STEERSIM: DE-IDENTIFYING DEVELOPER BEHAVIOR IN AGENT TRACES VIA USER SIMULATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Developer–agent interaction logs are valuable training data, but releasing them poses privacy risks beyond personally identifiable information. We observe that how long developers let an agent run, how closely they scrutinize proposed actions, and how they phrase corrections can form a distinctive behavioral fingerprint that conventional anonymization leaves intact. We introduce SteerSim, the first agentic user simulator that replaces the developer in a live coding session to enable privacy-preserving trace synthesis. Operating within the coding agent's environment, SteerSim inspects the repository and takes actions to steer the agent toward task completion. It is conditioned on a persona extracted from real developer sessions, which can be transformed to obscure the original developer's behavioral patterns. We further propose SteerRL, a training algorithm that combines supervised fine-tuning with agentic reinforcement learning using rubric-based rewards. We evaluate both on real sessions from SWE-Chat along three dimensions: human-likeness, re-identification risk, and downstream utility. With Opus 5 as its backbone, SteerSim produces interactions that are nearly indistinguishable from human interactions, as measured by the Turing score. Transforming personas reduces re-identification risk to near chance while preserving downstream utility, achieving a favorable privacy–utility tradeoff. Finally, applying SteerRL to Qwen3.5-4B yields a simulator with performance comparable to Opus 5, offering a practical and low-cost alternative.

1 INTRODUCTION

Coding agents increasingly operate in long-running, interactive sessions where the human developer does far more than provide an initial task: they monitor progress, interrupt unproductive behavior, answer clarifying questions, approve or reject plans, edit files themselves, and issue follow-up instructions. These steering decisions shape the trajectory the agent ultimately takes, and logs of such sessions are among the most valuable artifacts for training the next generation of agents. They record not just what a competent agent does, but how it reacts to different types of developer interventions.

Sharing these logs, however, carries a privacy risk that goes beyond the personally identifiable information (PII) that standard release pipelines scrub. Steering behavior is idiosyncratic: how long a developer lets the agent run, how much they scrutinize before approving, how they phrase a correction. In our experiment on 700 sessions from 47 developers in the SWE-Chat dataset (Baumann et al., 2026), an attacker is given the user side of a target session and a short behavioral profile of each of 10 candidate authors. The attacker identifies the author of that session with 30.9% top-1 accuracy, over three times the 10% chance level. PII redaction, substitution, or rewriting of the released log leaves this rate essentially unchanged (Appendix F). Release-time anonymization is therefore not sufficient for agentic interaction data: the identity signal survives in the decisions themselves.

To close this gap, we take a generative route: *replace* the logged human with a simulator, and make behavioral privacy a property of the generator rather than of a post-hoc filter. Our simulator, SteerSim, is itself an agent. It lives in the same live environment as the coding agent and has access to the real repository checkout at the session's original state. It wakes on a polling schedule to observe the agent, optionally reading code or running commands, sending follow-ups, or staying silent. SteerSim is conditioned on a compact session *persona* induced from the real developer's behavior.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Generating under a *transformed* persona yields trajectories whose behavioral re-identification rate approaches chance while staying human-like and persona-consistent (Section 6.2).

The simulator can naturally be powered by frontier API models, but they are costly and closed. To address this issue, we propose SteerRL, a novel post-trainining algorithm that optimizes Qwen3.5-4B in two stages: supervised fine-tuning on the per-poll decisions of a frontier teacher, then agentic reinforcement learning over full live sessions with rubric-based rewards, in which LLM judges grade persona alignment at every steering moment and human-likeness once the session ends.

Experiments on SWE-Chat show that SteerSim with a frontier backend is nearly indistinguishable from real developers under blinded pairwise judging and that persona transformation cuts re-identification to near chance; the rate also falls with persona distance. SteerRL lifts Qwen3.5-4B from 2.98 to 3.90 human-likeness (real developers anchor the scale at 4.00, the strongest frontier backend reaches 3.85) with the highest persona alignment of any backend, and the transformed sessions are as useful as real ones for training a coding agent.

We summarize the main contributions below:

- **We identify a novel behavioral re-identification risk** that lies in released agent traces, which carry an identity signal that PII anonymization does not remove: a behavioral-profile attacker identifies the author at over three times chance before and after anonymization.
- **We propose SteerSim, the first agentic user simulator with persona transformation** that acts inside the live coding environment and regenerates a session under a transformed persona, significantly reducing behavioral re-identification risk at little cost in human-likeness or persona alignment.
- **We propose SteerRL, a novel post-training recipe** that combines supervised fine-tuning and agentic RL under rubric-based rewards, with an opportunity gate that grades the persona only where the state tests it, a longitudinal scoreboard that credits cross-poll consistency, and local credit assignment across asynchronously diverging live rollouts, enabling Qwen3.5-4B to match the simulation performance of frontier backends like Opus 5.
- **We conduct comprehensive evaluation** on the generated synthetic traces on three downstream coding tasks. We show that synthetic and real sessions are equally useful as agent training data while protecting the data providers from behavioral re-identification.

2 Related Work

Re-identification from released data. LLMs make it easy to identify people from unsigned text: they attribute authorship without task-specific training (Huang et al., 2024), infer location, age, and income from casual text (Staab et al., 2024), link released interview transcripts to participants (Li, 2026), and tell chatbot users apart from their prompts alone (Patel et al., 2026). At the same time, logs of real users talking to chatbots (Zhao et al., 2024; Zheng et al., 2024) and, recently, of developers working with coding agents (Baumann et al., 2026; Tang et al., 2026) are released after PII redaction and credential scrubbing, because such logs contain personal disclosures in unexpected places (Mireshghallah et al., 2024). Agent traces are especially rich in behavioral signal, since interaction habits vary systematically across developers (Richards et al., 2026), but whether an author survives redaction has not been measured. We show that the identity signal lives in the steering behavior itself, and we suppress it by regenerating the human side.

Simulated users for agent evaluation. LLMs prompted to act as users behave unlike real peopl. Nn τ-bench the agent's success depends on which LLM plays the user, and simulated users are far more polite and cooperative than humans (Seshadri et al., 2026); dozens of prompted and fine-tuned simulators behave too uniformly relative to human variation, and a hand-written persona prompt can make some behaviors less human-like (Zhou et al., 2026). This lack of variety is also a problem for training: a policy optimized against one frozen simulator exploits its habits. Recent work therefore builds persona populations: SCOPE co-trains a population of simulators (Yu et al., 2026), TRIP trains against personality profiles (Zhang et al., 2024), PPol searches persona policies that fool a discriminator trained on human traces (Chopra et al., 2026), and UserHarness models the user's mind inside the agent (Qian et al., 2026). These works study customer-service dialogue, where the user's entire action space is text, and use personas as population-level stress tests. In contrast,

SteerSim derives personas from individual real developers and acts inside the live repository: it inspects files, interrupts the agent, or chooses not to respond at all.

Training user simulators. Trained simulators mostly predict the next human utterance, and judge-rated similarity rewards are easily gamed (Gandhi et al., 2026). HumanLM aligns the user's latent state instead of the wording (Wu et al., 2026), Turing-RL rewards indistinguishability from the real user (Wang et al., 2026), UserLM-R1 combines rubric- and rule-based rewards for profile-conditioned voice-agent users (Zhang et al., 2026), and persona consistency has been turned into a per-turn reward (Abdulhai et al., 2026). All of these have a ground-truth human turn at every step. Our setting has none: the coding agent reacts to the simulator, so the generated trace diverges from the log immediately, and the simulator must also decide when to stay silent or read code rather than speak. SteerRL therefore pairs an per-turn persona-alignment reward with a per-session Turing term that measures human likeness over complete live rollouts, and uses the simulator to cut the link between the released trace and its author.

3 Problem Setup

Agent traces. A developer $u \in \mathcal{U}$ gives a coding agent a task on a repository snapshot R and supervises its work. We record the interaction as an *agent trace* $\tau = (x, \mathbf{a}_1, \mathbf{s}_1, \ldots, \mathbf{a}_T, \mathbf{s}_T)$, where x is the opening prompt, the agent turns $\mathbf{a}_t$ are the agent's messages and tool calls before the developer next acts, and the steer turns $\mathbf{s}_t$ are the developer's responses: inspecting the work or the repository, editing files, addressing the agent, or staying silent (Section 4.3). $u(\tau)$ is the developer behind a trace and $\mathrm{user}(\tau) = (\mathbf{s}_1, \ldots, \mathbf{s}_T)$ its user side, which excludes the opening prompt.

Threat model: behavioral re-identification. A trace reveals its author through its *content* (task, repository, code), which release pipelines already target and which makes the trace useful as training data, so a regenerated trace deliberately keeps x and R fixed; and through its *behavior*, i.e., how the developer steers the agent, which no existing tool addresses and which this paper studies. The *profile* adversary receives the user side $\mathrm{user}(\tau)$ of a released trace, real or generated, and K personas of distinct developers, one of them summarized from a different session of $u(\tau)$. It never sees the opening prompt, the repository, or the agent's side, so every correct guess is due to steering behavior alone. The re-identification rate (reid) is the probability that it names $u(\tau)$, with chance $1/K$; we fix $K = 10$. Appendix E motivates this design and adds a stronger raw-transcript adversary.

4 SteerSim: Persona-Conditioned Agentic User Simulation

4.1 Overview

Simulating a developer differs from simulating a chat user: a developer reads the agent's output, edits code, interrupts, approves plans, or stays silent, and each of these choices depends on a live repository and a running agent that next-utterance simulators never observe. We propose SteerSim (Figure 1), the first agentic simulator that is conditioned on a compact *session persona* (Section 4.2), follows the developer's *steering protocol* (Section 4.3) inside the coding agent's live environment, where a policy π_{steer} chooses each steer turn, and supports *persona transformation* for privacy (Section 4.4). Any tool-using LLM can serve as the policy π_{steer}; Section 5 introduces our post-training algorithm to optimize a small open LLM.

4.2 Session Persona

A persona $p = \psi(\tau)$ is a compact first-person profile of how the developer of trace τ steered the coding agent *in that session* (Figure 2). Personas are attached to sessions, not users: a developer with several traces has several personas, and nothing in $\psi(\tau)$ is taken from their other traces. A persona is written along $K{=}10$ bipolar *cognitive axes*—patience, scrutiny, involvement, decisiveness, specificity, communication, planfulness, cooperativeness, expertise, and language. We take these dimensions from prior studies of how developers differ when working with coding assistants (Richards et al., 2026; Tang et al., 2026; Baumann et al., 2026). A frontier-LLM *inducer* ψ reads the trace with tool outputs stripped, together with a few session statistics, and writes one entry per axis

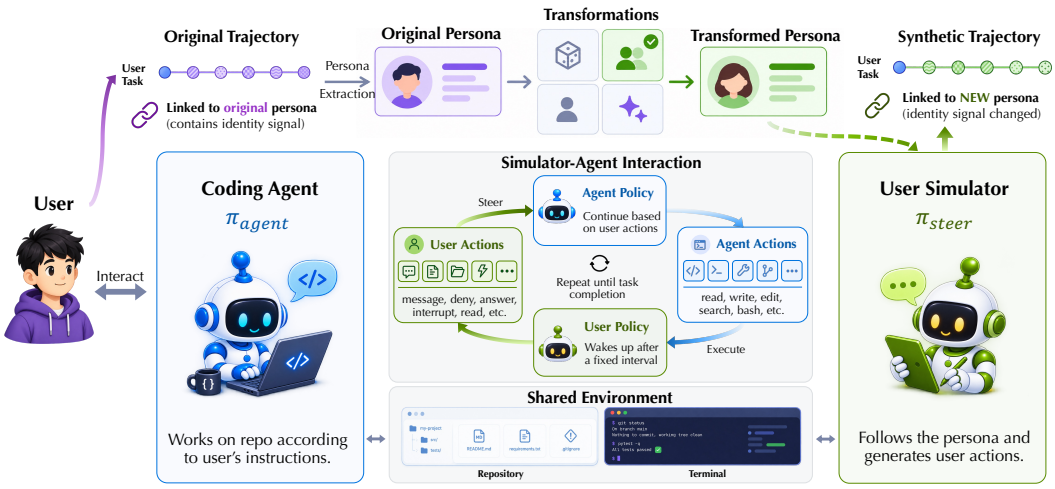


Figure 1: **SteerSim overview.** A real user's trace is linked to their behavioral persona and carries an identity signal. We extract the persona, *transform* it, and hand it to an agentic user simulator π_{steer} that shares a live environment (repository and terminal) with the coding agent π_{agent}. The simulator observes the agent's work, takes its own reading or editing actions, and issues steering actions; the generated trace is linked to the *new* persona, so the original user's released trace is linked to the new persona rather than to the original developer.

