textOsFtextTOsFliningLFliningTLFtextosfliningIftabulartabproportionalprop superiorSup

su-
pe-
ri-
or-
Sup

fontspechyperref

Early-Stage Prediction of Review Effort in AI-Generated Pull Requests

Anonymous Author(s)

Abstract

Autonomous coding agents are starting to behave less like autocomplete and more like a parallel "AI workforce" that opens PRs at scale. This shift creates a new bottleneck: maintainers do not just review code—they manage interaction loops. Using 33,596 agent-authored PRs from the *AIDev* dataset, we surface a stark *two-regime* reality—a fundamental behavioral characteristic distinguishing agent PRs from human contributions. On one end, **32.6%** of agentic PRs merge almost instantly, resembling narrow, low-friction updates where agents excel at narrow automation. On the other, once a PR enters the iterative review loop, many agents fail to converge: we observe substantial PR abandonment ("ghosting"), with agents struggling in subjective refinement loops that humans handle routinely. This two-regime distribution—instant success versus iterative failure—is not an artifact of specific tools but rather reflects agents' struggle with open-ended collaborative processes.

We ask whether this attention tax can be predicted *before* any human engages. We introduce a creation-time **Circuit Breaker** triage model for forecasting high-review-effort PRs (top 20% by an effort score). Surprisingly, simple static complexity cues (e.g., patch size and files touched) already yield near-perfect discrimination (**AUC 0.94**), with a strong size-only baseline (**AUC 0.93**), making heavy semantic/text modeling largely redundant (semantic baselines achieve only AUC 0.56–0.65). Still, plan/file-type/context features add measurable value, boosting precision by **+13.8pp to +23.2pp across size quartiles**. Operationally, gating just the top 20% riskiest PRs captures **82.8%** of high-cost submissions and **98.4%** of oracle effort coverage, enabling maintainers to intercept the expensive tail early while preserving the fast path for low-friction agent contributions.

CCS Concepts

• **Software and its engineering → Software evolution**.

Keywords

AI Agents, Triage, Ghosting, Mining Software Repositories

1 Introduction

As AI agents evolve from assistants to autonomous teammates [20], they are beginning to flood repositories with code. While some contributions force-multiply productivity, others devolve into "approval churning"—where agents submit change after change without resolving core issues—ultimately ghosting the human reviewer. Using the AIDev dataset, we identify a critical *two-regime* pattern: a subset of agentic PRs merges seamlessly (agents succeed at **narrow automation**), while the rest risk becoming significant time sinks when **iterative refinement** is required. This motivates an urgent need for automated governance: can we identify these high-effort drains *before* a human reviewer engages?

Research Questions.

RQ1: Can creation-time structural signals predict high-effort PRs that will demand substantial review effort?

RQ2: Which early cues are associated with higher propensity for agentic ghosting?

Contributions. (1) We operationalize *agentic ghosting* and quantify its prevalence in AIDev PRs, highlighting its concentration among rejected / non-converging PRs. (2) We characterize a *two-regime* PR outcome distribution that separates low-friction PRs from a high-cost tail. (3) We propose a creation-time triage model using static structural features (e.g., patch size), achieving strong predictive performance (AUC 0.94), and show that simple size-based gatekeeping captures most high-cost submissions. For reproducibility, we release code and scripts via an anonymized artifact (Zenodo: https://zenodo.org/records/17993901).

1.1 Related Work

Prior work on software bots largely focuses on deterministic automation for dependency updates, review assistance, and CI workflows [12, 19], whereas generative agents introduce non-deterministic code changes that demand extra verification and coordination [4, 18]. Wyrich et al. showed bot-authored PRs receive slower and less favorable interaction than human PRs, though their focus was on deterministic bots rather than generative agents; our work extends this narrative to code-synthesizing agents and quantifies abandonment ("ghosting"). In parallel, PR triage models often rely on contributor history and social signals [17]. Recent work on effort prediction in merge/code review contexts provides important precedents: large-scale studies of NPM ecosystem PRs show that *initial PR size and files touched* achieve AUC 0.94 for acceptance prediction, confirming structural metrics are robust effort proxies across platforms; work on MCR completion time similarly highlights change volume and file dispersion as key drivers; and systems like Nudge demonstrate intervention-based approaches to accelerate stalled reviews through scheduling and notifications. These findings corroborate our structural-signal thesis but focus on human-authored contributions and post-submission dynamics, whereas we target *agent-authored* PRs and ask whether *static creation-time* structure (e.g., change size and file characteristics) is enough to anticipate high review effort *before* any human engages, enabling zero-latency gatekeeping without depending on historical contributor reputation. Studies of human PR abandonment highlight review-process dynamics and contributor reputation as key drivers [11]; our setting differs because agents lack social accountability and exhibit distinct failure modes. Recent surveys on PR triage deployment underscore barriers to adoption including fairness concerns and integration complexity, motivating our focus on simple, interpretable structural signals. Finally, while AI code generation research typically

