BEAGLE: Behavior-Enforced Agent for Grounded Learner Emulation

Hanchen David Wang, Clayton Cohn, Zifan Xu,
Siyuan Guo, **Gautam Biswas**, **Meiyi Ma**
Vanderbilt University
{hanchen.wang.1,clayton.a.cohn,zifan.xu,
siyuan.guo,gautam.biswas,meiyi.ma}@vanderbilt.edu

Abstract

Simulating student learning behaviors in open-ended problem-solving environments holds potential for education research, from training adaptive tutoring systems to stress-testing pedagogical interventions. However, collecting authentic data is challenging due to privacy concerns and the high cost of longitudinal studies. While Large Language Models (LLMs) offer a promising path to student simulation, they suffer from *competency bias*, optimizing for efficient correctness rather than the erratic, iterative struggle characteristic of novice learners. We present **BEAGLE**, a neuro-symbolic framework for computational STEM (C-STEM) learning environments, where students learn disciplinary content by building executable computational models. **BEAGLE** addresses this bias with an architecture inspired by Self-Regulated Learning (SRL) theory. **BEAGLE** integrates three key technical innovations: (1) a semi-Markov model that governs the timing and transitions of cognitive behaviors and metacognitive behaviors; (2) Bayesian Knowledge Tracing with explicit flaw injection to enforce realistic knowledge gaps and "unknown unknowns"; and (3) a decoupled agent design that separates high-level strategy use from code generation actions to prevent the model from silently correcting its own intentional errors. In evaluations on four Python-based physics and applied-mathematics tasks, **BEAGLE** reproduces authentic trajectories more faithfully than nine baselines. In a human Turing test using our animated trace viewer, participants could not reliably distinguish rendered BEAGLE traces from real student traces (52.8%, $d' = 0.15$, $N = 71$). Code is available at https://github.com/HCWDavid/BEAGLE-NIPS2026.

1 Introduction

Generative AI is advancing education, especially through personalized learning and real-time feedback. However, these advances share significant challenges: they require authentic student data to train, evaluate, and refine pedagogical models. Acquiring such data is difficult because strict privacy regulations (e.g., FERPA, GDPR) limit access to collecting comprehensive classroom data [55], while the prohibitive costs of data collection, such as IRB approvals, parental consent, and longitudinal designs, create substantial barriers [6]. Moreover, intervening in how real students learn raises ethical concerns, particularly in K-12 settings where suboptimal and incorrect experimental interventions could disadvantage minors academically [22]. This problem has motivated researchers to explore synthetic student generation [26, 59, 37, 30], and simulating learner behaviors computationally enables researchers to stress-test pedagogical strategies without putting real students at risk.

Current Large Language Models (LLMs) offer an appealing foundation: their capacity for nuanced generation and persona adoption makes them natural candidates for simulating rich learner behaviors. However, LLMs exhibit ***competency bias***: the inherent tendency of preference-tuned models to

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

produce correct solutions even when prompted to simulate novice students [32, 39, 1]. Competency bias persists because authentic student behavior emerges from processes rather than outcomes.

Self-Regulated Learning (SRL) theory [65] provides a principled framework for this view, positing that learning unfolds as a cyclic process involving cognition (attempting tasks), metacognition (monitoring progress), and motivation and affect (engagement and seeking help, or disengaging). Authentic student errors are artifacts of breakdowns in these regulatory processes: students frequently iterate between planning, enacting, monitoring, and reflecting. *Prior attempts to induce novice-like outputs through persona prompting or knowledge suppression fail because preference-tuned LLMs are fundamentally optimized for outcome quality, not process fidelity.* Empirical evidence supports this claim. As shown in Fig. 1, student patterns observed in [53] reveal a heavy loop on iterative debugging. In contrast, LLM-generated students show the opposite behavior: dominating continuous construction transitions while bypassing the back-and-forth debugging process almost entirely.

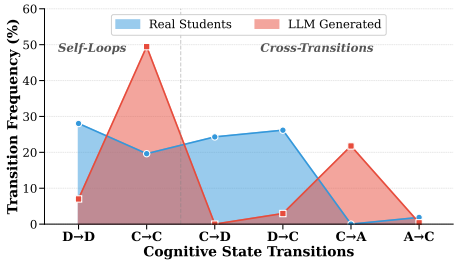


Figure 1: ***Competency bias***: real students debug persistently (D→D: 28%) while LLMs construct linearly. C (`Constructing`), D (`Debugging`), A (`Assessing`).

These limitations motivate our framing of the *multi-dimensional fidelity problem* across three dimensions: ***(1) Behavioral Infidelity:*** LLMs optimized for efficiency exhibit a *linear construction trap*, avoiding the iterative trial-and-error cycles central to SRL [47, 24]. ***(2) Epistemic Infidelity:*** Authentic novices suffer the *curse of incompetence* [16]: they lack the skill to solve problems (unknown unknowns) and the insight to recognize errors. LLMs, acting as "masked experts," frequently violate these constraints by diagnosing errors before execution or leaking suppressed knowledge [52, 54]. ***(3) Perceptual Infidelity:*** RLHF training reduces output diversity to a flattened caricature [28], failing to capture the noise of authentic student expression [11, 23].

We present **BEAGLE** (**B**ehavior-**E**nforced **A**gent for **G**rounded **L**earner **E**mulation), a neuro-symbolic framework that overcomes competency bias to generate high-fidelity synthetic student trajectories in computational STEM (C-STEM) learning environments, where students learn disciplinary content by building and testing executable computational models [51, 25, 3]. Our design principle is that student simulation is a capability-constraint problem: in our experiments, the constraint could not be delegated to the model that has the capability, so **BEAGLE** enforces it architecturally. **BEAGLE** enables scalable AI education research to train adaptive tutors, validate pedagogical theories, and stress-test interventions without the ethical constraints of real student experimentation. The contribution lies in the integration and enforcement strategy rather than in the individual components:

1. A semi-Markov model trained on real student logs that governs when and how long agents plan, act, or reflect, enabling the iterative C↔D transitions that LLM baselines systematically bypass.
2. Bayesian Knowledge Tracing that constrains what concepts the agent can utilize, with explicit flaw injection to simulate unknown unknowns and observation filtering to enforce the curse of incompetence, preventing premature diagnosis of errors.
3. A Strategist/Executor architecture that decouples planning from code generation, using task framing to prevent silent self-correction.
4. A comprehensive evaluation across four tasks, six frontier LLM backbones, and nine baselines, including a human Turing test on rendered trace stimuli ($N = 71$), in which BEAGLE reaches near-chance discriminability ($d' = 0.15$, $p_{\mathrm{TOST}} = 0.038$).