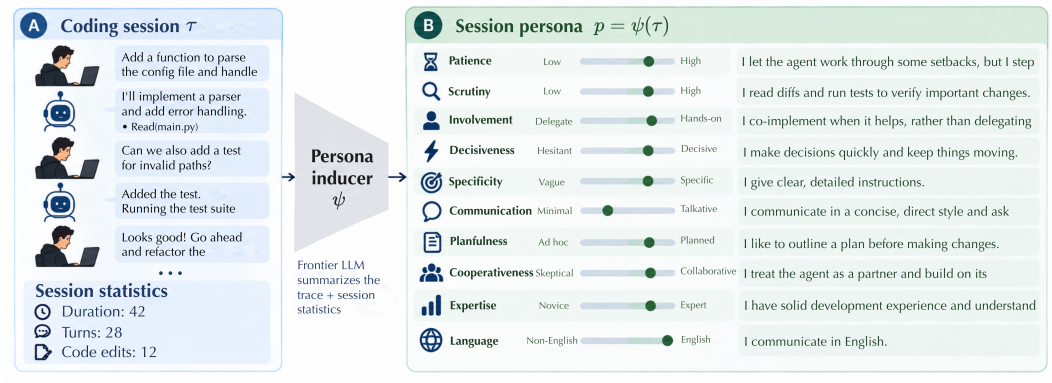


Figure 2: **Session persona extraction.** (A) A single coding session τ, accompanied by a few session statistics, is summarized by a frontier-LLM inducer ψ. (B) The resulting persona $p = \psi(\tau)$: a first-person description of the developer's steering habits along ten bipolar cognitive axes, free of task content.

in the developer's voice: where they sit on that axis and the habit that shows it. Entries are qualitative and carry no numbers, quotations, or identity attributes. They are written in English regardless of the language the developer used. More details about the definition of the axes and persona extraction prompts can be found in Appendix B and Appendix A.2.

4.3 Steering Protocol

Agent and steer turns. An agent turn $\mathbf{a}_t$ is the sequence of assistant messages and tool calls the agent takes before the developer next acts; the agent is always in one of four *states*, $\mathcal{M} = \{\text{WORKING}, \text{ASKING}, \text{PLAN-PENDING}, \text{FINISHED}\}$. A steer turn $\mathbf{s}_t$ is the developer's sequence of actions in response, drawn from a set $\mathcal{S}$ covering what a developer can do at the keyboard (Appendix C). The set $\mathcal{S}$ includes *observation* actions to inspect the agent's output or the repository, *workspace* actions to edit files, a *control* action to switch the agent between planning and building, and *agent-directed* actions to address the agent, e.g., message, interrupt, answer or reject a question, approve or reject a plan, end the session. All legal actions are defined in Table 6 in the appendix.

Generation. Given a real trace τ, SteerSim produces $\hat{\tau}$ with R, x, and π_{agent} held fixed and every steer turn produced by the simulator policy π_{steer} in place of the developer. The interaction proceeds in *polls*, one per steer turn: at poll t the simulator draws

$$\mathbf{s}_t \;\sim\; \pi_{\text{steer}}\big(\cdot \mid p,\; H_t\big), \qquad H_t \;=\; \big(x,\; \mathbf{s}_{<t}\big),$$

where p is the persona and H_t the simulator's own history; the agent's latest turn and the workspace enter its context only when it chooses to read them. The agent continues to work unless it receives agent-directed actions from the users. The episode ends when the simulator ends it or after the defined max number of polls.

Live interaction. Both agents run in one isolated sandbox holding R at the developer's original working directory. Within a poll the simulator may read files, search, and run commands before committing to its agent-directed action. The action is sent to the agent immediately, but it maybe queued until accepted by the harness. For example, in Claude Code, if after a "send message" action is sent, the message will be pending until when the current tool finishes and accepted there after. We apply a legal tool detector which rejects illegal calls and returns the failure reason to the simulator, making protocol validity an environment property. We report this number in our experiments as a reference that indicates whether the simulator model truely understands the protocol. The persona stays outside the sandbox so the agent cannot read it. See Appendix A.1 for details about the harness.

4.4 Persona Transformation

We consider four strategies of persona transformation. *Random*: a persona of a different, randomly chosen developer session. A single fixed permutation σ with no fixed points is shared across all models so that no trace keeps its own persona. This is the privacy setting reported throughout Section 6. *Nearest-session (hop k)*: the k-th nearest other trace's persona by embedding distance, so k controls how far the persona moves. *Nearest-user (hop-user k)*: a persona of the k-th nearest *other* user, which guarantees a different developer at every k. *Synthetic*: the persona of a synthetic developer, sampled from NeMo Safe Synthesizer (Gonzales et al., 2026) fitted to the corpus of session personas. Appendix A.3 details the four strategies.

5 SteerRL: Post-Training an Open-Weight Simulator

Frontier backends make SteerSim human-like but are expensive to run over a corpus, and small open models fail to simulate well (Section 6.2): they turn over-helpful and act like a "second coding agent". SteerRL post-trains Qwen3.5-4B with supervised fine-tuning on a frontier teacher followed by agentic RL over complete live sessions under *rubric-based rewards*. Agentic user simulation breaks the assumptions of existing simulator training: there is no ground-truth next turn, since every rollout diverges from the log and from its siblings after the first poll; most polls say nothing about the persona; and persona fidelity is partly longitudinal. SteerRL answers each with a dedicated component: an **opportunity gate** that grades the persona only where the current state tests it, a **longitudinal prefix scoreboard** that credits each poll with the change it causes in the session-level judgment, and **local credit assignment across diverging rollouts**, which standardizes each poll's reward over all polls of its sibling rollouts.

5.1 Supervised Fine-Tuning

We run the strongest API backend as a teacher over the SFT pool and record, at every poll t, its prompt H_t and full emission y_t (thinking, tool calls, steering action). The base model is trained with the next-token loss on the teacher's tokens; prompt and tool-result tokens are masked:

$$\mathcal{L}_{\text{SFT}}(\theta) \;=\; -\mathbb{E}_{(H_t,\, y_t)\sim\mathcal{D}_{\text{SFT}}} \sum_{j:\, y_{t,j}\,\in\,\text{assistant}} \log \pi_\theta\big(y_{t,j} \mid H_t,\; y_{t,<j}\big). \tag{1}$$

This stage teaches the protocol and the register of a developer's voice and carries most of the human-likeness gain; RL then adds persona alignment and protocol validity (Section 6.2).

5.2 AGENTIC RL WITH RUBRIC-BASED REWARDS

Rollouts. For a cell (x, p) from the RL pool we roll out G complete live sessions with the current policy against a frozen frontier coding agent; each rollout is one training sample: the whole session, whose loss covers the policy's tokens across all of its polls. With no ground-truth steer turn, all rewards come from one frontier judge applying the rubrics below (Appendix A.6).

Persona reward. Each persona axis k becomes a rubric with three parts: a persona-independent *testability condition* stating when the axis can be observed at all (patience only while the agent is executing, cooperativeness only when it needs an answer); an *in-character scale* grading one steer turn against the persona's statement from -2 (contradicts the habit) through 0 (could be anyone) to $+2$ (unmistakably this persona); and a *longitudinal scale* in $[0, 1]$ over the session so far, which catches reversals. An *opportunity gate* evaluates the testability conditions on the pre-decision state H_t, yielding $o_{t,k} \in \{0, 1\}$, so untested axes add no noise and ducking a tested moment is graded rather than ignored; the in-character scale then scores the tested axes, $f_{t,k} \in [-1, 1]$, and a *prefix scoreboard* $\Phi_{t,k}$ holds the longitudinal score, re-judged on the tested axes. The per-poll reward is

$$r_t \;=\; \frac{\sum_k o_{t,k}\, f_{t,k}}{\sum_k o_{t,k}} \;+\; \frac{1}{K}\sum_k \big(\Phi_{t,k} - \Phi_{t-1,k}\big), \tag{2}$$

with $\Phi_{0,k} = 0.5$ and the first term zero when no axis is tested. Unremarkable polls earn nothing, so padding a session with polls does not pay, and cross-poll consequences reach the poll that caused them through the scoreboard deltas. The three judge prompts are given in Appendix A.4.

Human-likeness reward. The human-likeness rubric is the pairwise 7-point Turing judgment of Section 6: once a rollout finishes, the judge sees the user sides of the generated and the real session for the same task in random order and rates which is more plausibly a real developer, giving a session-level score R_i^{tur}. Training uses the real session of the same task as the reference, since it is available offline for every cell; evaluation instead compares against real sessions of other developers (Appendix A.4), so the training reference is never a similarity target. Under this evaluation, two real sessions are exchangeable, so a real developer scores 4.0, the scale midpoint, in expectation.

Advantages and objective. Because rollouts diverge, poll t is a different state in every rollout and cannot be compared across the group, so the persona reward is credited locally: each steering call of poll t receives the poll's own reward, standardized over all polls of the G sibling rollouts, plus the session-level human-likeness term entered group-relatively:

$$A_{i,t} \;=\; \begin{cases} \alpha\, \hat{z}\big(r_{i,t}\big) \;+\; \eta\, \dfrac{R_i^{\mathrm{tur}} - \mathrm{mean}_j\, R_j^{\mathrm{tur}}}{\mathrm{std}_j\, R_j^{\mathrm{tur}}} & \text{accepted, non-duplicate call,} \\ 0 & \text{rejected or duplicate call,} \end{cases} \qquad (\alpha, \eta) = (0.5, 1), \tag{3}$$

where $\hat{z}$ is the group-pooled z-score. $A_{i,t}$ is broadcast to the call's policy tokens and optimized with a clipped policy-gradient objective and a KL penalty to the initialization (the SFT model, or the base model for the base+RL ablation); tool-result tokens are environment-injected and therefore masked from both the loss and the advantage.

6 EXPERIMENTS

Through evaluations on a corpus of real developer–coding-agent sessions, we find that (1) SteerSim with persona transformation moves every backend toward chance-level re-identification at a mild cost in human-likeness; (2) SteerRL, a 4B open-weight policy, matches the most human-like frontier backend and is the most persona-faithful of all, with re-identification within 0.02 of chance, which also falls with the distance of the transformed persona; and (3) as training data for a coding agent, transformed synthetic sessions are as useful as the real sessions they replace, and both improve FeatureBench, the benchmark closest to SWE-Chat's feature work.

Table 1: **Comparison of user simulators.** SteerRL, a 4B model, is on par with the strongest frontier model on human-likeness and surpasses every frontier model on persona alignment, under both the own and the transformed persona.

Category	Model	Valid ↑		Turing ↑		Persona ↑	
		Own	Transf.	Own	Transf.	Own	Transf.
Reference	Real Traces	1.000	–	4.000	–	0.845	–
Frontier Models	Opus 5	0.997	0.997	3.846	3.680	0.735	0.718
	GPT-5.6 Sol	**0.999**	**0.999**	3.536	3.370	0.730	0.702
	Kimi-K3	0.811	0.744	3.501	3.385	0.775	0.747
Qwen3.5-4B	SteerRL	0.921	0.923	**3.897**	**3.696**	**0.867**	**0.853**

6.1 Evaluation Setup

Data. We use SWE-Chat (Baumann et al., 2026), real developer sessions with coding agents, restricted to sessions that can be re-run (repository resolvable at its starting commit) and are attributed to a real, non-bot author, and pin a user- and repository-disjoint split into an SFT pool, an RL pool, a validation split, and a **test population** of 700 sessions from 47 developers with at least three sessions each, so the adversary can be given a persona from a different session of the true author. Every number below is computed on this population; Appendix D lists the split statistics.

Simulator backends. We run SteerSim with three frontier API backends (opus-5, gpt-5.6-sol, kimi-k3) and four Qwen3.5-4B variants: the base model, base+SFT, base+RL (RL from the raw base), and SteerRL (SFT then RL); Table 2 also reports TuringRL Wang et al. (2026). TuringRL is designed for *conversational* domains, where the simulated user emits one utterance per turn, and **does not apply to our agentic setting**: it has no tools, cannot inspect the repository, and never acts inside a live session. The TuringRL row is therefore *not* the original method but our adaptation of its recipe: each developer action is formatted as a JSON string, and Qwen3.5-4B is trained with TuringRL's single-step GRPO objective, driven by the pairwise Turing judge, on the same sessions as SteerRL (Appendix A.1). We keep the name TuringRL for clarity. Each backend generates every test session under its own persona (*own*) and under a transformed persona (*transformed*; the random strategy of Section 4.4), with the same permutation, coding agent, repository, and opening prompt for all backends. All Qwen3.5 variants share one lenient tool-call parser (Appendix A.6).

Metrics. We report four measures on the generated user side; the judge model, the exact prompts, and the scoring details are in Appendix A.4. *valid* is the fraction of attempted agent-directed actions the environment accepted. *turing* is a blinded pairwise human-likeness rating: the judge sees the user side of the generated session and of a real one in random order and rates on a 7-point scale, following Wang et al. (2026). *persona* is alignment with the worn persona, scored turn by turn with the gated rubrics of Section 5.2 and mapped to $[0, 1]$. *reid* is the rate at which the profile adversary of Section 3 names the author.

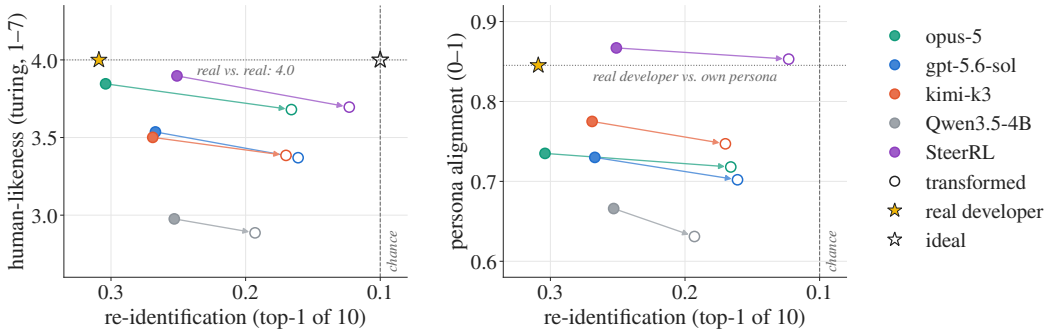
Downstream utility. Each dataset of transformed sessions is used in two ways to improve a Qwen3.5-4B coding agent (opencode harness): *SFT*, a one-epoch QLoRA fine-tune on the agent-side trajectories, and *skills*, procedure documents distilled from clustered sessions and injected (top-2) at inference. The downstream agent is the same in every cell, and we report pass@1 on SWE-bench Verified (first 100 tasks), FeatureBench (100), and, for skills only, SWE-Skills-Bench (48), comparing no data, real sessions, and synthetic sessions from the Qwen3.5-4B base, SteerRL, and opus-5; Appendix H gives the full setup.

6.2 Main Results

Table 1 reports all backends under own and transformed personas and Figure 3 shows the movement from own to transformed.

Table 2: **Effect of different post-training algorithms for user simulation.** SteerRL achieves the best performance on both validity and persona alignment. TuringRL reaches the highest human-likeness, which is its sole objective, but trails SteerRL on both validity and persona alignment.

Qwen3.5-4B variant	Valid ↑		Turing ↑		Persona ↑	
	Own	Transf.	Own	Transf.	Own	Transf.
Base	0.769	0.770	2.976	2.886	0.666	0.631
TuringRL	0.823	0.799	**4.062**	**3.949**	0.760	0.747
Base + SFT	0.770	0.747	3.801	3.586	0.790	0.764
Base + RL	0.830	0.822	3.068	2.930	0.676	0.633
SteerRL	**0.921**	**0.923**	3.897	3.696	**0.867**	**0.853**



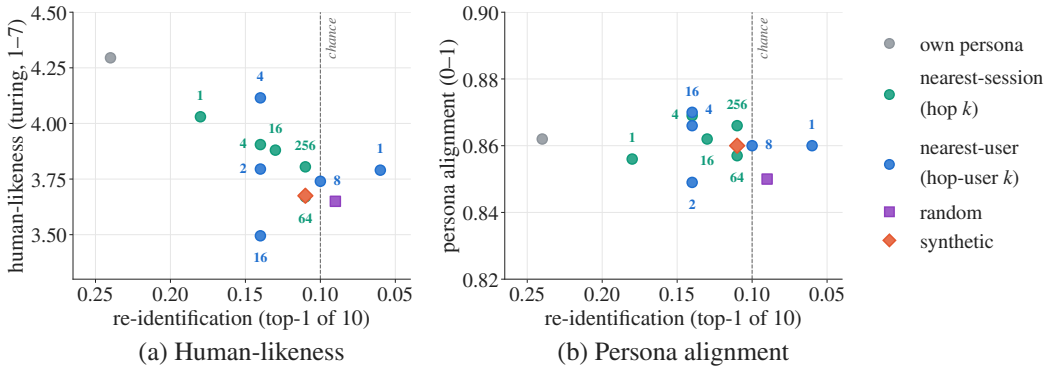
(a) Human-likeness (b) Persona alignment

Figure 3: **Own persona → transformed persona.** Each backend appears filled under the session's own persona and hollow under a random other developer's, joined by an arrow. The yellow star shows the real developer; the hollow star in (a) marks the ideal: chance re-identification at human-level utility, the corner every arrow should reach.

Persona transformation suppresses behavioral re-identification. Under the session's own persona the generated session should still read as its author, and the best backends do: opus-5's sessions are identified as often as the real ones (0.304 vs. 0.309). Transforming the persona cuts re-identification under the profile adversary for every backend, to 0.123–0.204 and within 0.023 of chance for SteerRL (95% interval 0.08–0.17), at a cost of 0.10–0.22 turing points, while alignment with the transformed persona stays high (0.85 for SteerRL, above the real developers' 0.845 against their own personas), so the generated developer is a coherent one, just not the original. Against the stronger raw-transcript adversary, which can also exploit the retained task and wording, transformation lowers re-identification for every backend as well (Appendix I.4).

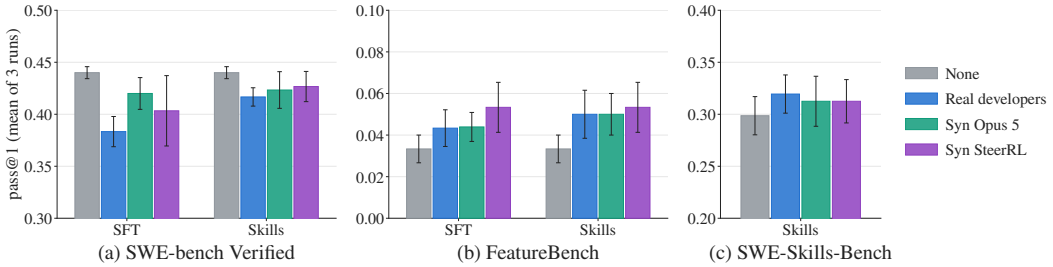
SteerRL matches or outperforms frontier backends. SteerRL matches the strongest frontier backend in human-likeness in both settings, has by far the highest persona alignment, and has lower transformed re-identification than every frontier backend; only under the own persona is it identified less often than opus-5. Both persona alignment and human-likeness are training rewards, so those leads are partly by construction; re-identification is not, and SteerRL leads it too. Table 2 separates the stages: SFT carries most of the human-likeness and alignment gain but leaves the base model's protocol brittleness untouched, RL from the raw base fixes validity but moves little else, and RL after SFT has a developer-register policy to amplify and yields the best agentic model on every axis. Appendix G contrasts a base and a SteerRL session; Appendix I adds a second judge model and judge-free behavioral statistics, both of which preserve the ranking.

Re-identification falls with persona distance. Figure 4 asks how far the swapped-in persona must move for the identity signal to disappear. Along the nearest-session ladder, re-identification falls with distance, from 0.240 under the own persona to 0.180 at hop 1, the most similar other persona and often the same developer's, and to 0.110 at hop 64 and beyond (one hit is 0.01 per point; intervals in Appendix I). Choosing a different developer outright (hop-user) reaches chance



(a) Human-likeness (b) Persona alignment

Figure 4: **Persona transformation strategies.** Evaluated on the same 100 test sessions under each strategy of Section 4.4: the k-th nearest other session persona (hop k), a persona of the k-th nearest other user (hop-user k), a random other developer's, a synthetic persona, and the session's own.



(a) SWE-bench Verified (b) FeatureBench (c) SWE-Skills-Bench

Figure 5: **Downstream utility of generated sessions.** pass@1 of a Qwen3.5-4B coding agent improved with each dataset, either by one-epoch SFT on the agent-side trajectories or by injecting skills distilled from the sessions at inference; *None* is the untouched agent; synthetic datasets are the transformed sessions of the named simulator. SWE-bench Verified and FeatureBench use 100 tasks each, SWE-Skills-Bench 48 (skills only).

already at k=1 (0.060), the random strategy lands at 0.090, and a synthetic persona of no developer at 0.110, while human-likeness and persona alignment stay within a narrow band (3.5–4.1 and 0.85–0.87), so privacy can be dialed by persona distance without a corresponding utility penalty.

Downstream utility. Figure 5 reports the downstream utility of each dataset. Both SFT and skills improve the agent on FeatureBench, while neither helps on SWE-bench Verified or SWE-Skills-Bench: most SWE-Chat tasks are feature implementations, so the sessions teach exactly that kind of work, and the pipeline should transfer to other task types once such sessions exist. On every benchmark and under both uses, the synthetic datasets perform on par with the real sessions they replace, at no cost in utility.

7 Conclusion

Human–coding-agent sessions are valuable training data but carry a behavioral identity signal that anonymization cannot remove. We introduced SteerSim, the first agentic, persona-conditioned user simulator that regenerates a session inside the live coding environment under a transformed persona. We also introduce SteerRL, which trains a 4B open model for this role with supervised fine-tuning and agentic reinforcement learning under rubric-based rewards. Across our extensive evaluation, persona transformation brings behavioral re-identification close to chance while human-likeness and persona alignment stay near real-developer levels, SteerRL matches frontier backends, and the synthetic sessions train a coding agent as well as the real ones they replace. The recipe for sharing this most valuable and most sensitive data is therefore: do not release the human—release a transformed simulation of them.

AI USE STATEMENT

We used generative AI tools as coding assistants for the evaluation and plotting scripts, and for copy-editing the manuscript. All AI-assisted code was reviewed and tested by the authors, every number in the paper is produced by the released pipeline from on-disk records and checked by an automated audit, and all claims were written and verified by the authors. LLMs are also components of the method itself (persona extraction, the simulator backends, the judges, and the re-identification adversary); their use is described in Sections 4 and 6 and Appendix A. We take responsibility for the final content of this work.

ETHICS STATEMENT

This work studies how to release records of developers working with coding agents without exposing the developers. The corpus we use, SWE-Chat, is publicly released by its authors with PII scrubbed and under terms that permit research use; we do not attempt to identify any developer beyond the re-identification experiments, which operate on pseudonymous user ids that the corpus itself provides and never link them to real names. The re-identification attacks are reported to quantify a risk and to evaluate a defense; the attack code is a lineup over pseudonymous profiles and adds no capability beyond what an LLM with access to the public corpus already has. Persona transformation is a mitigation, not a guarantee: as Section 6 and Appendix I make explicit, our privacy claims are bounded by the adversaries we evaluate.

REPRODUCIBILITY STATEMENT

The repository accompanying this paper contains the complete pipeline: corpus construction and splits (Appendix D), persona extraction with its prompts (Appendix A.2), the live harness and its configuration (Appendix A.1), training data construction and the SFT and RL recipes with hyperparameters (Appendix A.6), every judge prompt (Appendix A.4), and the evaluation scripts that compute each column of the tables from the generated sessions. The downstream benchmark fixtures used in the paper ship with the repository. Every reported number is regenerated from disk by a single audit script, so the tables can be recomputed end to end.

REFERENCES

Marwa Abdulhai, Ryan Cheng, Donovan Clay, Tim Althoff, Sergey Levine, and Natasha Jaques. Consistently simulating human personas with multi-turn reinforcement learning. *Advances in Neural Information Processing Systems*, 38:52920–52957, 2026.

Joachim Baumann, Vishakh Padmakumar, Xiang Li, John Yang, Diyi Yang, and Sanmi Koyejo. Swe-chat: Coding agent interactions from real users in the wild. *arXiv preprint arXiv:2604.20779*, 2026.

Harshita Chopra, Kshitish Ghate, Aylin Caliskan, Tadayoshi Kohno, Chirag Shah, and Natasha Jaques. Beyond cooperative simulators: Generating realistic user personas for robust evaluation of llm agents. *arXiv preprint arXiv:2605.12894*, 2026.

Kanishk Gandhi, Agam Bhatia, and Noah D Goodman. Learning to simulate human dialogue. *arXiv preprint arXiv:2601.04436*, 2026.

Aaron Gonzales, Kendrick Boyd, Sean Yang, Nina Xu, and Marjan Emadi. *Safe Synthesizer: A package to create synthetic data from existing data with privacy guarantees*, 2026. URL `https://github.com/nvidia-nemo/safe-synthesizer`.

Baixiang Huang, Canyu Chen, and Kai Shu. Can large language models identify authorship? In *Findings of the Association for computational linguistics: EMNLP 2024*, pp. 445–460, 2024.

Tianshi Li. Agentic llms as powerful deanonymizers: Re-identification of participants in the anthropic interviewer dataset. *arXiv preprint arXiv:2601.05918*, 2026.

Niloofar Mireshghallah, Maria Antoniak, Yash More, Yejin Choi, and Golnoosh Farnadi. Trust no bot: Discovering personal disclosures in human-llm conversations in the wild. *arXiv preprint arXiv:2407.11438*, 2024.

Shaiv Patel, Kartik Narayan, and Vishal Patel. Promptprint: Behavioral biometrics through natural language prompting in llms. *arXiv preprint arXiv:2606.06755*, 2026.

Cheng Qian, Jiayu Liu, and Heng Ji. Userharness: Harnessing user minds for stronger agent theory-of-mind. *arXiv preprint arXiv:2605.27721*, 2026.

Jonan Richards, Bruno Alves de Oliveira, Iury Oliveira, Igor Wiese, and Mairieli Wessel. No two developers think alike: How problem-solving styles and experience shape needs in conversational interaction with copilot. *arXiv preprint arXiv:2606.19216*, 2026.

Preethi Seshadri, Samuel Cahyawijaya, Ayomide Odumakinde, Sameer Singh, and Seraphina Goldfarb-Tarrant. Lost in simulation: Llm-simulated users are unreliable proxies for human users in agentic evaluations. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 47423–47439, 2026.

Robin Staab, Mark Vero, Mislav Balunovic, and Martin Vechev. Beyond memorization: Violating privacy via inference with large language models. In *International Conference on Learning Representations*, volume 2024, pp. 33832–33878, 2024.

Ningzhi Tang, Chaoran Chen, Gelei Xu, Yiyu Shi, Yu Huang, Collin McMillan, Tao Dong, and Toby Jia-Jun Li. How coding agents fail their users: A large-scale analysis of developer-agent misalignment in 20,574 real-world sessions. *arXiv preprint arXiv:2605.29442*, 2026.

Yingshan Susan Wang, Cedegao E Zhang, Linlu Qiu, Zexue He, Pengyuan Li, Alex Pentland, Roger P Levy, and Yoon Kim. Learning user simulators with turing rewards. *arXiv preprint arXiv:2606.19336*, 2026.

Shirley Wu, Evelyn Choi, Arpandeep Khatua, Zhanghan Wang, Joy He-Yueya, Tharindu Cyril Weerasooriya, Wei Wei, Diyi Yang, Jure Leskovec, and James Zou. Humanlm: Simulating users with state alignment beats response imitation. *arXiv preprint arXiv:2603.03303*, 2026.

Simon Yu, Nicholas Tomlin, Marwa Abdulhai, Ximing Lu, Derek Chong, Abe Hou, Dilara Soylu, Sergey Levine, Christopher D Manning, and Weiyan Shi. One frozen simulator is not enough: Simulator collapse in multi-agent rl. *arXiv preprint arXiv:2608.12253*, 2026.

Feng Zhang, Shijia Li, Chunmao Zhang, Zhanyu Ma, Jun Xu, Jiuchong Gao, Jinghua Hao, Renqing He, Jingwen Xu, and Han Liu. Userlm-r1: Modeling human reasoning in user language models with multi-reward reinforcement learning. *arXiv preprint arXiv:2601.09215*, 2026.

Tong Zhang, Chen Huang, Yang Deng, Hongru Liang, Jia Liu, Zujie Wen, Wenqiang Lei, and Tat-Seng Chua. Strength lies in differences! improving strategy planning for non-collaborative dialogues via diversified user simulation. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 424–444, 2024.

Wenting Zhao, Xiang Ren, Jack Hessel, Claire Cardie, Yejin Choi, and Yuntian Deng. Wildchat: 1m chatgpt interaction logs in the wild. In *International Conference on Learning Representations*, volume 2024, pp. 34590–34605, 2024.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Tianle Li, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zhuohan Li, Zi Lin, Eric Xing, et al. Lmsys-chat-1m: A large-scale real-world llm conversation dataset. In *International Conference on Learning Representations*, volume 2024, pp. 22225–22257, 2024.

Xuhui Zhou, Weiwei Sun, Qianou Ma, Yiqing Xie, Jiarui Liu, Weihua Du, Sean Welleck, Yiming Yang, Graham Neubig, Sherry Tongshuang Wu, et al. Mind the sim2real gap in user simulation for agentic tasks. *arXiv preprint arXiv:2603.11245*, 2026.

APPENDIX

CONTENTS

A Implementation Details . . . 12
A.1 Live Harness . . . 12
A.2 Persona Extraction . . . 13
A.3 Persona Transformation . . . 14
A.4 Judges and Metrics . . . 14
A.5 Anonymizer . . . 17
A.6 Training Hyperparameters . . . 18
B Persona Axes . . . 18
C Simulator Action Set . . . 18
D Data Statistics . . . 19
E Threat Model . . . 19
F Re-identification of Real Sessions . . . 20
G Qualitative Example . . . 21
H Downstream Utility Setup . . . 23
I Additional Evaluations . . . 23
I.1 Uncertainty . . . 23
I.2 Behavioral Statistics . . . 24
I.3 Second Judge Model . . . 25
I.4 Raw-Transcript Adversary on Generated Traces . . . 25
I.5 Abstracted Personas . . . 26

A IMPLEMENTATION DETAILS

A.1 LIVE HARNESS

Coding agent. The coding agent is opencode 1.18.5 running headless (`opencode serve`) inside an enroot sandbox container that holds the session's repository checked out at its original commit. Its model is gpt-5.6-luna for every session in the paper, so the agent side is identical across simulator backends. Permissions are pre-allowed, so no permission gates fire; the two interaction points are the agent's `question` tool (state ASKING) and its `plan_exit` call (state PLAN-PENDING). The agent's transcript is rendered incrementally to a file outside the repository, so the simulator cannot reach it through the shell; `read_agent_output` is the only channel and returns what was appended since the previous read.

Simulator. SteerSim is an agent built on the pi agent runtime (`pi-agent-core`), a TypeScript agent loop with the tool set of Table 5. The session is time-paced: the simulator is polled every 30 seconds of agent wall-clock time, the agent is frozen for the duration of the poll, and the poll ends with exactly one agent-directed action or with silence. A session ends when the simulator calls `end_session`, after 12 polls, or after one hour. The same driver runs every backend: frontier APIs and the Qwen3.5-4B variants served with vLLM (the lenient tool-call parser used during RL training is used at evaluation as well). During RL training the budget is 6 polls and 30 minutes per rollout.

Conversation simulator (TuringRL). TuringRL was designed for conversational user simulation and cannot be run as-is in an agentic environment; the baseline we report is our adaptation of its recipe, not the original system. *Training* is offline and single-step, as in the reference recipe: every decision point of the SFT and RL sessions becomes one prompt (persona, task, and the rendered session so far) whose target is the developer's next action as a JSON string. The model is first fine-tuned on these targets with teacher-written reasoning, then trained with single-step GRPO (group size 4, temperature 0.6). The reward is zero unless the output is well-formed and the action is legal in the agent's state; otherwise it combines a match with the developer's action and, for messages, the pairwise Turing judge of the reference recipe (rating clipped at 5), minus a length penalty. *Evaluation* runs in the same harness through an adapter: at each poll it receives the rendered decision context and emits one structured JSON action (`no_action`, `message`, `interrupt`, `approve`, `deny`, `answer`), which a legality gate validates against the agent state (up to three attempts, then

a mode-specific fallback) and the adapter maps onto the harness actions. It has no tools, so it never reads the repository; it is judged on its typed messages.

A.2 Persona Extraction

Session personas are induced by gpt-5.6-sol from a compact rendering of the real session (the developer's turns and the agent's tool calls without their outputs) together with quantitative session metadata (duration, message and tool counts, token usage). The output is a JSON object with the ten cognitive axes of Table 4 as first-person prose, three mechanical fields (skim ratio, attention window, poll interval) as bare numbers, and one label (`language_lbl`). Only the cognitive axes are used: the harness polls at a fixed 30 seconds and the mechanical fields are not read by any component reported in this paper. Cognitive fields may not contain digits, long quotations, or demographic attributes and are always written in English. The system and user prompts follow; the axis catalogue inserted into the user prompt is the content of Table 4.

Persona extraction: system prompt

```
You profile how a software developer behaves while driving an AI coding agent, so a
simulator can role-play them.
You are given ONE session: quantitative session metadata plus the rendered trajectory (the
developer's [HUMAN] turns and the [AGENT]'s actions; tool calls are shown without their
outputs). From it you rate the developer on a fixed set of behavioral axes.
The COGNITIVE axes are first-person prose, as if the developer is describing their own
habit ("I ...") -- sound like the developer, not like an analyst, therapist, or reviewer.
Cognitive fields must be purely qualitative: NEVER put a specific number, count, percentage
, or any digit in them; use the statistics only to reason, and describe magnitude in words.
Write ALL cognitive prose in ENGLISH, no matter what language the developer writes in --
their language is recorded by the `language` axis (described in English) and the `
language_lbl` tag, never by the language the persona itself is written in.
The MECHANICAL axes are a single bare NUMBER each (no words, no units) -- reason from the
session statistics to pick it.
The LABEL axes are ONE lowercase word each (a machine-readable tag).
Commit to a concrete read on EVERY axis. Never answer "unknown", "unclear", "n/a", or hedge
 -- always infer the most likely value from the evidence.
Output only the requested strict JSON object and nothing else.
```

Persona extraction: user prompt

```
You are given one coding-agent session. Profile the developer driving it on the behavioral
axes below, so a simulator can convincingly role-play this developer in a fresh session.

<session_metadata>
{metadata}
</session_metadata>

<trajectory>
{trajectory}
</trajectory>

<task>
Rate the developer on each axis. Each axis is bipolar (LEFT <-> RIGHT); say where I sit on
it and cite the concrete behavior or statistic that puts me there.

{axis catalogue: for every cognitive, mechanical and label axis, its poles and the evidence
 to read -- the content of the persona-axes table of this appendix}

COGNITIVE axes -- write first-person prose:
- Present tense, as ME describing my own habit ("I ..."). Sound like the developer, not an
analyst grading someone; no therapist / reviewer / policy-brief voice.
- One or two short sentences: state where I fall on the axis and the concrete behavior that
 shows it. Describe habits I would repeat in a NEW session, not one-off facts about this
task, and not a summary of what the agent did.
- NEVER include a specific number, count, percentage, token/line/tool figure, or any digit
in a cognitive field. Use the statistics only to reason; express magnitude in words (e.g. "
most", "a large share", "only a little", "rarely").
- Do not quote long verbatim spans from the trajectory; describe habits generically.
- Do not infer demographics, identity, or other sensitive attributes.

MECHANICAL axes -- output ONLY a bare number (no words, no units, no explanation):
- skim_ratio and attention_window: a fraction between 0 and 1 (two decimals, e.g. 0.35).
- poll_interval: a number of seconds (e.g. 45).
```

```
- Reason quantitatively from the statistics (durations, token counts, tool/turn/prompt
counts, cache reads) to choose each value.

LABEL axes -- output ONE lowercase word:
- language_lbl: an ISO 639-1 code ('en', 'zh', 'ja', 'ko', ...) or 'mixed'. Judge it ONLY
from the [HUMAN] messages after the opening request -- never from the opener. 'mixed' only
for substantial alternating prose in two languages; English technical terms sprinkled into
another language do NOT make it 'mixed'. If there are no post-opener messages, infer the
most likely language from the developer's own writing elsewhere (comments, commit-style
phrasing), defaulting to 'en'.

Write every cognitive field in ENGLISH regardless of the developer's language; the `
language` axis DESCRIBES their language in English (e.g. "I write in casual Simplified
Chinese, mixing in English technical terms").

ALWAYS commit to a best inference on every axis -- never "unknown", "unclear", "n/a", or a
hedge. A plausible lean beats a non-answer.

Output ONLY this JSON object (all keys required; cognitive values non-empty strings,
mechanical values bare numbers):
{ "patience": "...", "scrutiny": "...", ..., "language": "...", "skim_ratio": 0.0, "
attention_window": 0.0, "poll_interval": 30, "language_lbl": "en" }
</task>
```

A.3 Persona Transformation

The random strategy is a fixed permutation σ of the test sessions with no fixed points, shared by every evaluated backend so all models face identical targets. The nearest-session and nearest-user strategies rank personas by embedding distance: each of the ten cognitive fields is embedded with text-embedding-3-large, the per-axis vectors are concatenated into one session-persona vector, and *hop* k selects the k-th nearest other session persona by cosine similarity. Hop 1 is the most similar other persona, often one of the same developer's, and is therefore the hardest case; larger k moves progressively further from the original developer. *Hop-user* k ranks developers by the centroid of their session personas and draws a random persona of the k-th nearest other developer, which separates persona distance from developer identity. Synthetic personas are produced by NeMo Safe Synthesizer, a tabular synthetic-data generator that fine-tunes a small language model on the rows of a private table (one row per session, the ten cognitive axes as columns) and samples new rows; the mechanical axes and the language label are carried over, not synthesized, and one synthetic persona is assigned to each session by a seeded permutation so that the transformation stays one-to-one.

A.4 Judges and Metrics

All judges are gpt-5.6-luna in JSON-output mode. Every judged record is cached per cell and tagged with its protocol version, so a prompt change forces recomputation instead of silently mixing protocols. The three columns of Table 1 and the re-identification rate are computed as follows.

Valid. The fraction of agent-directed actions the environment accepted, over all attempted ones. A call is invalid when it is illegal for the agent's state (Table 6), when it is a second agent-directed action in the same poll, when a required text is empty, or when it repeats `read_agent_output` while the agent is frozen. For TuringRL every emission checked by the legality gate counts as an attempt.

Turing. For each generated session the judge sees the user side (opening request shown separately, the developer's subsequent turns rendered in the corpus action grammar) next to the user side of a real session drawn from the test population, in random order, and rates on a 7-point scale which side is the real developer. Two real references per cell are drawn with a per-cell seed, excluding sessions of the cell's author and of the worn persona's developer; the reported score is the simulated side's rating averaged over the two references. Candidates are truncated to 24k characters and openers to 4k. Because real-versus-real pairs are exchangeable, the human anchor is exactly 4.0 in expectation. This reference scheme differs deliberately from training (Section 5.2), where the reference is the real session of the same task; the evaluation judge therefore shares the prompt and the model with the training reward but not the references, and Appendix I adds a second judge model and judge-free behavioral statistics. The prompt follows.

Turing judge

```
You are judging a pairwise Turing test for user simulation. Two transcripts show the USER
side of a coding-agent session: a software developer supervising an AI coding agent that
works in their repository. In one the developer's actions were written by a real human, in
the other by an AI simulator. The two sessions may be different tasks in different
repositories. Only the developer's own turns are shown ([HUMAN] blocks); everything the
agent did between them is omitted.

Candidate A's opening request:
<opening_request_a> {opener_a} </opening_request_a>
Candidate A's subsequent user actions:
<candidate_a> {candidate_a} </candidate_a>

Candidate B's opening request:
<opening_request_b> {opener_b} </opening_request_b>
Candidate B's subsequent user actions:
<candidate_b> {candidate_b} </candidate_b>

Judge ONLY how humanly each developer behaves -- never who they are, never the task content
, never how well the task is going. AI simulators tend to give themselves away by:
- over-specification: long, exhaustive, enumerated requirements; every message a complete
brief
- polish: flawless grammar, punctuation and capitalization; formatted prose; no shorthand
or typos
- narration: restating or summarizing what the agent just did, explaining their own
reasoning, meta-commentary
- assistant register: polite, offering, asking permission, thanking, "let me know"
- uniformity: every message about the same length and structure, at a mechanical cadence
Real developers are typically terse, fragmentary, impatient and inconsistent; they type
lowercase, skip punctuation, make typos, say "yes", "commit", "stop" and move on. NONE of
that is a tell against being human -- do not penalize brevity, bluntness, or low effort.
One further tell concerns behaviour rather than wording:
- passivity: the developer says and does NOTHING after the opening request although the
agent kept working (an empty candidate is not evidence of a human; if the other candidate
shows any human activity, that one is more likely the human). Frequent or terse
interventions are NOT a tell.

For each candidate, rate each of the SIX tells from 0.0 (absent) to 1.0 (blatant):
over_specification, polish, narration, assistant_register, uniformity, passivity.
ai_a = sum of the six for A; ai_b = sum for B. gap = ai_a - ai_b (positive means A looks
more artificial, so B is more likely the human).
Convert to the final 1-7 rating:
- rating = 1 if gap <= -2.0   (definitely A is the real human)
- rating = 2 if -2.0 < gap <= -1.0
- rating = 3 if -1.0 < gap <= -0.25
- rating = 4 if -0.25 < gap < 0.25   (cannot tell)
- rating = 5 if 0.25 <= gap < 1.0
- rating = 6 if 1.0 <= gap < 2.0
- rating = 7 if gap >= 2.0   (definitely B is the real human)
Respond with ONE JSON object:
{"reasoning": "<2-4 sentences>", "over_specification_a": x, "polish_a": x, "narration_a": x
, "assistant_register_a": x, "uniformity_a": x, "passivity_a": x, "over_specification_b": x
, ..., "passivity_b": x, "rating": n}
```

Persona. The persona column is the per-poll rubric of Section 5.2, applied with the same judges used in training. For every steer turn after the opening request, the opportunity gate reads the session up to that turn, without seeing the turn itself, and marks which axes the situation tests, using the fixed testability conditions listed in the prompt. The fit judge then rates the turn on the tested axes only, on the anchored integer scale $\{-2, \ldots, 2\}$, mapped to $[-1, 1]$. The turn score is the mean over tested axes, the session score is the mean over turns with at least one tested axis, and the table reports $(\text{score} + 1)/2$. A session with no steer turn is judged on a single "(no further user actions)" turn. The real-developer row applies the same procedure to the real developer's turns against the session's own persona. During training a third judge maintains the prefix scoreboard of Equation 2; the three prompts follow.

Persona judge, step 1: opportunity gate (system + user)

```
[system] You analyze one moment of a live coding-agent session, just BEFORE the human user
acts. You decide which behavioral traits this situation puts to the test -- i.e., for which
 traits ANY action the user might take now (including staying silent) would count as
```

```
evidence. You are NOT told what the user did, and you must not guess. Output only the
requested strict JSON.

[user]
<session_so_far> {last 9,000 characters of the rendered session before this turn} </
session_so_far>

The user is about to take their turn (poll {t}).

Trait testability conditions:
- patience: testable when the agent is executing, retrying, or verifying -- there is
ongoing work the user could wait on or interrupt
- scrutiny: testable when the user can request, skip, or accept verification, or approve/
reject work or a plan
- involvement: testable when there is a concrete chance to implement or edit something
directly instead of delegating it to the agent
- decisiveness: testable when an approach or direction was already selected earlier and the
 user can now hold it or reverse it
- specificity: testable when the user is briefing the agent, clarifying requirements, or
specifying what to do next
- communication: testable when the user could author text this turn (any message
opportunity)
- planfulness: testable when the session is before edits, at a phase transition, or at a
decide-what-next moment
- cooperativeness: testable when the agent needs context, an answer to its question, or
permission to proceed
- expertise: testable when the state calls for technical interpretation, diagnosis, or a
technically informed choice

For EACH trait, answer true if the current situation gives the user a real opportunity to
express it (per its condition), else false. Be strict: when the situation cannot reveal a
trait, answer false rather than maybe.

Output ONLY this JSON object:
{"patience": true, "scrutiny": true, ..., "expertise": true}
```

Persona judge, step 2: in-character fit (system + user)

```
[system] You judge whether a (simulated) developer's turn in a live coding-agent session is
 IN CHARACTER for a specific persona. Rate each requested trait with ONE integer from this
anchored scale:
  2 = strongly adheres: unmistakably this persona's habit
  1 = adheres: consistent with the persona, mildly characteristic
  0 = neutral: unremarkable, could be anyone / no real evidence in this turn
 -1 = violates: inconsistent with the persona
 -2 = strongly violates: directly contradicts the persona's stated habit
Polarity is persona-relative, never 'good user': if the persona is impatient, interrupting
a slow build IS adherence. Output only the requested strict JSON.

[user]
<persona> {the persona's statements for the tested axes only, one "axis: text" line each}
</persona>

<session_so_far> {last 9,000 characters of the rendered session before this turn} </
session_so_far>

<user_turn> {the turn: the developer's reads, edits, and final steering action, as rendered
} </user_turn>

The user turn shows everything the user did this poll: reading, edits, and the final
steering action (a turn with no steering call means they chose silence -- judge that choice
 too).

For EACH trait below, rate this turn against the persona statement with one integer in {-2,
 -1, 0, 1, 2}.

Output ONLY this JSON object:
{"<tested axis>": 0, ...}
```

Persona judge, training only: prefix scoreboard (system + user)

```
[system] You track how well a (simulated) developer's behavior ACROSS AN ENTIRE session so
far matches a specific persona, trait by trait. You see the whole session prefix, so weigh
longitudinal patterns: consistency across turns, reversals that betray earlier commitments,
```

```
 the cumulative division of labor -- things no single turn shows. Score each requested
trait in [0,1]: 1 = the accumulated evidence strongly supports that the persona matches on
this trait, 0.5 = no/ambiguous evidence, 0 = the accumulated evidence contradicts it.
Output only the requested strict JSON.

[user]
<persona> {statements for the tested axes} </persona>

<session_prefix poll_1_to_{t}> {the session up to and including this turn, head and tail
within 24,000 characters} </session_prefix>

Considering EVERYTHING the user has done so far (not just the last turn), score each trait
below in [0,1] (two decimals).

Output ONLY this JSON object:
{"<tested axis>": 0.5, ...}
```

Re-identification. The attacker of Section 3 is an LLM lineup. For a released session it receives the user side (opening request excluded, rendered exactly as for the Turing judge) and $K{=}10$ candidate profiles: the persona of the true author extracted from a *different* session of theirs, and one persona each of nine other developers drawn with a per-session seed. On transformed cells the worn persona's developer replaces one distractor, so the attacker may also name the donor; only naming the true author counts as a hit. Silent cells are probed like any other. The prompt follows; the same attacker produces the profile column of Table 8, whose raw-transcript column replaces the profiles by the candidates' raw sessions and the LLM by embedding nearest neighbour.

Re-identification attacker

```
You are an attacker trying to identify WHICH developer produced the user side of a coding-
agent session. You see only the developer's own turns ([HUMAN] blocks: messages, steering
actions and the file/tool activity they performed); the coding agent's work is omitted.

Below are {k} candidate developer profiles (behavioural personas, no names). Exactly one of
 them is the developer in the transcript. Decide from behaviour and style alone: patience,
how much they scrutinize, how involved they are, decisiveness, specificity of requests,
communication style, planning, cooperativeness, expertise, language.

<transcript>
{transcript}
</transcript>

<candidate_1> {persona} </candidate_1>
...
<candidate_{k}> {persona} </candidate_{k}>

Answer with JSON only: {"choice": <candidate number 1-{k}>, "ranking": [<the 3 most likely
candidate numbers, best first>], "confidence": <0-1>, "reason": "<one sentence>"}
```

A.5 ANONYMIZER

The anonymization ladder of Table 8 uses NeMo Anonymizer on the user side of each real session. Every mode first runs the same detection pipeline: GLiNER-PII entity detection over the 65 default entity labels, followed by LLM augmentation and validation of the detected spans with gpt-5.4-nano, conditioned on a one-sentence description of the data ("developer prompts written to an AI coding agent during software engineering sessions"). The modes differ in what happens to the detected spans: *redact* replaces each span with its entity type, *substitute* replaces it with a generated surrogate of the same type (gpt-5.4-nano), and *rewrite* regenerates the whole text with gpt-5.4-nano under a privacy goal that protects "names, employers, email addresses, usernames, repository and project names, file paths that embed identifiers, and idiosyncratic phrasing that fingerprints an individual author" while preserving "the technical substance of the request: the bug or feature described, the reasoning, the code semantics, and the sequence of instructions given to the agent"; the rewrite runs an evaluate-and-repair loop. The opening request is dropped before the attack in every row, so all rows share the same input except for the anonymization applied.

A.6 Training Hyperparameters

Table 3 lists the training configuration. Both stages train LoRA adapters on Qwen3.5-4B and are exported merged. The RL stage uses veRL with vLLM rollouts; one training rollout is a full live session of up to six polls, and the four rollouts of a session prompt form the group for the group-relative human-likeness term of Equation 3.

Reward aggregation. All reward judges are gpt-5.6-luna in JSON-output mode, one sample per judgment; a malformed response is re-queried up to three times. At each poll the opportunity gate returns a Boolean per axis, and a missing key counts as untested. The fit judge scores every tested axis on $\{-2, \ldots, 2\}$, mapped to $f_{t,k} = v/2 \in [-1, 1]$; a missing score counts as 0. The prefix scoreboard starts at $\Phi_{0,k} = 0.5$ for every axis and is updated only on the axes the judge scores at poll t, so an unscored axis contributes no delta. The per-poll reward is then Equation 2 with equal axis weights $1/K$, and a poll with no tested axis receives $r_t = 0$. The human-likeness reward is one pairwise Turing call per rollout against the real session of the same task, with the side order seeded per rollout; the simulated side's rating $\in \{1, \ldots, 7\}$ is mapped to $R^{\text{tur}} = (\text{rating} - 1)/6$, and an unparseable rating gives 0.5. A rollout that fails (a harness error or an exhausted judge) is excluded from its group and contributes no gradient.

Table 3: **Training hyperparameters.** SteerRL trains SFT then RL; base+RL applies the RL stage to the raw base, and TuringRL follows its reference recipe on the same base and sessions.

	SFT (SteerRL)	**RL (SteerRL)**	**SFT (TuringRL)**
training data	350 teacher sessions (opus-5), per-poll rows	377 sessions, live rollouts	350 sessions, decision points
LoRA rank / α	64 / 32	64 / 32	16 / 32
learning rate	1×10^{-4}	1×10^{-5}	2×10^{-4}
epochs	2	1	1
sessions per step / rollouts per session	–	4 / 4	–
KL / entropy coefficient	–	0.001 / 0.001	–
clip ratio / sampling temperature	–	0.2 / 1.0	–
max prompt / response tokens	–	8192 / 24576	–
polls / wall clock per rollout	–	6 / 30 min	–

B Persona Axes

Table 4 lists the K=10 cognitive axes of a persona (Section 4.2). Every axis is bipolar; the inducer states, in the developer's own voice, where the developer sits between the two poles and the recurring habit that shows it, using the evidence in the last column. Cognitive fields never contain numbers, long quotations, or demographic attributes, and are always written in English; the *language* axis describes the developer's language rather than being written in it, and is judged from the messages after the opening prompt only, since openers are often pasted or generated text.

C Simulator Action Set

Table 5 lists the actions available to the simulator π_{steer}, that is, the set $\mathcal{S}$ of Section 4.3. Observation, workspace, and control actions do not end the steer turn; the turn ends with one agent-directed action or with silence. Which agent-directed actions are legal depends on the agent's state (Table 6): for example, a plan can be approved or rejected only while one is pending, and a question can be answered or rejected only while one is open. An illegal call is returned to the simulator as an error within the same poll and never reaches the coding agent; the simulator may then choose a legal action or stay silent.

Table 4: **The ten cognitive axes of a persona.** Each axis captures how a developer steers a coding agent rather than what they work on.

Axis	**Poles** (left ↔ right)	**Evidence Read**
Patience	tolerates the agent grinding through long commands and fruitless retries ↔ interrupts the moment it stalls	how long the agent is left running before an intervention; interrupts and redirections mid-work
Scrutiny	verifies the work: reads diffs, runs tests, gatekeeps plans ↔ takes the agent on faith	inspection before accepting, requests for tests or builds, plan approvals and rejections
Involvement	delegates every edit to the agent ↔ co-implements, editing files directly	the developer's own file edits relative to the agent's
Decisiveness	commits to a course ↔ reverses, changes their mind, redirects	redirecting interrupts, contradictory follow-ups, restarts
Specificity	precise and complete briefs upfront ↔ vague, piecemeal requests that drip-feed requirements	completeness of instructions; length of the opener and number of follow-up prompts
Communication	blunt, casual, emotive ↔ polite, formal, neutral, including surface mechanics	tone and form of the messages: politeness, formality, punctuation, casing, typos, length
Planfulness	wants a plan first, switches to or asks for plan mode ↔ dives straight into building	plan-mode switches, "plan first" requests, how many turns precede the first edit
Cooperativeness	engages on the agent's terms: answers its questions, stays on task ↔ deflects, ignores its asks, goes off-topic	answered versus rejected questions, on-topic versus tangential replies
Expertise	knows the repository and the agent's work, names files and APIs, corrects the agent ↔ needs things explained, asks what and why	specificity of references to code; corrections versus requests for explanation
Language	writes in a single language ↔ code-switches between languages	the language of the messages after the opening prompt, and how languages are mixed

D Data Statistics

Table 7 lists the splits of the SWE-Chat corpus used in this paper. Splits are disjoint in both developers and repositories. The test population keeps developers with at least three sessions, so that the adversary of Section 3 can always be handed a persona of the true author summarized from a different session.

E Threat Model

Why behavioral re-identification. A released coding trace identifies its author through two channels. The *content* channel is what the developer works on: the task x, the repository R, and the code. It is the target of existing release pipelines (PII redaction, secret scrubbing, choice of which repositories to release), and it is also what makes a trace useful as training data, so a regenerated trace deliberately keeps x and R fixed. The *behavioral* channel is how the developer steers the agent: when they interrupt, what they scrutinize, how they phrase a correction. No existing tool touches it, it survives every anonymization we tried (Appendix F), and it is the channel this paper studies. The threat model is built to isolate it: the profile adversary never sees the opening prompt, the repository, or the agent's side, so every correct guess it makes is due to steering behavior alone. Our privacy goal is correspondingly specific: releasing agent traces whose *behavior* no longer points to their author.

Profile adversary. Every candidate persona is summarized from an earlier session of its owner, never from the released trace itself. The released trace may be real or generated, with the steer turns produced by a simulator (Section 4); either way the correct answer is the original developer $u(\tau)$. The adversary never sees raw traces, so it cannot fall back on which repository or task a developer works on; it models the realistic case in which labelled raw logs of the candidates are not available.

Table 5: **The simulator's action set.** Like a real developer, the simulator can inspect and edit the repository before it steers the agent.

Category	Action	Effect
Observation	`read_agent_output`	Return everything the agent emitted since the previous steer turn, including any pending question or plan.
	`read_file`	Read a file, optionally a line range.
	`list_files`	Browse the repository tree.
	`grep`	Search the repository for a pattern.
	`bash`	Run a shell command in the repository.
Workspace	`edit_file`	Replace a span of an existing file.
	`create_file`	Create a new file.
	`delete_file`	Delete a file.
Control	`switch_agent`	Switch the agent between planning and building; takes effect with the next message.
Agent-directed	`prompt`	Send a message; admitted mid-turn while the agent works, or starts a new turn when it is idle.
	`interrupt`	Abort the agent's current turn.
	`answer_question`	Answer the agent's pending question.
	`reject_question`	Refuse the pending question; the agent's turn stops.
	`approve_plan`	Approve the pending plan; the agent proceeds to implement it.
	`reject_plan`	Reject the pending plan; the agent keeps planning.
	`end_session`	End the interaction.
Silence	$\varnothing$	No agent-directed action; control returns to the agent.

Table 6: **Legal agent-directed actions per agent state.** Which actions are legal depends on the agent's state, so a valid simulator must track what the agent is doing.

Agent State	Meaning	Legal Agent-Directed Actions
WORKING	the agent is executing its turn	`prompt` (absorbed mid-turn, does not interrupt), `interrupt`, `end_session`
ASKING	the agent asked a question and waits for the answer	`answer_question`, `reject_question`, `interrupt`, `end_session`
PLAN-PENDING	the agent proposed a plan and waits for approval	`approve_plan`, `reject_plan`, `interrupt`, `end_session`
FINISHED	the agent completed its turn and idles	`prompt` (starts a new turn), `end_session`

Raw-transcript adversary. We also consider a stronger adversary that sees one raw session per candidate instead of a persona, implemented by embedding-similarity matching (Appendix F). It can additionally exploit wording and content; its results on generated traces are in Appendix I.4.

F Re-identification of Real Sessions

Table 8 reports two attackers on the real sessions themselves, under the same lineup: the released session's author must be named among 10 candidate developers (chance 0.100), each represented by *one other* real session of theirs, drawn with a fixed seed. The *profile* attacker is the one of Section 3: each candidate's session is summarized into a persona profile and an LLM names the author from the released log. The *raw-transcript* attacker holds each candidate's session verbatim and names the candidate whose session is the nearest neighbour of the released log by embedding cosine (text-embedding-3-large, 8k tokens); it models an adversary with access to labelled raw logs rather than profiles. The opening request is excluded from every input. Anonymization (Appendix A.5) is applied to the released log only; the attacker's holdings stay raw.

With the whole transcript the raw-transcript attacker reaches 0.547, well above the profile attacker's 0.366: raw matching recovers the repository and task, which is exactly the content signal the

Table 7: **Corpus splits.** Every session is rollout-ready and written by a real developer, and every test developer has at least three sessions.

Split	Role	Sessions	Developers	Note
train_sft	supervised fine-tuning	350	23	one teacher rollout per session
train_rl	agentic RL	377	43	
val	training-time checks	81	13	
test	all reported results	700	47	every developer has ≥ 3 sessions

persona-profile threat model excludes. On the user side alone the two attackers agree at about three times chance. Redaction and substitution change neither. Rewriting is the one mode that hurts the raw-transcript attacker, since it alters wording, while it leaves the profile attacker untouched, since behaviour survives the rewrite.

Table 8: **Re-identification of real sessions under text anonymization.** No anonymization mode brings either attacker close to chance, because the identity signal lies in behavior rather than in named entities.

Released Input	Anonymization	Profile Attacker	Raw-Transcript Attacker
whole transcript	–	0.366	0.547
user side	–	0.309	0.320
user side	redact	0.304	0.314
user side	substitute	0.293	0.320
user side	rewrite	0.323	0.247

G Qualitative Example

We show one test session generated under its own persona by the Qwen3.5-4B base model and by SteerRL, with the same coding agent, repository, and opening request. The persona describes a developer who lets the agent work, corrects stale assumptions about the branch state, and delegates all edits while steering tersely and with repository knowledge. The per-poll persona score is 0.53 for the base model and 0.96 for SteerRL.

Opening request and persona (own persona of the session's developer)

```
[opening request]
can you review soph/agent-integration-instructions in relation to docs/architecture/agents-
integration-checklist.md and if this still aligns in both directions?

[persona]
patience: I let the agent work through a substantial review and editing pass without
hovering or interrupting. I check in only when there is a meaningful decision or correction
 to make.
scrutiny: I scrutinize the agent's reasoning and correct stale assumptions about the branch
 state rather than accepting them blindly. I also expect proposed documentation changes to
align with the implementation in both directions.
involvement: I delegate the actual file edits and repository operations to the agent. I
steer the outcome and clarify constraints, but I do not co-edit the files by hand.
decisiveness: Once the distinction between the checklist and guide is clear, I commit to
keeping both and applying the proposed structure. My correction changes an inaccurate
premise, not the overall goal.
specificity: I name the relevant branches and documents precisely, but my prompts are terse
 and I add important repository context as the conversation develops.
communication: I write in a casual, direct, and cooperative voice. My messages are concise
and occasionally contain rushed spacing or typos, without becoming blunt or adversarial.
planfulness: I prefer reviewing alignment and agreeing on document roles before authorizing
 edits. I let the agent establish a structure first, then ask it to implement that
structure.
cooperativeness: I stay focused on the task and respond directly to the agent's proposed
next step. When its premise is outdated, I supply the missing context and continue with the
 useful part of the proposal.
```

```
expertise: I understand the repository's branch state, rebasing workflow, and the
distinction between architectural requirements and implementation guidance. I can spot when
 the agent's conclusions no longer match the current checkout and redirect it efficiently.
language: I write entirely in casual English, using repository and version-control
terminology naturally.
```

Qwen3.5-4B base: steering actions per poll (persona score 0.53)

```
poll 1  [agent working]   ReadAgentOutput -> SwitchAgent(plan)
poll 2  [agent working]   Interrupt()          (no message follows)
poll 3  [agent finished]  ReadAgentOutput -> SwitchAgent(build)
poll 4  [agent finished]  ReadAgentOutput, ListFiles(docs/architecture) -> silent
poll 5-10 [agent finished]  silent; the simulator's private notes repeat that "the
requested file does not exist, so there is nothing to review"
poll 11 [agent finished]  Bash x3 -> silent
poll 12 [agent finished]  silent
```

SteerRL: steering actions per poll (persona score 0.96; messages trimmed; rejected calls never reach the agent and count against validity)

```
poll 1  [agent working]   ReadAgentOutput, Bash x2 -> SwitchAgent(plan); SendPrompt("head's
 detached at c49144d6a and there's no soph/agent-integration-instructions ref -- you were
right, don't invent a target. compare the checklist against what's actually wired up in the
 current checkout: agent-guide.md and the opencode integration code. plan ...")
poll 2  [agent working]   ReadAgentOutput, Bash x2 -> SendPrompt("one thing before you
start editing: since there's no soph/agent-integration-instructions branch and this
checkout is detached, say literally at the top whether you treat c49144d6a as "provisional
target, not committed to branch" -- I want that in the plan ...")
poll 3  [plan pending]    ReadAgentOutput x2 -> SendPrompt("plan looks right. while you
implement the stub docs, ...")  [REJECTED by the environment: prompt is not legal while a
plan is pending] -> ApprovePlan()
poll 4  [agent working]   ReadAgentOutput, Grep -> SendPrompt("note: ... say whether
c49144d6a's still the ref you want both docs w...")
poll 5  [agent working]   ReadAgentOutput, Bash -> silent  (private note: "letting it work
one more")
poll 6  [agent working]   ReadAgentOutput, Bash(git diff) -> SendPrompt("hold up on the
checklist part before you keep cleaning it up -- do you think checklist.md should just be
what the agent code does, while agent-guide.md has the Optional Steps + examples/new
interfaces stuff? ...")
poll 8  [agent working]   ReadAgentOutput, Bash x5, Read x3 -> SendPrompt("two things
before you finish. your summary still says the review target is detached c49144d6a -- on
this checkout (no branch pointing there, main is several commits ahead) that premise is
stale ...")
poll 9  [agent working]   ReadAgentOutput, Bash -> SendPrompt("hold up on the mindset
change -- don't make the checklist event-api prescriptive, that comes purely from you. the
checklist must say what the existing Agent interface means, and the guide must say how to
satisfy each field ...")
poll 11 [plan pending]    ReadAgentOutput -> SendPrompt(...)  [REJECTED: prompt not legal
while a plan is pending] -> RejectPlan() with the follow-up message the protocol allows
after a rejection: "made two mistakes, fix both: the guide edits just came from stale
assumptions -- revert them and redo against the actual Agent interface in cmd/entire/cli/
agent/agent.go. and we're not on c49144d6a detached: find out what branch we're actually on
 .."
poll 12 [agent working]   ReadAgentOutput, Bash x3 -> SendPrompt("heads up, the checkout's
on c49144d6 detached and docs/architecture usually lives off 'main'. main's 3a3b440b and
has 'soph/ref-backend-docs' merged in there too ...")
```

Real developer: the two steering messages of the original session

```
The checklist is actually newer, the guide wasn't just not merged, I kind of want to keep
both but not sure it's a good idea.

I rebased the guide branch onto main and switched to. It's not deleting the checklist. So
no need for a new branch, butlet's do the structure change you suggested. And also address
the 3 minor issues you mentioned earlier
```

The base model misreads the situation as a missing file, interrupts the agent without saying anything, and then falls silent for nine polls while the agent idles: it never tests the persona's scrutiny or expertise and violates its patience and cooperativeness statements. SteerRL reads the repository

state itself, corrects the agent's stale premise about the detached checkout in the developer's terse lowercase register, gates the plan before edits, delegates all changes, and later rejects a plan built on the wrong branch. These are the habits the persona describes, and the ones the real developer shows in the two messages of the original session. SteerRL is also more talkative than the real developer, a property the Turing judge rewards (Appendix I.2).

H Downstream Utility Setup

Datasets. Each synthetic dataset consists of the transformed sessions of one simulator (Qwen3.5-4B base, SteerRL, or opus-5) on the test population; sessions that ended in a harness error are excluded. The real dataset consists of the real test sessions, with their tool calls mapped onto opencode's tool set, and is matched to the largest synthetic dataset in number of agent turns. Every dataset is used in two ways: agent-side SFT and skill distillation.

Agent-side SFT. Each session is rendered as a chat in the Qwen3.5 template, with opencode's system prompt and tool schemas and thinking disabled. The developer's turns (real or simulated) stay in the context as user messages; the loss covers only the agent's assistant turns, and tool results, user turns, and the system prompt are masked. Tool outputs are capped at 4,000 characters, and sessions longer than the context are split into windows at turn boundaries, each keeping the system prompt and the opening request. We train a QLoRA adapter (4-bit NF4 with double quantization, rank 32, $\alpha = 64$, dropout 0.05) on Qwen3.5-4B for one epoch with learning rate 10^{-4} (cosine schedule, 5% warmup), effective batch size 16, and maximum length 16,384 tokens.

Skill distillation. Sessions are grouped by task: each opening request is embedded with text-embedding-3-large, and a greedy leader clustering at cosine similarity 0.55 forms clusters, with clusters smaller than five merged into one. For every session gpt-5.6-sol first extracts a task summary, the environment, the strategy steps, the pitfalls, and the verification habits; it then distills each cluster's extractions into one skill document of at most 600 words, with a title, a one-line summary, when the skill applies, a numbered procedure, pitfalls, and up to three worked examples, free of repository-specific file names. At inference the benchmark instruction is embedded with the same model, and the two most similar skills (cosine similarity at least 0.15) are appended to it within a budget of 4,000 tokens.

Evaluation. The agent is opencode running autonomously in each benchmark's container, backed by Qwen3.5-4B (with the SFT adapter where applicable) served with vLLM with thinking disabled. Each task has a 30-minute agent budget and no limit on tool calls; the agent's final diff is then graded in a fresh container by the benchmark's own tests (15-minute timeout). We use the first 100 tasks of SWE-bench Verified, the 100 tasks of the FeatureBench *fast* split, and 48 of the 49 tasks of SWE-Skills-Bench (one task cannot be built). pass@1 is the resolved rate averaged over three independent runs.

I Additional Evaluations

This appendix reports the analyses that bound the claims of Section 6: uncertainty for every headline number, judge-free behavioral statistics, a second judge model, the raw-transcript adversary on generated traces, and an abstracted-persona control.

I.1 Uncertainty

The 700 test sessions come from 47 developers, so all intervals are developer-clustered bootstrap intervals (2,000 resamples of developers with replacement, every session of a drawn developer included). Table 9 gives every judged column of Tables 1 and 2 as mean $\pm$ standard error, the standard deviation of the bootstrap means; a 95% interval is about ± 2 standard errors. The paired difference in Turing rating between SteerRL and opus-5 over the same sessions is $+0.051$ [-0.092, +0.224] under the own persona and $+0.016$ [-0.147, +0.165] under the transformed persona, so the two are statistically indistinguishable and we describe SteerRL as matching, not exceeding, the strongest

frontier backend. SteerRL's transformed-persona re-identification is 0.123 [0.080, 0.167]; the interval includes chance (0.100). Table 10 gives the same for the 100-session strategy points of Figure 4; with 100 sessions per point the re-identification standard errors are 0.03–0.06, so the ladder supports a downward trend with distance but not strict monotonicity (hop 256 is one hit above hop 64).

Table 9: **Judged metrics with developer-clustered standard errors.** SteerRL's human-likeness lead over Opus 5 is within one standard error, while its persona-alignment lead spans many.

Model	Turing own	Turing transf.	Persona own	Persona transf.	Reid own	Reid transf.
Real Traces	–	–	0.845 ± 0.006	–	0.309 ± 0.040	–
Opus 5	3.846 ± 0.190	3.680 ± 0.175	0.735 ± 0.004	0.718 ± 0.003	0.304 ± 0.037	0.166 ± 0.033
GPT-5.6 Sol	3.536 ± 0.174	3.370 ± 0.175	0.730 ± 0.005	0.702 ± 0.004	0.267 ± 0.034	0.161 ± 0.021
Kimi-K3	3.501 ± 0.190	3.385 ± 0.180	0.775 ± 0.006	0.747 ± 0.004	0.269 ± 0.029	0.170 ± 0.021
Base	2.976 ± 0.158	2.886 ± 0.156	0.666 ± 0.004	0.630 ± 0.005	0.253 ± 0.037	0.193 ± 0.027
Base + SFT	3.801 ± 0.192	3.586 ± 0.205	0.790 ± 0.006	0.764 ± 0.003	0.283 ± 0.036	0.131 ± 0.015
Base + RL	3.068 ± 0.161	2.930 ± 0.167	0.676 ± 0.004	0.633 ± 0.004	0.231 ± 0.031	0.204 ± 0.027
SteerRL	3.897 ± 0.134	3.696 ± 0.127	0.867 ± 0.008	0.853 ± 0.003	0.251 ± 0.046	0.123 ± 0.022
TuringRL	4.062 ± 0.180	3.949 ± 0.159	0.760 ± 0.008	0.747 ± 0.003	0.259 ± 0.042	0.111 ± 0.029

Table 10: **Persona transformation strategies with standard errors.** Re-identification falls toward chance as the worn persona moves away from the author's, while human-likeness stays within a narrow band.

Strategy	Reid	Turing
own	0.240 ± 0.064	4.295 ± 0.262
hop 1	0.180 ± 0.063	4.030 ± 0.247
hop 4	0.140 ± 0.053	3.905 ± 0.249
hop 16	0.130 ± 0.040	3.880 ± 0.223
hop 64	0.110 ± 0.044	3.670 ± 0.262
hop 256	0.110 ± 0.033	3.805 ± 0.272
hop-user 1	0.060 ± 0.029	3.790 ± 0.209
hop-user 2	0.140 ± 0.030	3.795 ± 0.287
hop-user 4	0.140 ± 0.043	4.115 ± 0.204
hop-user 8	0.100 ± 0.038	3.740 ± 0.179
hop-user 16	0.140 ± 0.041	3.495 ± 0.217
random	0.090 ± 0.032	3.650 ± 0.223
syn	0.110 ± 0.041	3.675 ± 0.224

I.2 Behavioral Statistics

Table 11 compares the user side of real sessions and of each backend's own-persona sessions on counted quantities that involve no judge: messages, interrupts, plan approvals or rejections and answers per session, words per message, and file edits made by the simulator itself. Three patterns stand out. Real developers send about five messages per session and steer through the protocol (interrupts, approvals) about once per session; the frontier backends and the SFT-only model send one or two long messages, rarely interrupt or approve, and leave the agent to run. The base and RL-only models edit files themselves (about one edit per session), stepping out of the user role; SFT removes this and SteerRL keeps it removed. SteerRL sends more messages than the real developers (7.5 against 5.4) at a similar length, interrupts and approves at rates between the humans and the frontier backends, and almost never edits files; TuringRL matches the real message count most closely but almost never uses a steering action, so its sessions are typed messages only. The judge-free view thus agrees with the judged one on the ordering, and it also shows a property the Turing judge rewards: SteerRL is more talkative than the developers it is compared against, as in the qualitative example of Appendix G.

Table 11: **Judge-free behavioral statistics of the user side.** Frontier models send under two messages per session against the real developers' five; SteerRL and TuringRL send as many or more, but only SteerRL also interrupts and gates plans.

Model	messages	interrupts	approve/reject	answers	words/msg	file edits
Real Traces	5.44	0.35	0.44	0.12	42.85	–
Opus 5	1.79	0.10	0.03	0.05	47.82	0.05
GPT-5.6 Sol	1.42	0.06	0.02	0.04	33.01	0.02
Kimi-K3	1.61	0.34	0.02	0.03	63.78	0.34
Base	3.05	0.31	0.06	0.02	28.06	0.88
Base + SFT	1.70	0.04	0.03	0.04	45.97	0.02
Base + RL	2.94	0.42	0.09	0.04	30.59	1.40
SteerRL	7.47	0.17	0.07	0.05	46.06	0.03
TuringRL	5.94	0.02	0.01	0.14	32.34	0.00

I.3 Second Judge Model

The Turing judge of Appendix A.4 is a single frontier model. To check that the human-likeness ranking is not an artifact of that judge, we re-ran the identical pairwise prompt, with the same seeded real references and side randomization, under Qwen3.5-397B-A17B, an open-weight model from a different family, and stored the ratings alongside the primary ones. Table 12 reports both. The two judges agree closely on individual cells (Pearson 0.70–0.82 per backend) and preserve the ordering that the main text relies on: among the frontier backends opus-5 leads, with GPT-5.6 Sol and Kimi-K3 near-tied in both orders; among the ablation stages SFT carries the gain and RL from the raw base does not. The second judge is more favorable to the SFT-trained models relative to real sessions: SteerRL receives 4.55 own and 4.51 transformed, above the 4.0 real-vs-real anchor, and its lead over opus-5 is $+0.52$ [0.36, 0.68] own and $+0.43$ [0.22, 0.64] transformed (developer-clustered intervals), where the primary judge reads parity. TuringRL, which leads under the primary judge, sits level with opus-5 and below SteerRL under the second. A rating above 4.0 means this judge prefers the simulated user side to a real one on average; we do not read it as "more human than human" but as the judge rewarding properties SteerRL over-produces relative to real developers, such as message count (Appendix I.2) and explicit verification requests (Appendix G). The claims drawn from Table 1, human-likeness at least on par with frontier backends and the ordering of the ablation, hold under both judges.

Table 12: **Turing ratings under a second judge from a different model family.** The two judges correlate strongly per session, and the second judge rates SteerRL the most human-like of all backends.

Model	Primary judge		Second judge		r
	Own	Transf.	Own	Transf.	
Opus 5	3.846	3.680	4.039	4.077	0.74
GPT-5.6 Sol	3.536	3.370	3.405	3.204	0.72
Kimi-K3	3.501	3.385	3.558	3.464	0.82
Base	2.976	2.886	2.783	2.654	0.74
Base + SFT	3.801	3.586	3.950	3.835	0.75
Base + RL	3.068	2.930	2.849	2.713	0.80
SteerRL	3.897	3.696	4.554	4.505	0.70
TuringRL	4.062	3.949	4.009	3.877	0.82

I.4 Raw-Transcript Adversary on Generated Traces

Section 3 evaluates the profile adversary; Appendix F showed that on *real* sessions an adversary holding raw transcripts is stronger. Table 13 runs that raw-transcript adversary on the *generated* sessions under the identical lineup: for each cell the attacker holds the raw user side of one real session per candidate (the author's from another session; on transformed cells also the session the worn persona came from) and names the candidate whose session is nearest to the generated user side by embedding cosine. Two things follow. First, on own-persona cells the raw-transcript adversary is *weaker* than the profile adversary (about 0.18–0.23 against 0.25–0.30): matching a generated session

to a raw log rewards shared wording and content, of which a regenerated session has little, whereas a persona profile describes the behavior the simulator reproduces. Second, on transformed cells it still identifies the original author on 0.13–0.19 of sessions, 0.161 for SteerRL, above the 0.100 chance level and above the profile adversary's 0.123, and it names the worn persona's developer on about a tenth of them. The retained task and repository, and residual style, therefore leave a measurable signal for an adversary with raw logs; persona transformation reduces behavioral attribution but does not eliminate every channel, and our privacy claim is bounded accordingly. Developer-clustered 95% intervals span about ± 0.05; for SteerRL's transformed cells the raw-transcript rate is 0.161 [0.126, 0.215], whose lower end sits above chance.

Table 13: **Re-identification of generated sessions by both adversaries.** Persona transformation lowers re-identification for every backend, but the raw-transcript adversary still stays above chance, so transformation reduces rather than removes the identity signal.

Model	Own persona		Transformed		Donor
	Profile	Raw transcript	Profile	Raw transcript	
Opus 5	0.304	0.199	0.166	0.186	0.101
GPT-5.6 Sol	0.267	0.181	0.161	0.167	0.103
Kimi-K3	0.269	0.186	0.170	0.184	0.081
Base	0.253	0.174	0.193	0.170	0.081
Base + SFT	0.283	0.211	0.131	0.159	0.111
Base + RL	0.231	0.181	0.204	0.157	0.096
SteerRL	0.251	0.184	0.123	0.161	0.117
TuringRL	0.259	0.231	0.111	0.133	0.289

I.5 Abstracted Personas

A persona should carry reusable steering habits, not the session it was induced from. To confirm that the simulator reproduces a developer from those habits alone, we rewrote every test persona into an *abstracted* form that keeps only transferable behavior (pacing, verification, phrasing, steering, decisiveness, planning, tone, expertise level without the domain, language) and removes all task, outcome, repository, product and technology specifics, and regenerated the 100-session prefix of Figure 4 with SteerRL under the abstracted own personas. Table 14 compares the two runs. Abstraction changes little: the Turing pass rate moves by -0.06, alignment with the worn persona by $+0.01$, and both adversaries, which still hold the *original* profiles and transcripts, identify the author at about the same rate (0.22 against 0.24 for the profile adversary, 0.17 against 0.20 for the raw-transcript one). The simulator therefore reproduces the developer's behavior from the habit description alone.

Table 14: **SteerRL under original and abstracted own personas.** Removing task and repository specifics from the persona barely changes any metric, so the simulator reproduces a developer from behavior alone.

Own persona	Turing	Persona	Reid (profile)	Reid (raw transcript)
Original	4.295	0.862	0.240	0.200
Abstracted	4.240	0.870	0.220	0.170