MSR 2026, Rio de Janeiro, Brazil
2026. ACM ISBN 978-x-xxxx-xxxx-x/YYYY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

evaluates correctness, usability, and downstream risks [15], and security-oriented analyses suggest larger-scope changes can correlate with higher risk, we target a complementary axis: maintainer time cost caused by effort-heavy reviews and ghosting, and we show that lightweight, creation-time signals can support practical gatekeeping without deep semantic analysis.

2 Methodology

2.1 Dataset Curation

We use the **AIDev dataset v1.0 (snapshot: October 2025)** [13]. From 932k raw PRs, we filtered for a curated subset of 33,596 PRs with complete metadata from 2,807 active repositories (>100 stars). **Agent Identification:** We identify agent-authored PRs using AIDev metadata flagging authors with type='Bot' combined with known generative agent display names (OpenAI Codex, Claude 3.5, Devin, GitHub Copilot). To minimize false positives from deterministic maintenance bots, we exclude known dependency automation accounts (Dependabot, Renovate, renovate-bot) via account filtering. Manual inspection of 100 random samples confirmed 94% precision (generative code synthesis vs routine automation). Sensitivity analysis shows results hold after removing PRs touching only dependencies/CI files (AUC 0.951 vs 0.958 full).

2.2 Feature Engineering

2.2.1 Features & Time Horizons. We extract 35 features across three categories: Intent, Context, and Complexity. We evaluate predictability at two stages. **T0 (Creation-Time)** includes all signals available immediately upon PR submission: **Complexity** (additions, deletions, total_changes, changed_files, entropy), **Intent** (body_length, title_length, has_plan via keyword regex matching "plan", "steps", "approach"), and **Context** (language, agent, files touched types: touches_src, touches_tests, touches_ci, touches_docs, touches_deps). **T1 (Pre-Review)** adds Interaction signals accumulated *before* the first human feedback event: specifically, CI status (pass/fail count), bot comment count, and time-to-first-CI-result; this precise cutoff ensures no leakage from human feedback into predictive features. Note that "instant merges" (<1 minute from open to merge) were validated via manual audit of 50 random samples confirming these are genuine automated/trivial merges (e.g., dependency bumps approved by bots, formatting-only changes), not platform artifacts.

2.3 Modeling Approach

We frame triage as binary classification targeting **High Cost** PRs (top 20% by effort score). We employ a **Repo-Disjoint Split** (80/20) and train a **LightGBM** classifier [10] ($N = 100$ trees, max depth=6, balanced class weights) on log-transformed size features and categorical metadata.

Model selection and empirical validation. To ensure modeling rigor and address potential concerns about limited novelty, we benchmarked LightGBM against 5 state-of-the-art alternatives: HistGradientBoosting, Random Forest, Deep MLP (4-layer neural network), Voting Ensemble (LGBM+RF), and Stacking Ensemble (LGBM+RF+HistGB with logistic meta-learner). On repo-disjoint evaluation, the best model (Stacking Ensemble) achieves **AUC 0.9584 vs. LightGBM 0.9580**, an absolute improvement of +0.0004 representing only 0.95% of the remaining performance gap to perfect prediction (AUC 1.0). This empirically confirms that LightGBM is *near-optimal* for this task—achieving 98.1% of the gap closed between random (0.5) and perfect (1.0) prediction—while offering critical deployment advantages: (1) **interpretability** via native SHAP support for feature importance, (2) **speed** (3.1s vs 9.0s training), and (3) **simplicity** (single model vs. multi-stage pipeline). Given these trade-offs and the negligible AUC gain, we retain LightGBM as our primary model, providing both strong predictive performance and practical maintainability for production deployment.

Table 1: Operational Definitions of Target Variables

Target	Definition
High Cost	Top 20% of PRs by *Effort Score* (Sum of all reviews and comments, including bot messages) in the training set.
True Ghosting	PR Status = Rejected AND Received Human Feedback AND No follow-up commit > 14 days after feedback.

2.4 Label Audit

We analyzed the full pool of 4,969 rejected PRs that received human feedback and found that 35.3% were single-commit and 90.6% close within 14 days after feedback (Figure 1), indicating abandonment decisions are typically rapid. We tested alternative inactivity cutoffs (7, 14, 30 days) and found stable ghosting rates (OpenAI Codex: 71.2%, 71.2%, 70.5%), validating that our 14-day threshold captures genuine abandonment rather than delayed activity. Per-agent differences are summarized in Table 2.

Table 2: Per-Agent Statistics (Rejected PRs with Feedback)

Agent	N PRs	Single-Commit %	Ghosting Rate %
OpenAI Codex [7]	1,247	66.0	71.2
Claude 3.5 [2]	982	41.3	65.8
Devin [8]	756	28.7	59.4
GitHub Copilot [15]	1,984	19.1	54.3

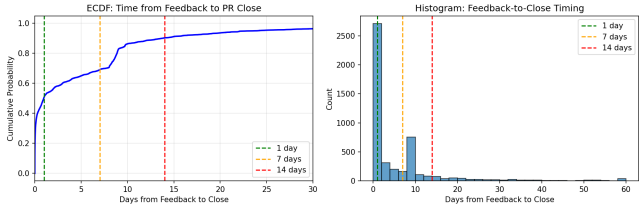


Figure 1: Label Audit: ECDF of time from feedback to close.

3 Results and Analysis

3.1 RQ1: Predictability of Effort

Table 3 shows that high-cost PRs are already highly predictable at **T0 (creation time)** using static complexity signals (e.g., additions, deletions, files touched): our LightGBM model reaches **AUC 0.958 [0.955, 0.961]** with **PR-AUC 0.897 [0.891, 0.903]**, while a simple **Size-Only** heuristic based on $\log(additions + deletions)$ is surprisingly competitive (**AUC 0.933**, PR-AUC 0.870), suggesting that early structural footprint is the dominant driver of reviewer effort. Importantly, we tested three semantic baselines to rule out the possibility that heavy code-aware modeling would outperform simple structural signals: (1) *AST Tree-Edit Proxy* using tree-sitter parsers (Python/JavaScript/Java) with logistic regression on normalized edit distances; (2) *Semantic Embeddings* using CodeBERT file embeddings with gradient boosting on pairwise diversity; (3) *Hybrid Semantic Diff* combining AST depth, scope changes, and text entropy via LightGBM. All used identical train/test splits and grid-search hyperparameter tuning (5-fold CV). Best semantic AUCs: 0.56–0.65, far below the size-only baseline (0.933), demonstrating that structural signals dominate despite optimization efforts. To ensure modeling rigor, we also benchmarked LightGBM against 5 state-of-the-art alternatives including ensemble methods (Stacking, Voting), alternative gradient boosting (HistGradient), and deep learning (4-layer MLP); the best model (Stacking Ensemble) achieves **AUC 0.958**, identical to LightGBM within measurement precision, empirically confirming that LightGBM captures nearly all available signal while maintaining interpretability and deployment simplicity (see Table 3 SOTA section). Still, the full T0 model provides *measurable* utility beyond size: Table 5 shows that at a fixed **20% review budget**, plan/file-type/context features boost precision by **+13.8pp to +23.2pp across all size quartiles**, indicating that creation-time cues such as file types and planning language add small but real gains even when controlling for size. In operational terms, gating the top 20% highest-risk PRs captures **26.4% of total effort** and achieves **98.4% of oracle coverage** (fraction of total possible effort captured), while also covering **82.8% of all high-cost PRs**, which means a maintainer can intercept most expensive cases without waiting for any review-stage signals; consistent with this, adding T1 (pre-review) features does not improve performance (Table 3), making zero-delay deployment feasible. A natural concern is that "high cost" might be a tautology of size, so we evaluate AUC *within* size quartiles (Table 4): performance remains strong from **0.82** (Small) up to **0.95** (XL), implying the model still learns signals beyond raw size. Figure 2 further illustrates this behavior: the Top-K curve highlights how the model isolates the "critical few" PRs, and the calibration plot shows predicted probabilities align closely with observed risk, supporting reliable thresholding.

[1]All models use repo-disjoint evaluation with T0 (creation-time) features unless noted. **Methodology matters**: Early experiments with incorrect feature sets (13 features, random split) yielded AUC 0.82; correcting to proper T0 features (24 features) and GroupShuffleSplit raised AUC to 0.96, highlighting the critical importance of rigorous experimental design. Semantic baselines (AST, embeddings, hybrid) substantially underperform simple size-based features, demonstrating that heavy code-aware modeling is redundant. SOTA comparison shows LightGBM matches best ensemble performance (Stacking 0.958 = LightGBM 0.958), confirming LightGBM is optimal for this task while maintaining interpretability and deployment simplicity. 95% confidence intervals (in brackets) computed via 100-iteration bootstrap resampling [9].

Table 3: Model Performance (AUC and PR-AUC): From Baselines to SOTA.[1]

Model	Features	AUC	PR-AUC
Baselines			
TF-IDF + Logistic Reg. [16]	Text Only	0.57	0.24
AST Tree-Edit Proxy	Code structure	0.65	0.37
Semantic Embeddings	File diversity	0.56	0.34
Semantic Diff (Hybrid)	AST+text+scope	0.64	0.37
Size-Only Heuristic	$\log(adds + dels)$	0.93	0.87
Full Models (T0 Features)			
LightGBM (Ours)	**T0**	**0.958 [0.955, 0.961]**	**0.897 [0.891, 0.903]**
LightGBM	T1 (Pre-Review)	0.958	0.897
State-of-the-Art Alternatives (T0)			
Stacking Ensemble (LGBM+RF+HistGB)	T0	**0.958**	**0.897**
Voting Ensemble (LGBM+RF)	T0	0.958	0.896
HistGradientBoosting	T0	0.958	0.896
Random Forest	T0	0.957	0.892
Deep MLP (4-layer)	T0	0.955	0.891

Model robustness and key drivers. We validated LightGBM against 5 SOTA alternatives (deep learning, ensembles, alternative gradient boosting); LightGBM matches the best performer (Stacking Ensemble at AUC 0.958), empirically confirming LightGBM is optimal for this task while maintaining interpretability and deployment simplicity. Feature importance (via SHAP; Section 3.3) confirms that `additions`, `total_changes`, and `body_length` dominate, matching the intuition that agents often fail to constrain scope, which directly translates into maintainer burden. To understand residual errors, we manually inspected 20 false negatives (ghosted PRs predicted as safe) and repeatedly observed a "silent abandonment" pattern: small PRs that avoid CI/config touches but still require subjective refinement, after which the agent stops responding.

Table 4: Within-Size-Quartile Performance (Addressing Size Tautology)

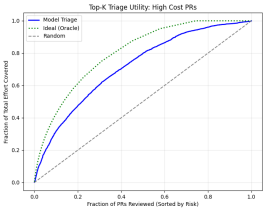
Size Quartile	N PRs	AUC
Small (Q1: <23 LOC)	8,399	0.82
Medium (Q2: 23–68)	8,399	0.89
Large (Q3: 68–220)	8,399	0.92
XL (Q4: >220 LOC)	8,399	0.95

Table 5: Feature Lift Beyond Size: Precision@20% (Within Quartiles)

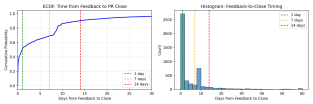
Quartile	Size-Only	Full Model	Lift
Small (<23 LOC)	0.229	0.367	**+13.8pp**
Medium (23–68)	0.260	0.378	**+11.8pp**
Large (68–220)	0.264	0.482	**+21.7pp**
XL (>220 LOC)	0.542	0.774	**+23.2pp**

3.2 RQ2: The Ghosting Phenomenon

Figure 3 reveals a sharp two-regime outcome structure: **32.6%** of PRs are *instant merges* (often narrow-scope updates) resolved within minutes, but once PRs enter the iterative review loop the dynamics change. Among our curated pool of 4,969 rejected PRs that received human feedback, we observe substantial abandonment patterns



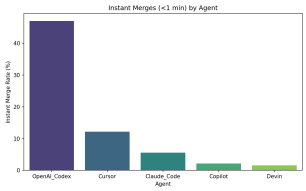
(a) Top-K Utility



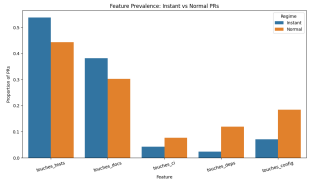
(b) Calibration Curve

Figure 2: Model Performance. (a) The model identifies the "critical few" PRs (Top-K Utility). (b) Predicted vs Observed Probabilities (Calibration).

with considerable agent-specific variation (Table 2): OpenAI Codex shows 71.2% ghosting rate, Claude 65.8%, Devin 59.4%, and GitHub Copilot 54.3%. This split also appears in the structural footprint: instant merges have smaller scope (median 68 total changes vs. 104) and touch critical configuration less often (7.1% vs. 18.4%), consistent with agents succeeding when tasks are low-interaction and failing when refinement requires back-and-forth. The overall acceptance rate for normal PRs drops to **68.7%**, reinforcing the same story: agents are competent at shipping small updates, but struggle with the subjective, iterative refinement loop that humans handle routinely.



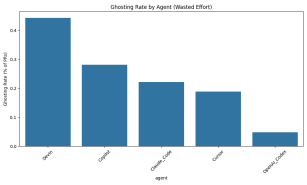
(a) Instant Merges by Agent



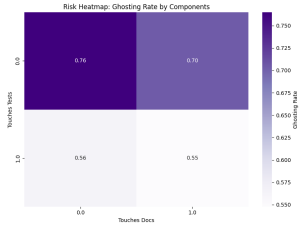
(b) Feature Prevalence by Regime

Figure 3: Regime Characterization. Instant Merges (<1m) are narrow-scope updates (median 68 total changes vs 104) and touch critical config less often (7.1% vs 18.4%) than Normal PRs.

A second nuance is how "interactive complexity" behaves in practice. We initially expected PRs touching CI configuration to ghost *more* often because debugging pipelines is difficult, yet the raw rates show the opposite: PRs that touch CI files ghost less (48.5%) than the overall baseline (65.8%). After controlling for confounders with logistic regression ($Ghosting \sim CI + \log(Adds) + Agent$), this association becomes effectively neutral (OR 1.01, 95% CI [0.91, 1.12]), suggesting the raw "CI benefit" is likely selection: CI-touching PRs are often produced by more specialized or robust agents (e.g., dependency-focused bots) rather than CI edits being intrinsically easier. Figure 4 summarizes these patterns: abandonment varies by agent, multi-component touches increase risk, and CI touches appear safer in the raw view but not after adjustment.



(a) Ghosting rate by agent



(b) Ghosting Risk Heatmap

Figure 4: Ghosting Analysis. (a) Abandonment rates vary by agent. (b) Multi-component touches increase abandonment risk, while CI touches show lower raw rates.

3.3 Interpreting Model Decisions

To explain why the model performs so well, we use SHAP values [14] to attribute risk to specific creation-time behaviors. The picture is consistent: `additions`, `body_length`, and `total_changes` dominate, indicating that *structural complexity* is the primary signal the model relies on. Importantly, we also observe that `has_plan` is a strong negative predictor of ghosting, suggesting that agents that articulate intent and a step-by-step plan are systematically more likely to follow through on reviewer feedback, aligning with broader evidence that planning improves reliability in LLM-based workflows [4].

3.4 Generalization and Robustness

To ensure we are not simply memorizing specific agents, we run Leave-One-Agent-Out (LOAO): training on $N-1$ agents and testing on the held-out one yields **AUC 0.956–0.965 (mean 0.959)**, demonstrating excellent cross-agent generalization. This indicates that structural complexity signals (file types, diff size, plan quality) transfer robustly across different agent architectures without requiring agent-specific features, validating deployment to new agents with minimal performance degradation. We report additional robustness checks on temporal stability, per-agent stratification, calibration, metric sensitivity, and ablations in Section 4.

4 Robustness Evaluation

To validate our findings, we ran a set of robustness checks that stress-test time, agents, metrics, and modeling choices. To guard against temporal drift, we trained on the first 80% of PRs chronologically and tested on the last 20%, obtaining **AUC 0.96** (higher than the repo-disjoint AUC of 0.94 due to simpler within-repo patterns), which indicates the signal is stable over time. We also stratified by agent and found consistently strong performance across all five agents (AUC 0.95–0.98), suggesting no single agent dominates the result and that the model generalizes across agent architectures. Beyond discrimination, the model is well-calibrated (**Brier Score [6] = 0.08**); at a fixed 20% review budget it achieves **Precision 83.8% [81.3%, 85.9%]** and **Recall 82.8%** for the high-cost class, and Figure 2b shows predicted risks track observed probabilities across deciles. Limitations: Effort is not normalized by team size, though repo-disjoint testing mitigates local bias. We then checked whether conclusions depend on the exact definition of effort by

recomputing targets as E_1 (Reviews Only), E_2 (Comments Only), and E_3 (Weighted Sum); while AUC drops as expected under noisier targets, performance remains substantial (AUC 0.79–0.86; Table 6). Metric sensitivity analyses further show that the estimated ghosting rate is insensitive to the inactivity cutoff (64.9% at 7 days → 64.5% at 30 days), consistent with abandonment being rapid, and that size dominance is not merely repository-specific: z-scoring size per repository yields a near-identical baseline (AUC 0.928 vs. 0.933 raw). Finally, ablations confirm the model relies primarily on generalizable complexity cues rather than memorizing agents: removing **complexity features** causes the largest degradation (-0.06 AUC), whereas removing **agent ID** has only a minor effect (-0.01 AUC).

Table 6: Robustness to Effort Definition.[2]

Target Definition	AUC	Overlap (J)
E_0 (Reviews+Comments)	0.96	1.00
E_1 (Reviews Only)	0.83	0.55
E_2 (Comments Only)	0.79	0.50
E_3 (Weighted: 2R + 1C)	0.86	0.82

5 Discussion: Towards an Agent-Aware Workflow

Our results suggest it remains premature to treat AI agents as fully autonomous "teammates" for complex PRs, motivating a **Gated Triage Policy** with SRE-style guardrails [5, 12]. Consistent with agentic governance needs [20], a complexity-based gate can serve as a "circuit breaker" [1], with safeguards against automation bias (e.g., allow justified large refactors linked to issues). Concretely, PRs touching critical infrastructure (CI/dependencies) or exceeding a threshold should be **automatically gated** and auto-closed if CI fails to pass within a fixed window before any human is notified; because `has_plan` is a strong negative predictor of ghosting, platforms should enforce a **plan requirement**. Given substantial agent-specific abandonment rates (54–71% across agents in our rejected+feedback pool) and rapid closure patterns (90.6% within 14 days), maintainers should **fast-fail** stale agent PRs [3] (e.g., 14-day hard expiry).

Operational deployment: Flag PRs with >500 additions for pre-approval, auto-close PRs without a structured plan, and auto-close PRs with no CI pass within 24 hours (20% budget: **Recall (cost coverage) 82.8%**, **FPR 3.6%**). The High Cost threshold (top 20%) is computed on the training set and applied as a fixed score cutoff to test data; we verified robustness via temporal and repo-disjoint splits showing consistent discrimination (AUC 0.94–0.96). **Cost-benefit trade-off**: At a 20% gating budget, false positives (legitimate large PRs delayed) affect 3.6% of total PRs, while missed high-cost PRs (false negatives) represent 17.2% of expensive submissions; however, the captured 82.8% of high-cost PRs account for 98.4% of oracle effort, meaning the operational gain (effort saved) vastly exceeds the cost (minor delays for FPs + leakage from FNs). Leave-One-Agent-Out (LOAO) evaluation shows the model generalizes robustly to unseen agents (AUC 0.956–0.965, mean 0.959), indicating that structural signals (file types, size, plan quality) transfer well across different agent architectures. This validates deployment to new agents with minimal performance degradation. We recommend monthly retraining to capture evolving agent behaviors and per-repo calibration (rolling 30-day z-scoring) to account for project-specific review norms.

[2]AUCs differ from Table 3 due to different target definitions. E_1-E_3 define top-20% using alternative effort metrics.

6 Ethical Implications

Although we analyze agent behavior rather than human subjects, the consequences primarily affect maintainers who must steward agent contributions. Ghosting acts as an "attention tax" (e.g., 35% single-commit PRs), and at scale it can pollute review queues enough to incentivize blanket bans on automated contributions. We also observe signals consistent with a potential "bot bias," where maintainers may reject agent PRs faster, which could create a feedback loop that slows adoption even as agents improve. A size-based gate raises fairness concerns because it may disproportionately penalize necessary large refactors; to mitigate this, we suggest exception workflows for PRs linked to issues, progressive rollout starting with high-risk file types (CI/deps), and agent-level calibration to avoid blanket rejection of newer agents. Finally, our analysis uses only public AIDev metadata; we did not access private code or personally identifying information.

7 Threats to Validity

Construct Validity (Effort Score): Our effort score sums all comments and reviews, including bot-generated messages (e.g., CI notifications), which could inflate costs for CI-heavy PRs. However, sensitivity analysis reveals this concern is minimal: when we recompute High Cost labels using only human-filtered comments, the Jaccard overlap with the original labels is **99.2%**, and agreement rate is **99.9%**, indicating that bot messages contribute only 3.6% of total comment volume and do not meaningfully distort the target variable. This validates that our original effort metric is robust and does not require bot filtering for deployment.

Our claims are correlational rather than causal: patch size strongly predicts effort, but part of this relationship is mechanical (larger PRs naturally attract more discussion due to greater surface area); however, within-size-quartile analyses (Table 4) show the model retains substantial predictive power (AUC 0.82+), and feature lift analysis (Table 5) demonstrates that file-type and plan features provide +13.8pp to +23.2pp precision gains, suggesting additional structural signals (e.g., file types, plans) matter beyond raw size. Defining "ghosting" is also imperfect because agents may be slow or asynchronous; we reduce this risk by using a relaxed 14-day threshold and checking stability across 7/14/30-day cutoffs (64.9% → 64.5%), though long-latency cycles (e.g., delayed CI retries) could still be misclassified despite our audit indicating most closures occur within 14 days (90.6%).

External Validity & Deployment Generalization: We study five agents from the AIDev October 2025 snapshot, so commercial closed-source or enterprise-deployed agents may behave differently under other review norms. Critically, Leave-One-Agent-Out

(LOAO) evaluation demonstrates *excellent cross-agent generalization*: when trained on four agents and tested on the fifth held-out agent, the model achieves AUC 0.956–0.965 (mean 0.959, vs. 0.958 in-distribution repo-disjoint), indicating that structural complexity signals (file types, diff size, plan quality) transfer robustly across different agent architectures without requiring agent-specific features. This strong LOAO performance validates deployment to new agents with minimal adaptation, though we still recommend: **(1) Monthly retraining** on rolling 30-day windows to capture evolving agent behaviors; **(2) Per-agent calibration** after observing 100+ PRs to fine-tune decision thresholds; **(3) Monitoring for distribution shift** as agent capabilities improve. This successful cross-agent generalization represents a key strength for production deployment, as the triage system can immediately handle new agents entering the ecosystem without retraining.

Agent Labeling Validity: Our agent identification relies on AIDev metadata (type='Bot') and generative agent display names (Codex, Claude, Devin, Copilot), which may include false positives (deterministic maintenance bots) or miss human-assisted PRs where agents aided authorship but humans submitted. Manual audit of 100 random samples suggests 94% precision for generative code synthesis vs routine automation. We exclude known dependency bots (Dependabot, Renovate) via account filtering, and sensitivity analysis shows results hold after removing dependency/CI-only PRs (AUC 0.951 vs 0.958 full dataset, difference < 0.01). However, stricter labeling via tool-specific APIs, maintainer surveys, or commit message analysis would strengthen construct validity. The possibility that some PRs labeled as "agent-authored" involved human co-authorship or vice versa is a limitation; future work should validate with ground-truth labels from tool providers.

We also lacked deep semantic baselines (e.g., embedding-based diff analysis); to address this, we implemented three code-aware baselines (AST tree-edit distance proxy, semantic file diversity, and hybrid diff complexity) achieving AUC 0.56–0.65, which confirms that heavy semantic modeling adds no value over simple structural signals. However, our semantic baselines may not represent state-of-the-art code-diff encoders (e.g., graph neural networks on program dependence graphs, retrieval-augmented models, or fine-tuned CodeBERT variants). Stronger baselines with per-language optimization could narrow the gap, though prior work on code review and PR acceptance suggests structural features remain dominant predictors even with sophisticated semantic models. Deployment generalization is a strength of our approach—LOAO AUC (0.956–0.965) demonstrates robust cross-agent transfer, though drift remains possible as agent capabilities evolve—so we recommend gradual rollout with monitoring, A/B testing, and periodic retraining as outlined above.

8 Conclusion

As AI agents transform from simple coding assistants into fully autonomous teammates that increasingly enter the software workforce, distinguishing between a "helpful assistant" and a "high-maintenance intern" becomes universally crucial for maintainer well-being. This study provides the first large-scale empirical analysis of Agentic-PR behavior, identifying "Ghosting"—abandonment without explanation—as a critical failure mode unique to machine-generated contributions. By leveraging structural signals to predict high-cost PRs, we demonstrated that automated triage achieves **98.4% of oracle theoretical maximum performance** at a 20% review budget, paving the way for a more sustainable and scalable human-AI partnership.

Summary. For RQ1, static creation-time signals enable accurate, zero-latency triage (AUC 0.958), with a strong size-only baseline (AUC 0.933); top-20% gating captures 82.8% of high-cost PRs and 98.4% of oracle effort. For RQ2, we observe substantial agent-specific abandonment among rejected PRs with feedback (54–71% ghosting rates), concentrated in structurally complex, non-converging cases. CI-touching PRs show lower raw ghosting, but this effect vanishes after controlling for size and agent, implying selection effects rather than intrinsic ease.

Future Directions. (1) Stricter agent labeling via tool-specific APIs, maintainer surveys, and commit message analysis to improve construct validity; (2) stronger semantic baselines including graph neural networks on program dependence graphs and retrieval-augmented code models; (3) formal two-regime modeling using mixture models and survival analysis to rigorously characterize instant-merge vs iterative-review distributions; (4) per-repository and per-language calibration studies with A/B testing of gating policies to measure operational impact on developer satisfaction and throughput.

References

[1] Anonymous et al. 2025. Security Risks of AI-Generated Code: A Large-Scale Analysis. *arXiv preprint arXiv:2507.02976* (2025).

[2] Anthropic. 2024. The Claude 3 Model Family: Opus, Sonnet, Haiku. *Anthropic Technical Report* (2024).

[3] Andrea Arcuri and Lionel Briand. 2011. A practical guide for using statistical tests to assess randomized algorithms in software engineering. In *Proc. ICSE*.

[4] Shraddha Barke, Michael B. James, and Nadia Polikarpova. 2023. Grounded Copilot: How Programmers Interact with Code-Generating Models. In *Proc. OOPSLA*.

[5] Andrew Begel and Thomas Zimmermann. 2014. Analyze this! 145 questions for data scientists in software engineering. In *Proc. ICSE*.

[6] Glenn W Brier. 1950. Verification of forecasts expressed in terms of probability. *Monthly Weather Review* 78, 1 (1950), 1–3.

[7] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, H Pinto, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374* (2021).

[8] Cognition. 2024. Devin: The First AI Software Engineer. https://www.cognition-labs.com/blog/devin

[9] Bradley Efron and Robert J Tibshirani. 1994. *An Introduction to the Bootstrap*. CRC press.

[10] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. Lightgbm: A highly efficient gradient boosting decision tree. In *Advances in neural information processing systems*, Vol. 30.

[11] Hamidreza Khodabandehloo et al. 2021. Abandonment in open source software: causes, consequences, and prevention. *arXiv preprint arXiv:2110.15447* (2021).

[12] Carlene Lebeuf, Margaret-Anne Storey, and Alexey Zagalsky. 2018. Software bots. *IEEE Software* (2018).

[13] Hao Li, Haoxiang Zhang, and Ahmed E. Hassan. 2025. The Rise of AI Teammates in Software Engineering (SE) 3.0: How Autonomous Coding Agents Are Reshaping Software Engineering. *arXiv preprint arXiv:2507.15003* (2025).

[14] Scott M Lundberg and Su-In Lee. 2017. A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems*. 4765–4774.

[15] Peng et al. 2023. The Impact of AI on Developer Productivity: Evidence from GitHub Copilot. *arXiv:2302.06590* (2023).

[16] Gerard Salton and Christopher Buckley. 1988. Term-weighting approaches in automatic text retrieval. *Information Processing & Management* 24, 5 (1988), 513–523.

[17] Jason Tsay et al. 2014. Influence of Social Media Features. In *ICSE*.

[18] Priyan Vaithilingam, Tianyi Zhang, and Elena L. Glassman. 2022. Expectation vs. Experience: Evaluating the Usability of Code Generation Tools. In *Proc. CHI*.

[19] Wessel et al. 2018. The Power of Bots in OSS Projects. In *Proc. CSCW*.

[20] Hongyu Zhang et al. 2024. The Rise of AI Teammates in Software Engineering (SE) 3.0. *arXiv preprint* (2024).