2 Problem Formulation

We formulate synthetic student generation as a sequence synthesis problem defined over two semantic domains: the universe of natural language strings $\mathcal{S}_{\text{text}}$ and the universe of executable source codes $\mathcal{S}_{\text{code}}$. The process is conditioned on a static configuration $\Phi = (\mathcal{P}, \mathcal{K}, \rho)$, where $\mathcal{P} \in \mathcal{S}_{\text{text}}$ represents the problem description and $\mathcal{K}$ denotes the set of tracked knowledge components. The student profile $\rho = (\rho_{\text{behav}}, \rho_{\text{persona}})$ consists of two dimensions: a behavioral parameter $\rho_{\text{behav}} \in \{\textsc{High}, \textsc{Low}\}$

governing transition dynamics, and a linguistic persona $\rho_{\text{persona}} \in \{\text{HIGH}, \text{LOW}\}$, which maps a proficiency level to specific stylistic constraints.

We model interaction dynamics with action space $\mathcal{A} = \mathcal{S}_{\text{code}} \times \mathcal{S}_{\text{text}}$ and a *hierarchical* latent state: *metacognitive behaviors* $\mathcal{M}$ (e.g., PLANNING) govern high-level phases, while *cognitive behaviors* $\mathcal{C}$ (e.g., `Constructing`) are fine-grained actions *within* each phase. Trajectories split into metacognitive segments indexed by n (segment n has behavior $M_n \in \mathcal{M}$ and duration D_n); segments are concatenated into cognitive steps $t \in \{1, \ldots, T\}$ with $T = \sum_n D_n$ and $n(t)$ mapping step t to its segment. External interventions $\mathbf{I} = \{\mathcal{I}_t\}_{t=1}^{T}$ ($\mathcal{I}_t \subseteq \mathcal{S}_{\text{text}}$) carry optional tutor feedback. A synthetic trajectory is $\tau = \{(M_{n(t)}, C_t, a_t, o_t)\}_{t=1}^{T}$ with $a_t \in \mathcal{A}$ and observation o_t (agent sees filtered $\tilde{o}_t$; §3.4). We write the generative mapping as $\tau \sim P_\theta(\cdot \mid \Phi, \mathbf{I})$, where θ collects the architectural mechanisms (semi-Markov dynamics, knowledge constraints, agent design).

The Multi-Dimensional Fidelity Problem. We formalize three orthogonal fidelity objectives. Let $P_{\text{real}}(C)$ denote the ground-truth marginal distribution over cognitive behaviors and $P_\theta(C)$ the simulated distribution; transition-level dynamics (e.g., debugging stickiness) are captured separately by the compound D_{debug} metric (App. E.4).

1. ***Behavioral Fidelity:*** We minimize the KL divergence between the simulated and real cognitive behavior distributions:
$$D_{\text{KL}}(P_\theta(C) \mid\mid P_{\text{real}}(C))$$
2. ***Epistemic Fidelity:*** We enforce the *curse of incompetence* via the error recurrence rate. Real novices repeatedly encounter the same error type due to limited diagnostic expertise, while LLMs fix errors immediately:
$$\mathbb{P}(\exists t' > t : e \in o_{t'} \mid e \in o_t)$$
3. ***Perceptual Fidelity:*** We maximize the expected realism score assigned by an evaluator function $\mathcal{J}(\tau)$ (e.g., human or LLM judge):
$$\mathbb{E}_{\tau \sim P_\theta}[\mathcal{J}(\tau)]$$

3 Method

3.1 System Overview

BEAGLE is a training-free framework: Neural Action does zero-shot in-context generation with frozen LLMs, emitting free-form Python rather than choosing from a fixed action vocabulary. Symbolic Control parameters (semi-Markov transitions, Gamma durations, interrupt rates) are fit once on historical student logs (transitions by maximum likelihood, durations by moment matching) [53] (App. A, A.3); BKT slip/guess/learning rates use standard priors [14] (App. B.1). No parameter is tuned on evaluation tasks. Three coupled subsystems implement this design: **Symbolic Control** enforces SRL workflow and knowledge constraints (§3.2); **Neural Action** factors generation into a Strategist/Executor pipeline whose task framing prevents silent self-correction (§3.3); and the **Environment** executes code and gates feedback by metacognitive behavior to enforce novice perceptual blindness (§3.4).

Step Lifecycle. At each cognitive step t, Symbolic Control samples the current metacognitive behavior M_n (semi-Markov) and cognitive behavior C_t, together with discretized knowledge constraints κ_t (BKT + EFI). The Strategist consumes these and the Shared Context $\Omega_t = (c_{t-1}, \tilde{o}_{t-1}, \kappa_t, \mathcal{I}_t)$ to emit a plan (g_n, m_n, d_n) (Goal, Mindset, Directive); the Executor expands the plan into action $a_t = (c_t, u_t)$ (code and monologue). The Environment runs c_t in the IDE Oracle, gates the observation by M_n to yield $\tilde{o}_t$, and updates BKT mastery only when $M_n \in \{\text{REFLECTING}, \text{MONITORING}\}$. The remaining subsections specify each block; Fig. 2 traces the same flow visually.

3.2 Symbolic Control

Semi-Markov Dynamics. To enforce realistic temporal consistency, we employ a semi-Markov Controller. We adopt this architecture because standard Markov models inherently enforce geometric duration distributions that fail to capture the irregular persistence (e.g., "getting stuck") observed in real student logs (see App. A for empirical validation).

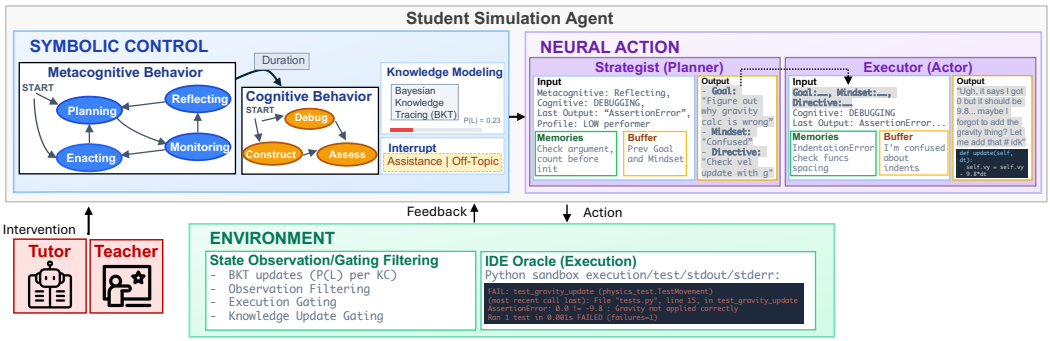


Figure 2: Overview of the **BEAGLE** agent architecture. **Symbolic Control** (left) governs high-level behavior via semi-Markov metacognitive and cognitive behavior state machines, with BKT-based knowledge tracking. **Neural Action** (right) factors generation into a two-stage pipeline: a *Strategist* that emits a goal, mindset, and directive, and an *Executor* that produces code and monologue conditioned on that directive (prompts in App. C). **Environment** (bottom) executes code via the IDE Oracle and feeds gated observations back to both subsystems.

Each metacognitive segment n is characterized by a behavior M_n, a sampled duration D_n, and a sequence of cognitive behaviors $\mathbf{C}_n = (C_n^{(1)}, \ldots, C_n^{(D_n)})$. Concatenating all segments yields the global trajectory $(C_1, \ldots, C_T)$ with $T = \sum_n D_n$. We factorize the joint distribution as:

$$P(M_n, D_n, \mathbf{C}_n) = P(M_n \mid M_{n-1}) \cdot P(D_n \mid M_n) \cdot \prod_{i=1}^{D_n} P(C_n^{(i)} \mid M_n, C_n^{(i-1)})$$

The metacognitive layer operates as a first-order Markov chain, while durations D_n follow Gamma distributions tailored to ρ_{behav}. Two stochastic interrupt behaviors, ASSISTANCE and OFF-TOPIC, extend this chain as progress-conditioned Gaussians: ASSISTANCE peaks mid-task (proactive help-seeking) and OFF-TOPIC peaks late (frustration), with parameters fit on empirical patterns (App. A.3). Architecturally, **BEAGLE**'s components are domain-agnostic, requiring only a knowledge component (KC) taxonomy and action schema to port; the semi-Markov layer can also host alternative theoretical frameworks by replacing M and C.

Knowledge Modeling (BKT). Furthermore, we employ a hybrid mechanism combining Bayesian Knowledge Tracing (BKT) with Explicit Flaw Injection (EFI; §3.2). For each knowledge component $k \in \mathcal{K}$ at step t, we model the probability of mastery as a continuous variable $P(L_k^{(t)}) \in [0, 1]$. This probability is updated via Bayes' rule after every synthetic observation generated during REFLECTING or MONITORING behaviors (Details in App. B.1).

Explicit Flaw Injection (EFI). Standard BKT attributes errors solely to performance slips, failing to model the complete absence of conceptual awareness. To capture these "unknown unknowns," EFI constrains selected KCs as *Blocked*: they deterministically yield incorrect observations, freeze BKT updates, and are treated as nonexistent in the agent's vocabulary so the Executor cannot fall back on suppressed expert knowledge. A Blocked KC cannot be recovered through self-practice alone; the agent must receive a tutor hint targeting that KC (App. B.4). Mastered and Partial KCs surface as natural-language constraints κ_t, with Partial sampled stochastically Correct/Incorrect via BKT slip/guess.

3.3 Neural Action

We decompose the agent function into a two-stage pipeline: a *Strategist* that synthesizes intent and an *Executor* that realizes it. To streamline notation, let $\Omega_t = (c_{t-1}, \tilde{o}_{t-1}, \kappa_t, \mathcal{I}_t)$ denote the *Shared Context*. Additionally, each stage maintains an independent memory buffer, $\Gamma_{\text{Strat},n}$ and $\Gamma_{\text{Exec},t}$, preventing circular dependencies between planning and execution logic.

The Strategist. The Strategist bridges the gap between symbolic constraints and natural language. It maps the current metacognitive segment to an intermediate representation (g_n, m_n, d_n): a durable *Goal*, an affective *Mindset*, and an actionable *Directive*.

$$f_{\text{Strat}} : (M_n, C_t, \Gamma_{\text{Strat},n}, \Omega_t; \rho_{\text{persona}}) \mapsto (g_n, m_n, d_n) \tag{1}$$

This architecture decouples *planning* from *writing*. The metacognitive behavior M_n bounds the strategic depth (e.g., ENACTING forces impulsive directives), while the persona ρ_{persona} conditions the formulation. By operating solely on high-level goals, the Strategist serves as an internal monologue, separate from the details of coding.

The Executor. The Executor consumes this high-level directive to generate the final action $a_t = (c_t, u_t)$ at each cognitive step. Crucially, it is constrained by the distilled directive: the symbolic state shapes vocabulary and output format (via Mandate, Profile, and $\text{Task}_{\text{Exec}}$ blocks; App. C), but the executable plan comes only from (g_n, m_n, d_n):

$$f_{\text{Exec}} : (M_n, C_t, \Gamma_{\text{Exec},t}, \Omega_t, g_n, m_n, d_n; \rho_{\text{persona}}) \mapsto a_t \tag{2}$$

This task framing is what enforces epistemic fidelity: the Executor implements a directive rather than diagnosing the problem, so even with full access to code and error context it follows flawed directives instead of overriding them with better fixes. Merging the two stages collapses this constraint and enables silent self-correction (validated in ablation, §4.4). Both stages use a block-based prompt architecture (App. C); the Strategist's (g_n, m_n, d_n) is injected into the Executor's user prompt as a hard constraint, keeping planning and writing aligned without merging them.

3.4 Environment

Observation Filtering. To simulate the selective attention of novices, we gate feedback based on metacognitive behavior. When the agent is in an impulsive ENACTING state, error traces are redacted, rendering the student blind to the root cause of failure:

$$\tilde{o}_t = \begin{cases} \text{“[Error]: [}\textit{output omitted...}\text{]”} & \text{if } M_{n(t)} = \text{ENACTING} \\ o_t & \text{otherwise} \end{cases} \tag{3}$$

This forces the agent into realistic trial-and-error loops until the symbolic controller transitions it to any other behaviors.

Execution and Knowledge-Update Gating. Code execution is just-in-time per cognitive behavior: `Constructing` writes blindly, `Debugging` runs code first to expose errors, `Assessing` runs solely for evaluation. BKT mastery updates are restricted to MONITORING and REFLECTING states, mirroring how students consolidate knowledge only when consciously evaluating outcomes.

4 Evaluation

We assess whether **BEAGLE** achieves authentic student simulation across behavioral (RQ1), epistemic (RQ2), and perceptual (RQ3) dimensions, complemented by ablation studies and generalization analyses across tasks and LLM backbones. The appendix reports the full experimental setup (App. E) and additional analyses: BKT mastery–accuracy adherence (App. Fig. 18), $\rho_{\text{behav}} \times \rho_{\text{persona}}$ decoupling (App. F.5), Turing test protocol and statistics (App. H), tutor-mechanism case study (App. G), and per-task / per-backbone numerics (App. F.1, F.2).

Problem Domain. Our evaluation suite spans **three domains** (physics, applied math, and computing) in an established research paradigm where disciplinary knowledge and programming are studied *together* [25, 57, 42, 10, 34]. Specifically, we target **four tasks**, three of which are in-distribution (class-based kinematics simulation matching the physics+code structure of the Snyder corpus and similar to our pilot and the Cohn study): kinematics via a *Particle Simulator* (2D projectile motion with gravity and drag), and dynamics via a *Bouncing Ball* and an *Inclined Plane* (collision dynamics, friction). We additionally target one cross-domain task, *Gradient Descent* (applied math, non-OOP), as a natural extension since it shares the same core coding KCs with the in-distribution tasks, while its math KCs (derivatives, numerical differentiation, iterative convergence) belong to the same computational-thinking-in-mathematics curriculum studied in prior work [62]. The four tasks share 24 unique knowledge components (11 coding, 8 physics, 5 math); each problem invokes an 11–14 KC subset (App. E.1). The 19 coding and physics KCs were elicited from the Snyder corpus (1 CS professor + 2 PhD students, grounded in SRL theory), and the 5 math KCs were added with the Gradient Descent task; each is an AST-parsable atomic unit, with an IDE Oracle attributing test outcomes per-KC for BKT credit assignment (full mapping in App. B.2).

Table 1: Main evaluation on Particle Simulator (N=50, T=30, Gemini 2.0 Flash). Gray = LLM-as-Judge (human agreement $\kappa = 0.76$; App. F.3). D_{KL}, D_{debug} vs. Real. Best **bold**, 2nd underline; '–' = n/a (rule-based). P_{Recur}: bold = inside both real-student intervals (§4.2).

	Task Perf.			Behavioral			Epistemic			Perceptual		Overall
Method	Solve	Steps	Gap	$D_{\text{KL}}\downarrow$	$D_{\text{debug}}\downarrow$	Nonlin↑	P_{Recur}	Lag↑	Debug↑	Code	Lang	Realism↑
Vanilla	100.00±0.00%	6±3	+0%	3.82±0.12	0.09±0.03	0.00±0.02	7.8±20.3%	1.00±0.00	1.20±0.53	2.88±0.32	2.00±0.49	1.18±0.52
Vanilla+$\mathcal{M}$	92.00±3.84%	11±7	+8%	3.89±0.58	0.08±0.03	0.02±0.06	10.1±22.2%	1.02±0.14	1.34±0.71	**2.92±0.27**	2.04±0.63	1.32±0.71
CoT	100.00±0.00%	7±4	+0%	0.53±0.18	0.06±0.02	0.02±0.04	26.4±35.7%	2.62±2.57	1.08±0.39	2.26±0.74	2.16±0.67	1.08±0.39
CoT+$\mathcal{M}$	90.00±4.24%	16±8	+12%	1.54±0.14	0.52±0.07	0.00±0.01	33.3±47.1%	1.50±0.50	1.00±0.00	1.72±0.63	1.64±0.59	1.00±0.00
Few-Shot	60.00±6.93%	22±8	+16%	1.76±0.17	0.10±0.01	0.04±0.05	56.7±37.4%	1.00±0.00	1.74±0.91	2.86±0.40	2.36±0.74	1.74±0.91
FewShot+$\mathcal{M}$	22.00±5.86%	28±5	-20%	0.63±0.04	0.47±0.08	0.00±0.00	75.0±40.3%	1.00±0.00	1.42±0.72	2.62±0.52	2.30±0.67	1.64±0.84
SimStudent	0.00±0.00%	30±0	–	0.95±0.04	0.45±0.02	0.00±0.00	92.0±18.3%	–	1.00±0.00	2.20±0.45	1.00±0.00	1.00±0.00
LLMSS	96.00±2.77%	8±6	-8%	0.83±0.22	0.20±0.03	0.01±0.02	28.3±41.2%	1.65±1.00	1.06±0.24	2.46±0.61	2.30±0.73	1.02±0.14
CoderAgent	56.00±7.02%	19±11	+72%	3.81±0.49	0.23±0.03	0.05±0.06	49.6±46.7%	**5.47±5.38**	1.14±0.49	2.02±0.58	1.80±0.82	1.14±0.45
BEAGLE	10.00±4.24%	29±4	+4%	**0.31±0.05**	**0.02±0.01**	**0.34±0.13**	**86.2±21.3%**	2.62±1.98	**2.46±0.85**	2.90±0.41	**2.62±0.66**	**2.44±0.85**

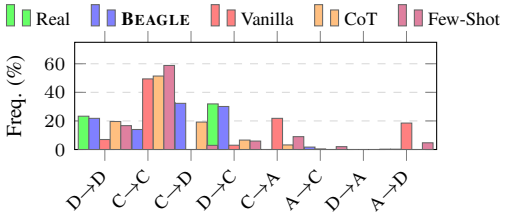


Figure 3: **Cognitive behavior transitions. BEAGLE** (blue) closely matches real data.

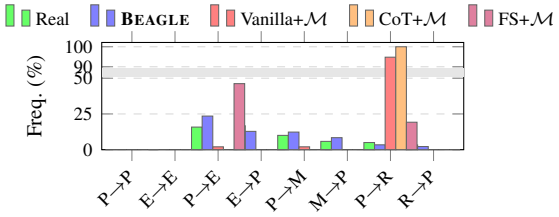


Figure 4: **Metacognitive behavior transitions. BEAGLE** closely matches Real. Baselines show extreme P→R bias (axis break at 50%).

Datasets. We use four datasets, each with a distinct role. *(i) Snyder corpus* [53] ($N = 9$ dyad sessions, 18 students, 227 segments): used once to calibrate the semi-Markov model, with HIGH/LOW labels assigned at a 0.8 final-score threshold. Snyder is never used for evaluation. *(ii) Cohn block-based study* [12] ($N = 10$ dyad sessions, 20 students, 411 segments): introductory physics in a block-based environment, serving as the primary behavioral ground truth. *(iii) Python pilot study* ($N = 13$ sessions from 5 students, 306 segments): introductory physics in a Python environment, collected under IRB-approved protocols (#170887) with informed consent (App. A.4); used as additional behavioral ground truth and as stimuli for the human Turing test. *(iv) Bielefeld dataset* [44] ($N = 15$, keystroke-level): grounds the Gradient Descent task and used 2-second pause protocol for Turing test (App. H). The Cohn and pilot datasets are aggregated into a combined evaluation set (denoted Real) for behavioral metrics (D_{KL}, D_{debug}).

Baselines and Configuration. We compare against: (1) pure prompting (Vanilla, CoT, Few-Shot); (2) rule-based SimStudent [40]; (3) exemplar-conditioned LLM-SS [43] using DCU dataset traces [2]; and (4) CoderAgent [64] with ACT-R-inspired memory (Table 3). The $+\mathcal{M}$ variants inject **BEAGLE**'s metacognitive prompt into baselines without the architectural control, isolating prompt effects from architectural ones. We evaluate across six frontier backbones (Gemini 2.0/2.5/3 Flash, GPT-4o-mini, GPT-4.1-mini, Claude Haiku 4.5); unless noted otherwise, runs use Gemini 2.0 Flash with $N = 50$ simulations and $T = 30$ steps, balancing quality and cost in our multi-call-per-step setting.

Evaluation Metrics. We evaluate task performance (solve rate, steps-to-solve, performance gap) and the three fidelity objectives: *behavioral fidelity* via D_{KL}, D_{debug}, and nonlinearity; *epistemic fidelity* via P_{recur} (operationalizing the error recurrence objective), reaction lag, and Debug realism; and *perceptual fidelity* via Code and Lang style scores, plus a human Turing test ($N = 71$ raters; App. H) on pilot stimuli, validating fidelity beyond surface realism. All LLM-as-judge metrics use a 1–3 Likert scale (unfaithful/neutral/faithful) calibrated against human raters following established practice [13, 8, 7], achieving substantial agreement [41] on the top-level real-versus-simulated judgment (Cohen's $\kappa = 0.76$) and on the graded Realism score ($\kappa_w = 0.73$; full protocol and per-dimension agreement in App. F.3). D_{KL} and P_{recur} validate mechanisms that **BEAGLE** enforces and are anchored in real-student data; mastery alignment (Fig. 7), perceptual scores, and the Turing test are downstream outcomes that no component is directly fit to produce (App. E.4). Uncertainty for D_{KL} and D_{debug} is reported as bootstrap standard error [15] (App. E.4); solve rate reports the binomial standard error $\sqrt{p(1-p)/N}$ since each run yields a single Bernoulli outcome; remaining metrics report sample standard deviation across runs. (App. F.3).

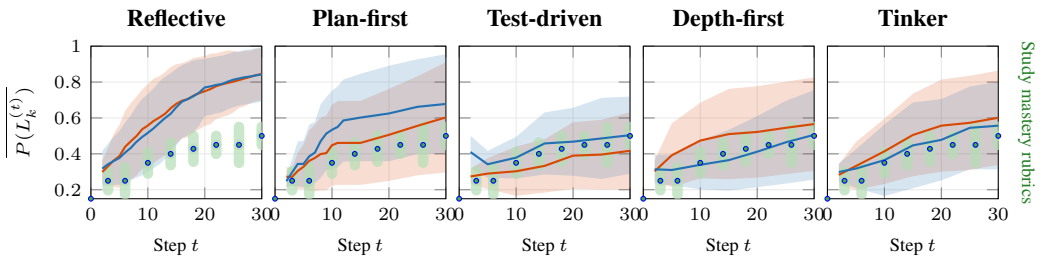


Figure 7: **BKT mastery growth conditioned on dominant strategy** (Particle Sim., pooled across six backbones). Orange = low performer, blue = high performer; shaded halos show $\pm 1\sigma$ across runs. Green capsules: rubric mastery from the Cohn block-based study (Inter-Quartile Range with median dot).

4.1 RQ1: Behavioral Fidelity



Figure 5: **Behavioral DNA of cognitive behaviors.** Longer contiguous blocks = closer to authentic behavior; bolded segments are debugging loops.

The *linear construction trap* manifests clearly in our baselines. Vanilla solves problems in ≈6 steps at 100% accuracy (Table 1), while **BEAGLE** requires ≈29 steps, mirroring real student temporal density. Fig. 3 reveals why: vanilla exhibits dominant constructing self-loops (49.4%) with zero C→D transitions; **BEAGLE** shows balanced bidirectional flow (C→D 32.9%, D→C 31.9%), capturing iterative attempt-fail-debug cycles. At the metacognitive level (Fig. 4), CoT+$\mathcal{M}$ shows extreme P→R bias (∼99%), bypassing enacting entirely; **BEAGLE** maintains diverse P→E (23.5%) and E→P (12.8%) flows. **BEAGLE** achieves $D_{\text{KL}} = 0.31$ (vs. baselines ≥ 0.53) and $D_{\text{debug}} = 0.02$ (vs. 0.06–0.52). Fig. 5 visualizes this as behavioral DNA: vanilla terminates quickly in `Constructing` segments, while **BEAGLE** shows extended interleaving with characteristic *debugging loops* (Details in App. Fig. 16). The +$\mathcal{M}$ variants stress-test context engineering by injecting the same metacognitive vocabulary **BEAGLE** uses, yet none beat $D_{\text{KL}} = 0.63$ or Realism $= 1.64$, and removing the semi-Markov controller while preserving prompts pushes D_{KL} to 6.81 (§4.4): *sequence-level student simulation cannot be recovered through context engineering alone; architectural enforcement closes the gap.*

Performance Differentiation. Beyond aggregate fidelity, **BEAGLE** separates HIGH and LOW profiles by strategy: HIGH performers allocate 72% of steps to PLANNING and MONITORING, while LOW performers spend 50.8% trapped in ENACTING (Fig. 6), reproducing the "wheel-spinning" pattern documented in SRL literature [4]. This profile-driven differentiation translates to solve-rate gaps that scale with backbone capability (e.g., +40% on Gemini 2.5 Flash; App. F.2), whereas vanilla LLMs show no profile sensitivity (+0%, Table 1).

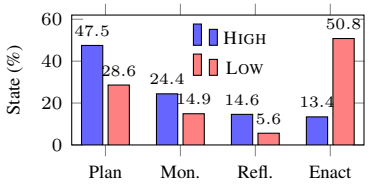


Figure 6: **Metacognitive distribution.** HIGH succeed via PLANNING; LOW trap in ENACTING.

4.2 RQ2: Epistemic Fidelity

Authentic simulation requires enforcing the *curse of incompetence* [16]: novices lack both the skill to solve a problem and the metacognition to recognize their errors. Real novices struggle repeatedly with the same errors due to limited diagnostic expertise, whereas LLMs exhibit expert-like error recognition, a fundamental "gap between LLM's underlying reasoning process and human cognitive processes" [35]. We measure this through the error recurrence rate, P_{recur} (App. E.4): the proportion of runs in which the same error type recurs. We anchor this metric with two real-student references (App. E.4): applying the same P_{recur} implementation to our pilot logs gives 76.4% (95% CI [62.5, 90.3]), and replaying the Bielefeld snapshots through that study's own unit tests gives 92.2% [83.3, 100.0]. Vanilla LLMs achieve only 7.8% recurrence (Table 1), quickly resolving diverse errors; **BEAGLE** (86.2%) is the only simulator inside both human intervals. A stricter variant, which counts an

Table 2: **Ablation Study** ($N = 50$, $T = 30$,Gemini 2.0 Flash). D_{KL} = behavioral divergence vs. combined test data. Combined Agent merges Strategist/Executor into a single LLM. Strict = strict recurrence variant (an error type must return after being resolved; §4.2); the top rows give the real-student references.

Variant	Steps	$D_{\mathrm{KL}}\downarrow$	P_{recur}	Strict	Debug↑	Lang↑	Realism↑
Real students (pilot)	–	–	76.4%	29.2%	–	–	–
Real students (Bielefeld)	–	–	92.2%	81.7%	–	–	–
Full **BEAGLE**	29±4	0.31±0.05	86.2±21.3%	49.3%	2.46±0.85	2.62±0.66	2.44±0.85
No BKT	29±4	0.26±0.05	90.2±16.1%	68.7%	2.33±0.91	2.49±0.73	2.33±0.84
No semi-Markov	27±4	6.81±0.29	83.2±17.5%	53.8%	2.38±0.87	2.62±0.60	2.38±0.85
No Interrupts	29±4	0.27±0.05	91.0±17.4%	32.3%	2.68±0.65	2.60±0.66	2.54±0.78
No Γ_{exec}	30±1	0.27±0.05	83.2±18.2%	61.0%	2.36±0.82	2.70±0.54	2.38±0.77
No Γ_{strat}	29±3	0.29±0.05	86.3±19.9%	57.7%	2.42±0.87	2.70±0.54	2.42±0.87
Combined Agent	26±7	0.32±0.05	65.3±20.3%	50.8%	1.76±0.95	2.62±0.56	1.88±0.93

error type as recurrent only if it returns after the student had moved past it, separates recurrence from perseveration: rule-based SimStudent's 92.0% falls to 0% because its errors repeat until termination rather than resolve and return, every LLM baseline stays at or below 13.3%, and real students show 29.2% (pilot) to 81.7% (Bielefeld), with **BEAGLE** (49.3%) between the two. Two mechanisms enable **BEAGLE**'s epistemic fidelity: observation filtering withholds diagnostic feedback during impulsive ENACTING states, preventing premature error recognition; and the Strategist/Executor split forces the Executor to implement flawed plans without silent correction (ablation confirms merging them reduces recurrence by 21%). Beyond aggregate recurrence, *strategy-conditional* BKT mastery growth (Fig. 7) shows that *Reflective* trajectories saturate near 0.85 prior-independently, whereas *Plan-first*, *Test-driven*, and *Depth-first* show clear high > low gaps, confirming strategy choice shapes mastery trajectories, and BKT meaningfully tracks this in simulated students.

4.3 RQ3: Perceptual Fidelity

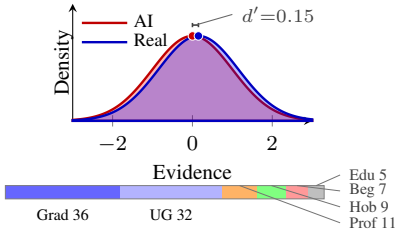


Figure 8: Turing test SDT and participant composition (%); N=71.

In the LLM-as-Judge evaluation, **BEAGLE** traces achieve the highest realism score (2.44/3.00, Table 1), far exceeding baselines (1.00–1.74). We further conducted a human Turing test with $N = 71$ participants (852 classifications; 12 traces each across 2 scenarios, 3 real and 3 simulated per scenario) spanning graduate students (Grad), undergraduates (UG), professionals (Prof), hobbyists (Hob), beginners (Beg), and educators (Edu); composition shown in Fig. 8. Participants viewed each trace as a 25+ snapshot animated progression before classifying it as real or AI-generated. Overall accuracy was 52.8%, not statistically distinguishable from chance ($p = 0.053$). Signal Detection Theory (SDT) analysis (Fig. 8) reveals a bias toward labeling any trace as real ($c = -0.24$). This bias manifests asymmetrically: participants correctly identified 62.2% of real traces but only 43.4% of AI traces, worse than random guessing. To formally test whether **BEAGLE** traces are perceptually equivalent to real data (rather than merely failing to reject a difference), we applied a Two One-Sided Tests (TOST) equivalence procedure. Results confirm that the discriminability index ($d' = 0.15$, where $d' = 0$ indicates perfect indistinguishability) falls significantly within the equivalence bound of ± 0.3 ($p_{\mathrm{TOST}} = 0.038$). Notably, evaluator expertise offered no advantage: graduate students and professionals performed no better than beginners, suggesting the discriminability ceiling is set by the trace itself rather than by reader experience. Together, these analyses show that rendered BEAGLE trace stimuli are difficult for humans to distinguish from rendered real-student stimuli under our evaluation interface; this should be interpreted as evidence of perceptual realism of the overall presentation pipeline, not a fully isolated estimate of BEAGLE's underlying cognitive fidelity (see App. H for details).

4.4 Ablations and Generalization

Ablation Study. We further evaluate individual modules (Table 2). Three key findings emerge: (1) *The semi-Markov controller is the primary driver of behavioral fidelity.* Its removal causes catastrophic divergence from real student distributions (D_{KL} rises to 6.81, compared to $\leq$0.32 for all other ablations), confirming that standard LLMs inherently default to linear construction.

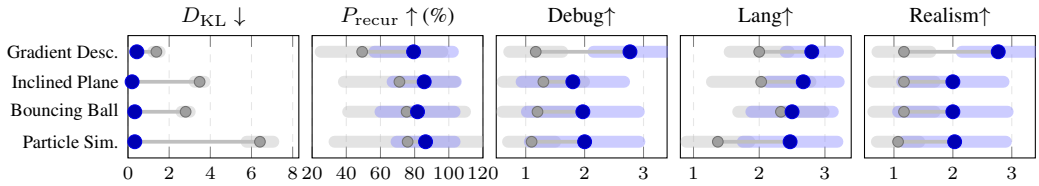


Figure 9: **Cross-task generalization (BEAGLE vs. best non-BEAGLE baseline, per task).** • Best baseline (CoT+$\mathcal{M}$ or CoderAgent, whichever is stronger per cell); • **BEAGLE**. Connecting lines show the gap; translucent halos show ± 1 std. All simulations are generated using Gemini 2.5 Flash. Across the four tasks, **BEAGLE** shrinks behavioral divergence by $\geq 3\times$, raises error recurrence to $\geq 79\%$, and improves all three perceptual scores (Debug, Lang, Realism). Full numerics in App. F.1.

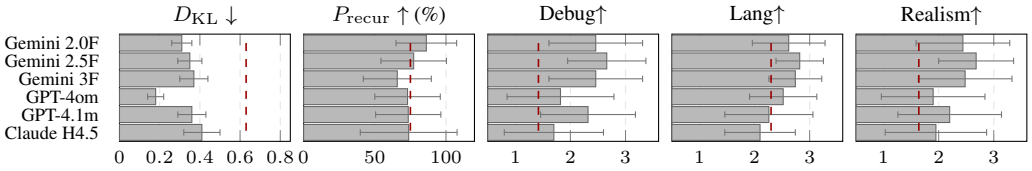


Figure 10: **Cross-model generalization on Particle Simulator** ($N = 50$, $T = 30$). Bars show **BEAGLE**'s fidelity scores for each backbone across the same five metrics as Fig. 9. *Red dashed line*: strongest non-**BEAGLE** LLM baseline (FewShot+$\mathcal{M}$ on Gemini 2.0 Flash). Every backbone clears the baseline on D_{KL}, Debug, and Realism, and four of six on Lang; on P_{recur}, two exceed it, three sit within about 1.5 points, and Gemini 3 Flash falls below (the baseline's own standard deviation is ± 40 points). Complete results in App. F.2.

(2) *Architectural separation is essential for epistemic fidelity.* Merging Strategist/Executor into a Combined Agent produces the largest realism drop ($2.44 \rightarrow 1.88$) and reduces error recurrence by 21% ($86.2\% \rightarrow 65.3\%$). Without this task framing, the LLM reverts to superficial, skilled error resolution, silently correcting mistakes rather than exhibiting diagnostic incompetence. (3) *BKT and interrupts shape how errors resolve.* Removing BKT degrades Debug realism ($2.46 \rightarrow 2.33$) and overall Realism ($2.44 \rightarrow 2.33$). On raw recurrence, No BKT (90.2%) and No Interrupts (91.0%) look alike, but the strict variant moves them in opposite directions: without mastery updates, previously resolved errors return more often ($49.3\% \rightarrow 68.7\%$), and without interrupts, errors persist in place rather than resolve and return ($49.3\% \rightarrow 32.3\%$). The full model lies between the two, within the range spanned by the real-student references (App. G).

Generalization. We test two transfer settings. *Cross-task* (Fig. 9, App. F.1): **BEAGLE** runs on the three remaining problems: two in-distribution physics tasks (Bouncing Ball, Inclined Plane) and one cross-domain applied-math task (Gradient Descent, non-OOP). Even on Gradient Descent, which lies outside the Snyder corpus, **BEAGLE** still shrinks D_{KL} by $\geq 3\times$, raises error recurrence to $\geq 79\%$, and improves all three perceptual scores. Because the semi-Markov is applied as-is, this suggests the transferred signal is general novice dynamics rather than task-specific physics structure; we read it as cross-domain transfer within executable C-STEM, not broad out-of-distribution generalization. *Cross-backbone* (Fig. 10, App. F.2): every one of six frontier backbones clears the strongest baseline on D_{KL}, Debug, and Realism, and four of six on Lang; on P_{recur}, two exceed it, three sit within about 1.5 points, and one falls below.

EFI Ablation (Qualitative). We also show the design feature of blocking the knowledge using EFI. As shown in Fig. 11, **BEAGLE** forces agents to navigate conceptual gaps when enabling EFI. When the `math` library is blocked via EFI, HIGH performers attempt to implement manual trigonometric approximations, while LOW performers guess based on expected outputs. This shows that BKT+EFI effectively forced the agent to operate in genuine ignorance, improvising alternative solutions.

5 Related Work

Prior educational data generation has focused on static artifacts [56, 9] rather than longitudinal trajectories. Computational student models span symbolic (e.g., SimStudent [40], grounded in ACT-R), neural (knowledge tracing models such as EKT [36], which predict correctness but not granular actions), and hybrid RL-LLM [49] approaches; recent LLM-based simulators include Generative

Without EFI (math library available)

```
theta_rad = math.radians(self.angle)
a = g * (math.sin(theta_rad) - self.mu *
    math.cos(theta_rad))
```

With EFI (math library blocked → manual approx)

```
def sin_approx(a): return a / 90.0
                # improvised
def cos_approx(a): return 1.0 - a / 90.0
a = g * (sin_approx(self.angle) - self.mu
    * cos_approx(self.angle))
```

Figure 11: **EFI forces authentic knowledge gaps.** Without EFI, the agent trivially uses `math.radians()`. With EFI, the agent improvises (incorrect) manual approximations. Full comparison in App. Fig. 17.

Agents [46], Generative Students [37], LLM-SS [43], EduAgent [60], and CoderAgent [64], targeting competency bias [48] via memory modules [19] or iterative reflection [61] (Table 3).

Method	Temp.	Know.	Coupl.	Theory
SimStudent [40]	✗	✓	✗	ACT-R
Gen. Agents [46]	✗	✗	✗	–
Gen. Students [37]	✗	✓	✗	KLI
LLM-SS [43]	✗	✓	✗	–
EduAgent [60]	✗	✗	✗	CogSci
CoderAgent [64]	✓	✓	✗	ACT-R
BEAGLE (ours)	✓	✓	✓	SRL

Table 3: Simulated-student frameworks. *Temp.* = temporal dynamics, *Know.* = knowledge modeling, *Coupl.* = behavior and knowledge jointly constrained.

LLM persona and student simulators rely on the LLM's internal reasoning to respect knowledge boundaries, leaving them vulnerable to leakage that interpretable knowledge tracing [31, 5] does not resolve; this is the competency-bias failure that our $+\mathcal{M}$ baselines quantify. **BEAGLE** enforces knowledge boundaries architecturally (BKT with EFI) and is the only framework in this comparison that jointly constrains behavioral dynamics and knowledge. **BEAGLE** is theory-inspired rather than theory-driven: Winne and Hadwin's SRL framework [58] determines the state inventory, the gating of knowledge updates, and the help-seeking interrupts, while the fitted state machine carries the empirical load, with transitions quantified via recent work [33] and operationalized through data-driven semi-Markov models [17, 20] that extend previous work on Markov modeling [53] and Hidden Markov modeling [27].

6 Limitations

BEAGLE's demonstrated scope is C-STEM: all four evaluated tasks are executable Python tasks in physics and applied mathematics, and **BEAGLE** presupposes an oracle that can execute and verify student work. Transfer to writing or other subjective domains is untested and would require new behavioral states, KC and flaw taxonomies, feedback mechanisms, and calibration data (App. E.3). Our real-student corpora are small (58 students, 47 sessions). The Turing test used a convenience sample without attention checks or a positive-control condition, so near-chance accuracy cannot be fully separated from limited instrument sensitivity (App. H.3). The LLM judge was calibrated on only 30 **BEAGLE** traces, which cannot rule out differential bias across methods (App. F.3).

7 Conclusion

We presented **BEAGLE**, a neuro-symbolic framework that overcomes *competency bias* via architectural constraints (semi-Markov control, BKT with EFI, and a decoupled Strategist/Executor) rather than prompting. To our knowledge, it is the first system to jointly enforce behavioral, epistemic, and perceptual fidelity in C-STEM learning environments: D_{KL}=0.31 (best baseline 0.53), error recurrence 86.2%, and rendered traces perceptually equivalent to real-student traces under our evaluation interface (N=71, d'=0.15, p_{TOST}=0.038). Future work follows from the remaining gaps: residual competency bias on the strongest backbones (Fig. 10) motivates capability-adaptive symbolic-control gain; the knowledge–fluency trade-off (Table 2) calls for richer mastery representations; and **BEAGLE**'s single-agent scope opens the door to multi-agent classrooms and adaptive tutoring policies trained on synthetic trajectories.

Acknowledgments and Disclosure of Funding

This work was supported by the Institute of Education Sciences (IES), U.S. Department of Education, under Award Number R305C240010. The opinions expressed are those of the authors and do not represent the views of the Institute or the U.S. Department of Education.

References

[1] Gati V Aher, Rosa I Arriaga, and Adam Tauman Kalai. Using large language models to simulate multiple humans and replicate human subject studies. In *ICML*, pages 337–371. PMLR, 2023.

[2] David Azcona and Alan Smeaton. +5 Million Python & Bash Programming Submissions for 5 Courses & Grades for Computer-Based Exams over 3 academic years., 7 2020. URL https://figshare.com/articles/dataset/_5_Million_Python_Bash_Programming_Submissions_for_5_Courses_Grades_for_Computer-Based_Exams_over_3_academic_years_/12610958.

[3] Satabdi Basu, Gautam Biswas, and John S. Kinnebrew. Learner modeling for adaptive scaffolding in a computational thinking-based science learning environment. *User Modeling and User-Adapted Interaction*, 27(1):5–53, 2017.

[4] Joseph E Beck and Yue Gong. Wheel-spinning: Students who fail to master a skill. In *AIED*, pages 431–440. Springer, 2013.

[5] Antonin Berthon and Mihaela van der Schaar. Language bottleneck models: A framework for interpretable knowledge tracing and beyond. *arXiv preprint arXiv:2506.16982*, 2025.

[6] Jessica Blom-Hoffman, Stephen S Leff, Debra L Franko, Elana Weinstein, Kelly Beakley, and Thomas J Power. Consent procedures and participation rates in school-based intervention and prevention research: using a multi-component, partnership-based approach to recruit participants. *School mental health*, 1(1):3–15, 2009.

[7] Tyler Burleigh, Jenny Han, and Kristen DiCerbo. Beyond the hint: Using self-critique to constrain llm feedback in conversation-based assessment. In *Proceedings of the Artificial Intelligence in Measurement and Education Conference (AIME-Con): Coordinated Session Papers*, pages 79–85, 2025.

[8] Arijit Chakma, Peng He, Honglu Liu, