionais, um *script* reutiliza as políticas já persistidas e as reavalia contra os dois planos seguros; isso é viável porque o plano seguro participa apenas da avaliação final, não da geração. Todo o código, os cenários, as políticas geradas e os dados estão em `https://github.com/rodriguesedub/WTICG-2026-IaC-LLM-Final-Evaluation`. O repositório traz ainda um *script* de auditoria que recomputa, a partir da telemetria bruta, as taxas da Tabela 2 e os testes pareados reportados neste artigo.

4. Resultados

A Tabela 2 traz as taxas descritivas por modelo e condição, na unidade execução ($n = 90$ por célula, IC de Wilson 95%); nos testes pareados das Seções 4.2 e 4.3, a unidade é o cenário.

4.1. RCI eleva a taxa de aprovação na verificação estática

A taxa de aprovação na verificação estática aumenta sob RCI nos cinco modelos, tanto por execução quanto por cenário. Por execução, o ganho é notório no *qwen2.5-coder:7b* (33,3% $\rightarrow$ 52,2%), no *deepseek-reasoner* (85,6% $\rightarrow$ 95,6%) e no *o3-mini* (82,2% $\rightarrow$

[1] O predicado sobre o *minimal pair* está fixado na implementação da análise desde a coleta dos dados, antes de qualquer teste estatístico; o histórico do repositório permite conferir isso.

Tabela 2. Taxas por modelo e condição (unidade = execução, $n = 90$). ZS = Zero-shot; RCI = *Recursive Criticism and Improvement*. Só execuções avaliáveis — $n = 737$ em *adj-block* (900 menos 163 reprovações estáticas) e $n = 605$ em *clean-block* (menos 132 dos cinco cenários sem plano *Checkov-clean*).

Modelo	Cond.	compile@1	eval@1	strict@1	FN-rate	adj-block	clean-block
deepseek-reasoner	ZS	85,6%	83,3%	22,2%	2,7%	71,4%	72,3%
deepseek-reasoner	RCI	95,6%	93,3%	11,1%	2,5%	86,0%	84,7%
gpt-4o	ZS	88,9%	85,6%	28,9%	4,1%	63,7%	71,2%
gpt-4o	RCI	92,2%	92,2%	**0,0%**	0,0%	**100,0%**	95,7%
o3-mini	ZS	82,2%	82,2%	16,7%	0,0%	79,7%	80,0%
o3-mini	RCI	94,4%	93,3%	22,2%	1,3%	75,3%	67,1%
qwen2.5-coder:7b	ZS	33,3%	21,1%	10,0%	40,7%	43,3%	22,2%
qwen2.5-coder:7b	RCI	52,2%	24,4%	4,4%	55,6%	38,3%	34,2%
qwen3.6-plus	ZS	95,6%	95,6%	25,6%	0,0%	73,3%	68,1%
qwen3.6-plus	RCI	98,9%	98,9%	7,8%	0,0%	92,1%	85,1%

94,4%); modelos já próximos do teto sob ZS, como o *qwen3.6-plus* e o *gpt-4o*, exibem ganhos modestos, compatíveis com saturação. Tomando o cenário como unidade, nenhuma dessas diferenças atinge significância ($p \geq 0{,}30$), pelo problema de potência discutido na Seção 4.2. A maior parte dos erros é resolvida nas duas primeiras tentativas de correção; a taxa cumulativa por iteração consta do Anexo B (Tabela 7).

4.2. Ganhos na detecção de falhas

A Tabela 3 traz o teste exato de McNemar sobre *eval_pass@1*, com o cenário como unidade. A direção do efeito é consistente: o RCI aumenta a detecção nos cinco modelos e, em quatro deles, não há um único cenário em que o ZS detecte e o RCI falhe ($b = 0$). Nenhuma das comparações, porém, atinge significância ($p \geq 0{,}25$).

O motivo é de potência, e não de ausência de efeito. No teste exato, o menor p alcançável é $2 \times 0{,}5^{b+c}$, de modo que com menos de seis pares discordantes não há como rejeitar a hipótese nula a 5%. Em quatro modelos $b + c \leq 3$, e o teste está saturado nesse limite. Daí a leitura conservadora adotada aqui: o ganho de detecção sob RCI é *direcional* em todos os modelos e não é confirmado estatisticamente no nível de cenário. A análise por execução trata as 90 amostras como independentes e indica ganho significativo no *o3-mini* ($\Delta = +11{,}1$pp, $p = 0{,}0063$) e, de forma marginal, no *deepseek-reasoner* ($p = 0{,}0490$); ela está no Anexo B e deve ser lida com a ressalva da Seção 6.

Tabela 3. Teste exato de McNemar sobre *eval_pass@1* por cenário ($n = 30$ pares; $K = 3$ amostras colapsadas por maioria). b = ZS acerta e RCI falha; c = ZS falha e RCI acerta. p_{Holm} corrige as cinco comparações da família.

Modelo	ZS	RCI	Δ	b	c	p	p_{Holm}	g
deepseek-reasoner	90,0%	100,0%	+10,0pp	0	3	0,2500	1,0000	+0,500
gpt-4o	86,7%	96,7%	+10,0pp	0	3	0,2500	1,0000	+0,500
o3-mini	90,0%	96,7%	+6,7pp	0	2	0,5000	1,0000	+0,500
qwen2.5-coder:7b	20,0%	23,3%	+3,3pp	3	4	1,0000	1,0000	+0,071
qwen3.6-plus	96,7%	100,0%	+3,3pp	0	1	1,0000	1,0000	+0,500

4.3. A métrica estrita revela degradação sob RCI

A Tabela 4 repete a análise sobre a *strict_pass@1*, que exige aprovação na verificação estática, detecção da vulnerabilidade no plano vulnerável e ausência de bloqueio sobre

o *minimal pair*.

O resultado do *gpt-4o* é conclusivo e sobrevive a todas as variantes de colapso: a métrica cai de 30,0% para 0,0% ($b = 9$, $c = 0$, $p = 0{,}0039$; $p_{Holm} = 0{,}0195$; $g = -0{,}50$). Nenhum dos 30 cenários produziu, sob RCI, uma política que detectasse a vulnerabilidade sem bloquear o plano seguro correspondente.[2] O *qwen3.6-plus* cai de 23,3% para 6,7%, com $b = 5$ e $c = 0$; o valor $p = 0{,}0625$ é o menor alcançável para cinco pares discordantes, e o resultado é aqui classificado como direcional. O *deepseek-reasoner* (20,0% $\rightarrow$ 10,0%; $b = 3$, $c = 0$) e o *qwen2.5-coder:7b* (10,0% $\rightarrow$ 3,3%; $b = 3$, $c = 1$) declinam sem significância. O *o3-mini* permanece a única exceção, com variação positiva não significativa (16,7% $\rightarrow$ 23,3%).

O padrão das células discordantes é o ponto principal: em três dos quatro modelos que declinam, $c = 0$, ou seja, o RCI não recuperou a especificidade em nenhum cenário que o ZS já resolvia. O efeito é unidirecional, e é essa assimetria, mais do que a magnitude de cada queda isolada, que sustenta a interpretação da Seção 5. Vale notar que, no teste de McNemar, g fica limitado ao intervalo $[-0{,}5, +0{,}5]$ e satura em $\pm 0{,}5$ sempre que uma das células discordantes é nula. Os valores de $g = -0{,}50$ observados aqui decorrem de $c = 0$ e devem ser lidos junto com as contagens brutas.

Como análise de sensibilidade, trocou-se o *minimal pair* pelo plano *Checkov-clean* no predicado. A direção do efeito se mantém em quatro dos cinco modelos, com magnitudes menores (*gpt-4o*: 20,0% $\rightarrow$ 3,3%, $p = 0{,}0625$), o que é coerente com a natureza mais discriminativa do par mínimo. As regras alternativas de colapso também preservam o achado do *gpt-4o* (*ao menos uma*: $p = 0{,}0039$; *todas as três*: $p = 0{,}0078$).

A análise pareada por cenário muda a força das evidências, mas não a direção delas: a degradação de especificidade continua conclusiva no *gpt-4o* e direcional nos demais modelos que declinam, enquanto os ganhos de detecção, que a análise por execução sugeria significativos, deixam de sê-lo.

Tabela 4. Teste exato de McNemar sobre *strict_pass@1* por cenário ($n = 30$ pares; $K = 3$ amostras colapsadas por maioria).

Modelo	ZS	RCI	Δ	b	c	p	p_{Holm}	g
deepseek-reasoner	20,0%	10,0%	−10,0pp	3	0	0,2500	0,7500	−0,500
gpt-4o	30,0%	**0,0%**	−30,0pp	9	0	**0,0039**	**0,0195**	−0,500
o3-mini	16,7%	23,3%	+6,7pp	2	4	0,6875	1,0000	+0,167
qwen2.5-coder:7b	10,0%	3,3%	−6,7pp	3	1	0,6250	1,0000	−0,250
qwen3.6-plus	23,3%	6,7%	−16,7pp	5	0	0,0625	0,2500	−0,500

4.4. Consistência entre os dois critérios de plano seguro

As taxas *adj-block* e *clean-block* diferem em menos de 9 pontos percentuais em nove das dez células (Tabela 2); a exceção é o *qwen2.5-coder:7b* sob ZS (43,3% *vs.* 22,2%), célula com denominadores reduzidos pela baixa aprovação estática. A proximidade nas demais sustenta que a maior parte dos disparos sobre planos seguros é *over-blocking* em sentido estrito, e não detecção colateral de vulnerabilidades residuais. O caso mais extremo continua sendo o *gpt-4o* sob RCI: suas políticas bloqueiam todos os planos *minimal pair* avaliados (*adj-block* = 100,0%) e 95,7% dos planos *Checkov-clean*.

[2]Os 30,0% do ZS correspondem à unidade cenário; por execução, a mesma célula rende 28,9% (Tabela 2). A diferença vem só da regra de colapso, e não de dados distintos.

4.5. Desempenho por nível de complexidade

A Tabela 8, no Anexo B, desagrega as métricas pelos três níveis do conjunto de cenários. Agregando os cinco modelos, a *strict_pass@1* sob ZS é de 3,3% em L1, 43,3% em L2 e 15,3% em L3; sob RCI, passa a 4,7%, 19,3% e 3,3%, respectivamente. A degradação induzida pelo RCI concentra-se, portanto, nos níveis L2 e L3, enquanto L1 exibe um piso de especificidade comum às duas condições. No *gpt-4o*, a *strict_pass@1* sob RCI é nula nos três níveis (Anexo B), o que mostra que o efeito não se restringe a uma faixa de complexidade.

Duas ressalvas delimitam a leitura desses números. A primeira: os níveis diferem em complexidade e também no *prompt*, já que, conforme a Seção 3.3, o *template* estrutural canônico de regra *deny* é anexado apenas em L2 e L3. O gradiente crescente de aprovação na verificação estática entre L1 e L3 (64,7% $\rightarrow$ 88,0% sob ZS) confunde, assim, complexidade do cenário e presença do *template*, e fica particularmente visível no modelo local, cuja aprovação sob ZS vai de 0/30 em L1 a 18/30 em L3. A segunda: os contrastes entre níveis são descritivos, sem testes de hipótese, uma vez que cada estrato reúne apenas dez cenários. As comparações entre ZS e RCI reportadas nas seções anteriores são internas a cada nível e a cada modelo, e essas ressalvas não as afetam.

5. Discussão

O resultado mais relevante da pesquisa é a degradação em *strict_pass@1*, e o caso do *gpt-4o* é o mais nítido: o modelo passa a ser aprovado na verificação estática e a detectar em cerca de 92% das execuções sob RCI, e ainda assim a fração de políticas que detectam a vulnerabilidade-alvo sem bloquear o plano seguro cai a zero. Uma explicação decorre da própria estrutura do RCI implementado. A Fase 3 (*Compiler-in-the-Loop*) executa um laço de reparo cujo único critério de sucesso é a aprovação na verificação estática; o laço não tem acesso ao plano seguro nem a qualquer sinal sobre *over-blocking*. Em modelos que buscam essa aprovação por meio de regras de maior cobertura, generalizando além do estritamente necessário, o resultado natural é uma política que é aprovada e detecta, mas que também dispara contra recursos legítimos com estrutura semelhante à do cenário vulnerável.

O *o3-mini*, único modelo cuja *strict_pass@1* não decai sob RCI, sugere a hipótese de que modelos de raciocínio integrariam considerações de especificidade já na Fase 2. A hipótese não se generaliza: o *deepseek-reasoner*, também de raciocínio, exibe degradação direcional, de modo que o padrão não corresponde a um corte limpo entre modelos *reasoning* e *non-reasoning*.

O Anexo A mostra esse mecanismo em um caso concreto. No cenário T03, cuja vulnerabilidade-alvo é a desativação do versionamento de um *bucket* S3, o *gpt-4o* sob ZS produz uma política conceitualmente correta, com duas regras *deny* dirigidas exclusivamente ao versionamento, que acaba reprovada na verificação estática por um erro de tipo: um predicado definido como regra completa e invocado como função. Sob RCI, o laço corrige a sintaxe, mas a política resultante enumera nove regras, que passam a exigir *tags*, criptografia, política de *bucket*, *logging*, *MFA delete* e ainda atributos do *provider*. O alvo continua detectado; o que muda é o escopo. Aplicada ao plano seguro, essa política o bloqueia por regras que nada têm a ver com a vulnerabilidade avaliada. O caso indica que a perda de especificidade não vem do reparo sintático em si: a Fase 3 apenas corrige a sintaxe, e a enumeração abrangente de controles é induzida pela crítica da Fase 1.

A implicação metodológica é direta: métricas de *pass@1* isoladas, sem mensuração simétrica de *over-blocking*, não caracterizam a utilidade operacional do código gerado em tarefas de *policy-as-code*, e critérios como a *strict_pass@1* parecem necessários para

que avaliações reflitam as restrições de produção. A implicação aplicada é a extensão *RCI-dual* (Seção 7): um laço de reparo que considere também os bloqueios indevidos sobre planos legítimos.

6. Ameaças à Validade

Destacam-se cinco ameaças à validade deste experimento.

A primeira é a sobreposição de variáveis sob o rótulo único "RCI", que agrupa a Fase 1 (*Critique*) e a Fase 3 (*Compiler-in-the-Loop*). O desenho atual não permite afirmar se os ganhos em *eval_pass@1* vieram da crítica de segurança ou do laço de correção sintática, pois as duas fases ocorrem juntas; isolá-las exigiria condições separadas. O achado principal, porém, permanece válido: a queda na métrica estrita mostra que essas fases, em conjunto, degradam a especificidade, independentemente de qual delas a cause.

A segunda é o fator de confusão dos modelos de raciocínio (*o3-mini* e *deepseek-reasoner*), que não permitem fixar `temperature` em 0, de modo que a variação de suas respostas tem origem distinta da dos demais. A observação de que o *o3-mini* foi o único sem piora na *strict_pass@1* pode ser característica de modelos de raciocínio ou apenas efeito desse fator; confirmá-lo exigiria comparar outros modelos da mesma família.

A terceira diz respeito ao uso do Checkov no *pipeline*. A implementação prevê injetar, no *prompt* da Fase 1, a contagem de achados do Checkov para o cenário. A verificação da telemetria confirma, contudo, que essa contagem esteve indisponível nas 900 execuções, e o bloco não foi incorporado a nenhum *prompt*. As duas condições receberam, portanto, exatamente a mesma informação sobre o plano, e nenhuma assimetria de contexto afeta a comparação. O Checkov segue em uso na construção dos cenários e do plano *Checkov-clean*; avaliar o efeito de fornecer essa contagem, às duas condições ou a nenhuma, integra a agenda futura.

A quarta é de natureza estatística: a dependência entre as $K = 3$ amostras de cada cenário. O tratamento é a análise pareada por cenário, com as amostras colapsadas por maioria, regras alternativas como sensibilidade e correção de Holm em cada família de comparações. O custo é potência: os pares caem de 90 para 30, o que explica a perda de significância dos ganhos de detecção (Seção 4.2); uma reanálise por efeitos mistos, com o cenário como agrupamento, aproveitaria as 900 execuções sem supor independência e integra a agenda futura.

A quinta é um limite de validade externa. O conjunto reúne $N = 30$ cenários, restrição estrutural: a avaliação por plano dual exige, para cada cenário vulnerável, uma variante segura espelhada, construída e revisada manualmente. Os *benchmarks* maiores (IaC-Eval [Kon et al. 2024]; DPIaC-Eval [Zhang et al. 2025]) avaliam geração de *templates*, tarefa distinta, sem pares mínimos; incorporar TerraGoat ou KaiMonkey exigiria a mesma construção manual. Somam-se o foco em recursos AWS e arquitetura ZTA, a restrição à linguagem Rego v1 e a confusão entre nível e conteúdo do *prompt* (Seção 4.5).

7. Conclusão e Trabalhos Futuros

Este trabalho apresenta um *benchmark* controlado de geração de políticas OPA/Rego para IaC via LLMs, comparando ZS e RCI com *Compiler-in-the-Loop*. A contribuição principal é a métrica *strict_pass@1*, ancorada em avaliação por plano dual, e o *trade-off* que ela deixa ver: o RCI eleva a aprovação na verificação estática e melhora direcionalmente a detecção em todos os modelos, mas degrada a métrica que penaliza *over-blocking* em quatro dos cinco avaliados — de forma conclusiva no *gpt-4o* ($p = 0{,}0039$; $p_{Holm} = 0{,}0195$) e direcional nos demais. O achado sustenta a recomendação de métricas pareadas para

avaliação de LLMs em *policy-as-code*. Seis direções decorrem do estudo. A primeira, e mais imediata, é decompor o RCI: condições que isolem a crítica de segurança (Fase 1) do laço de reparo (Fase 3) permitiriam atribuir a cada componente a origem dos ganhos e das perdas, hoje somados sob um rótulo único; a evidência qualitativa da Seção 5 sugere a Fase 1 como principal candidata. Completam a agenda a extensão *RCI-dual* — um critério de reparo que verifique, além da aprovação estática, o não-bloqueio de planos legítimos —, *datasets* externos, validação dinâmica em AWS, RAG e *LLM-as-Judge* com dois juízes de famílias distintas [Panickssery et al. 2024].

Uso de Inteligência Artificial Generativa

Os autores utilizaram ferramentas de IA generativa (Claude, da Anthropic, e Gemini, do Google) para correção gramatical e revisão de redação, geração e reescrita de trechos de texto, formatação de referências e tabelas, e apoio na construção de *scripts* de análise estatística. Nenhuma ferramenta de IA é listada como autora; os autores revisaram e validaram todo o conteúdo gerado e assumem integral responsabilidade por ele.

Anexo A – Exemplo de *over-blocking* (cenário T03)

O cenário T03 ilustra de forma concreta o *trade-off* caracterizado na Seção 5. A vulnerabilidade-alvo é a desativação do versionamento de um *bucket* S3 — o atributo `status` do recurso `aws_s3_bucket_versioning` igual a `Suspended`. As Listagens 1 e 2 reproduzem as políticas geradas pelo *gpt-4o* para este cenário (amostra $k = 0$) sob ZS e sob RCI, respectivamente.

Sob ZS (Listagem 1), o modelo produz uma política específica: duas regras *deny* voltadas exclusivamente ao versionamento, auxiliadas por um predicado `bucket_versioning_enabled`. A política não é aprovada na verificação estática: o predicado é definido como regra completa (sem parâmetros) e invocado como função com argumento, e o verificador `opa check` o rejeita com erro de tipo (`rego_type_error`: *undefined function*, linha 7). A amostra falha em *strict_pass@1* por razão sintática, não conceitual: o modelo havia identificado corretamente o escopo do problema.

```
package terraform.policies
import rego.v1

deny contains msg if {
    some resource in input.resource_changes
    resource.type == "aws_s3_bucket"
    not bucket_versioning_enabled(resource)
    msg := sprintf("S3 bucket '%s' does not have versioning enabled.",
   [resource.address])
}

deny contains msg if {
    some resource in input.resource_changes
    resource.type == "aws_s3_bucket_versioning"
    versioning_status := object.get(resource.change.after, "
   versioning_configuration", [])
    versioning_status[0].status == "Suspended"
    msg := sprintf("S3 bucket versioning for '%s' is suspended.", [
   resource.address])
}

bucket_versioning_enabled if {
```

```
    some resource in input.resource_changes
    resource.type == "aws_s3_bucket_versioning"
    versioning_status := object.get(resource.change.after, "
   versioning_configuration", [])
    versioning_status[0].status == "Enabled"
}
```

Listing 1. T03 sob ZS: política específica ao versionamento, reprovada na verificação estática (o predicado é definido como regra, mas invocado como função).

Sob RCI (Listagem 2), o *loop* de reparo corrige a sintaxe, mas a política resultante enumera nove regras *deny* que extrapolam a vulnerabilidade-alvo: além do versionamento, passa a exigir *tags*, criptografia *server-side*, política de *bucket*, *logging*, *MFA delete* e ainda regras sobre credenciais e *flags* do *provider* AWS. A política é aprovada na verificação estática e detecta corretamente o versionamento suspenso no plano vulnerável; porém, suas regras auxiliares bloqueiam qualquer *bucket* legítimo que não configure simultaneamente todos esses controles. Aplicada ao plano seguro espelhado, no qual o versionamento está habilitado, a política o bloqueia, pois o plano não define *tags*, criptografia, *logging* e demais atributos exigidos pelas regras excedentes. A amostra falha em *strict_pass@1* por *over-blocking*.

```
package terraform.policies
import rego.v1

# Deny if S3 bucket versioning is not enabled
deny contains msg if {
    some resource in input.resource_changes
    resource.type == "aws_s3_bucket_versioning"
    status := object.get(resource.change.after, "
   versioning_configuration", [{}])[0].status
    status != "Enabled"
    msg := sprintf("S3 bucket versioning is not enabled for resource %s
   ", [resource.address])
}

# Deny if S3 bucket tags are not set
deny contains msg if {
    some resource in input.resource_changes
    resource.type == "aws_s3_bucket"
    tags := object.get(resource.change.after, "tags", null)
    tags == null
    msg := sprintf("S3 bucket tags are not set for resource %s", [
   resource.address])
}

# Deny if S3 bucket server-side encryption is not configured
deny contains msg if {
    some resource in input.resource_changes
    resource.type == "aws_s3_bucket"
    sse_config := object.get(resource.change.after, "
   server_side_encryption_configuration", null)
    sse_config == null
    msg := sprintf("S3 bucket server-side encryption is not configured
   for resource %s", [resource.address])
```

```
}

# Deny if S3 bucket policy is not set
deny contains msg if {
    some resource in input.resource_changes
    resource.type == "aws_s3_bucket"
    policy := object.get(resource.change.after, "policy", null)
    policy == null
    msg := sprintf("S3 bucket policy is not set for resource %s", [
    resource.address])
}

# Deny if S3 bucket logging is not configured
deny contains msg if {
    some resource in input.resource_changes
    resource.type == "aws_s3_bucket"
    logging := object.get(resource.change.after, "logging", null)
    logging == null
    msg := sprintf("S3 bucket logging is not configured for resource %s
    ", [resource.address])
}

# Deny if S3 bucket MFA delete is not enabled
deny contains msg if {
    some resource in input.resource_changes
    resource.type == "aws_s3_bucket_versioning"
    mfa_delete := object.get(resource.change.after.
    versioning_configuration[0], "mfa_delete", null)
    mfa_delete != "Enabled"
    msg := sprintf("S3 bucket MFA delete is not enabled for resource %s
    ", [resource.address])
}

# Deny if AWS provider credentials are hardcoded
deny contains msg if {
    access_key := object.get(input.configuration.provider_config.aws.
    expressions, "access_key", {}).constant_value
    secret_key := object.get(input.configuration.provider_config.aws.
    expressions, "secret_key", {}).constant_value
    access_key != null
    secret_key != null
    msg := "AWS provider credentials are hardcoded"
}

# Deny if AWS provider skip metadata API check is enabled
deny contains msg if {
    skip_metadata_api_check := object.get(input.configuration.
    provider_config.aws.expressions, "skip_metadata_api_check", {}).
    constant_value
    skip_metadata_api_check == true
    msg := "AWS provider skip metadata API check is enabled"
}

# Deny if AWS provider skip credentials validation is enabled
deny contains msg if {
    skip_credentials_validation := object.get(input.configuration.
```

```
    provider_config.aws.expressions, "skip_credentials_validation", {}).
    constant_value
     skip_credentials_validation == true
     msg := "AWS provider skip credentials validation is enabled"
}
```

Listing 2. T03 sob RCI: a política é aprovada na verificação estática e detecta o alvo, mas nove regras *deny* extrapolam a vulnerabilidade e bloqueiam também o plano seguro.

O caso sugere que a amplitude excessiva da política RCI não decorre do reparo sintático em si. A Fase 3 apenas corrige a sintaxe; a enumeração abrangente de controles de segurança é induzida pela crítica da Fase 1. A observação é consistente com a ameaça de sobreposição de variáveis registrada na Seção 6: o ganho de detecção e a degradação de especificidade, reunidos sob o rótulo único "RCI", têm origens distintas e não separáveis no desenho experimental atual.

Anexo B – Análises complementares

Testes pareados por execução. As Tabelas 5 e 6 reproduzem os testes de McNemar tomando a execução como unidade ($n = 90$ pares), pareada por cenário $\times$ *sample_idx*. Esta análise trata as $K = 3$ amostras de um mesmo cenário como observações independentes e, por isso, produz p-valores otimistas (Seção 6); é reportada apenas para comparabilidade com a literatura que adota essa unidade.

Tabela 5. McNemar sobre *eval_pass@1* por execução ($n = 90$ pares por modelo).

Modelo	ZS	RCI	Δ	b	c	p	g
deepseek-reasoner	83,3%	93,3%	+10,0pp	4	13	0,0490	+0,265
gpt-4o	85,6%	92,2%	+6,7pp	6	12	0,2379	+0,167
o3-mini	82,2%	93,3%	+11,1pp	1	11	0,0063	+0,417
qwen2.5-coder:7b	21,1%	24,4%	+3,3pp	10	13	0,6776	+0,065
qwen3.6-plus	95,6%	98,9%	+3,3pp	1	4	0,3750	+0,300

Tabela 6. McNemar sobre *strict_pass@1* por execução ($n = 90$ pares por modelo).

Modelo	ZS	RCI	Δ	b	c	p	g
deepseek-reasoner	22,2%	11,1%	−11,1pp	12	2	0,0129	−0,357
gpt-4o	28,9%	0,0%	−28,9pp	26	0	$< 10^{-6}$	−0,500
o3-mini	16,7%	22,2%	+5,6pp	7	12	0,3593	+0,132
qwen2.5-coder:7b	10,0%	4,4%	−5,6pp	8	3	0,2266	−0,227
qwen3.6-plus	25,6%	7,8%	−17,8pp	16	0	$< 10^{-4}$	−0,500

Refinamento iterativo e estratificação por nível. A Tabela 7 traz a taxa de sucesso cumulativa por iteração do *Compiler-in-the-Loop*, discutida na Seção 4, e a Tabela 8 agrega as métricas por nível de complexidade, discutidas na Seção 4.5.

Desagregação por modelo e nível. A Tabela 9 detalha, por modelo, condição e nível, as métricas agregadas na Tabela 8. Valem as duas ressalvas da Seção 4.5.

Tabela 7. Taxa de sucesso cumulativa por iteração do *Compiler-in-the-Loop* (condição RCI, $n = 90$).

Modelo	itr≤1	itr≤2	itr≤3	itr≤4	itr≤5
deepseek-reasoner	55,6%	77,8%	85,6%	93,3%	95,6%
gpt-4o	72,2%	90,0%	92,2%	92,2%	92,2%
o3-mini	84,4%	92,2%	93,3%	93,3%	94,4%
qwen2.5-coder:7b	23,3%	47,8%	50,0%	52,2%	52,2%
qwen3.6-plus	82,2%	94,4%	96,7%	97,8%	98,9%

Tabela 8. Métricas por nível de complexidade, agregando os cinco modelos (unidade = execução; $n = 150$ por célula). Leitura descritiva: a agregação reúne modelos de capacidades distintas. IC de Wilson 95%. O desdobramento por modelo consta da Tabela 9.

Nível	Cond.	n	compile@1	eval@1	strict@1
L1	ZS	150	64,7% [56,7–71,9]	64,7% [56,7–71,9]	3,3% [1,4–7,6]
L1	RCI	150	89,3% [83,4–93,3]	74,7% [67,2–81,0]	4,7% [2,3–9,3]
L2	ZS	150	78,7% [71,4–84,5]	74,0% [66,4–80,4]	43,3% [35,7–51,3]
L2	RCI	150	83,3% [76,6–88,4]	81,3% [74,3–86,8]	19,3% [13,8–26,4]
L3	ZS	150	88,0% [81,8–92,3]	82,0% [75,1–87,3]	15,3% [10,4–22,0]
L3	RCI	150	87,3% [81,1–91,7]	85,3% [78,8–90,1]	3,3% [1,4–7,6]

Referências

Bruni, M., Gabrielli, F., Ghafari, M., and Kropp, M. (2025). Benchmarking prompt engineering techniques for secure code generation with GPT models. arXiv preprint arXiv:2502.06039.

Chowdhary, N., Dutta, T., Chattopadhyay, S., and Chakraborty, S. (2025). AutoPAC: Exploring LLMs for automating policy to code conversion in business organizations. In *2025 17th International Conference on COMmunication Systems and NETworks (COMSNETS)*. IEEE.

Khan, R. N. H., Wasif, D., Cho, J.-H., and Butt, A. (2025). Multi-agent code-orchestrated generation for reliable infrastructure-as-code. arXiv preprint arXiv:2510.03902v1.

Kon, P. T. J., Liu, J., Qiu, Y., Fan, W., He, T., Lin, L., Zhang, H., Park, O. M., Elengikal, G. S., Kang, Y., Chen, A., Chowdhury, M., Lee, M., and Wang, X. (2024). IaC-Eval: A code generation benchmark for cloud infrastructure-as-code programs. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*.

Li, Y., Grella, M., Nahmias, D., Engelberg, G., Klein, D., Guizzardi, G., van Ede, T., and Continella, A. (2025). GENSIAC: Toward security-aware infrastructure-as-code generation with large language models. arXiv preprint arXiv:2511.12385.

Panickssery, A., Bowman, S. R., and Feng, S. (2024). LLM evaluators recognize and favor their own generations. arXiv preprint arXiv:2404.13076.

Romeo, F., Arena, L., Blefari, F., Pironti, F. A., Lupinacci, M., and Furfaro, A. (2025). ARPACCINO: An agentic-RAG for policy as code compliance. arXiv preprint arXiv:2507.10584v2.

Tabela 9. Métricas por modelo, condição e nível de complexidade (unidade = execução; $n = 30$ por célula, correspondendo a 10 cenários $\times$ $K = 3$). IC de Wilson 95%.

Modelo	Cond.	Nível	compile@1	eval@1	strict@1
deepseek-reasoner	ZS	L1	80,0% [62,7–90,5]	80,0% [62,7–90,5]	0,0% [0,0–11,4]
deepseek-reasoner	ZS	L2	86,7% [70,3–94,7]	80,0% [62,7–90,5]	53,3% [36,1–69,8]
deepseek-reasoner	ZS	L3	90,0% [74,4–96,5]	90,0% [74,4–96,5]	13,3% [5,3–29,7]
deepseek-reasoner	RCI	L1	93,3% [78,7–98,2]	90,0% [74,4–96,5]	0,0% [0,0–11,4]
deepseek-reasoner	RCI	L2	100,0% [88,6–100,0]	96,7% [83,3–99,4]	30,0% [16,7–47,9]
deepseek-reasoner	RCI	L3	93,3% [78,7–98,2]	93,3% [78,7–98,2]	3,3% [0,6–16,7]
gpt-4o	ZS	L1	70,0% [52,1–83,3]	70,0% [52,1–83,3]	6,7% [1,8–21,3]
gpt-4o	ZS	L2	96,7% [83,3–99,4]	96,7% [83,3–99,4]	60,0% [42,3–75,4]
gpt-4o	ZS	L3	100,0% [88,6–100,0]	90,0% [74,4–96,5]	20,0% [9,5–37,3]
gpt-4o	RCI	L1	86,7% [70,3–94,7]	86,7% [70,3–94,7]	0,0% [0,0–11,4]
gpt-4o	RCI	L2	100,0% [88,6–100,0]	100,0% [88,6–100,0]	0,0% [0,0–11,4]
gpt-4o	RCI	L3	90,0% [74,4–96,5]	90,0% [74,4–96,5]	0,0% [0,0–11,4]
o3-mini	ZS	L1	73,3% [55,6–85,8]	73,3% [55,6–85,8]	6,7% [1,8–21,3]
o3-mini	ZS	L2	80,0% [62,7–90,5]	80,0% [62,7–90,5]	30,0% [16,7–47,9]
o3-mini	ZS	L3	93,3% [78,7–98,2]	93,3% [78,7–98,2]	13,3% [5,3–29,7]
o3-mini	RCI	L1	100,0% [88,6–100,0]	96,7% [83,3–99,4]	23,3% [11,8–40,9]
o3-mini	RCI	L2	83,3% [66,4–92,7]	83,3% [66,4–92,7]	30,0% [16,7–47,9]
o3-mini	RCI	L3	100,0% [88,6–100,0]	100,0% [88,6–100,0]	13,3% [5,3–29,7]
qwen2.5-coder:7b	ZS	L1	0,0% [0,0–11,4]	0,0% [0,0–11,4]	0,0% [0,0–11,4]
qwen2.5-coder:7b	ZS	L2	40,0% [24,6–57,7]	23,3% [11,8–40,9]	10,0% [3,5–25,6]
qwen2.5-coder:7b	ZS	L3	60,0% [42,3–75,4]	40,0% [24,6–57,7]	20,0% [9,5–37,3]
qwen2.5-coder:7b	RCI	L1	66,7% [48,8–80,8]	0,0% [0,0–11,4]	0,0% [0,0–11,4]
qwen2.5-coder:7b	RCI	L2	36,7% [21,9–54,5]	30,0% [16,7–47,9]	13,3% [5,3–29,7]
qwen2.5-coder:7b	RCI	L3	53,3% [36,1–69,8]	43,3% [27,4–60,8]	0,0% [0,0–11,4]
qwen3.6-plus	ZS	L1	100,0% [88,6–100,0]	100,0% [88,6–100,0]	3,3% [0,6–16,7]
qwen3.6-plus	ZS	L2	90,0% [74,4–96,5]	90,0% [74,4–96,5]	63,3% [45,5–78,1]
qwen3.6-plus	ZS	L3	96,7% [83,3–99,4]	96,7% [83,3–99,4]	10,0% [3,5–25,6]
qwen3.6-plus	RCI	L1	100,0% [88,6–100,0]	100,0% [88,6–100,0]	0,0% [0,0–11,4]
qwen3.6-plus	RCI	L2	96,7% [83,3–99,4]	96,7% [83,3–99,4]	23,3% [11,8–40,9]
qwen3.6-plus	RCI	L3	100,0% [88,6–100,0]	100,0% [88,6–100,0]	0,0% [0,0–11,4]

Tony, C., Díaz Ferreyra, N. E., Mutas, M., Dhif, S., and Scandariato, R. (2025). Prompting techniques for secure code generation: A systematic investigation. arXiv preprint arXiv:2407.07064.

Zhang, T., Pan, S., Zhang, Z., Xing, Z., and Sun, X. (2025). Deployability-centric infrastructure-as-code generation: An LLM-based iterative framework. arXiv preprint arXiv:2506.05623.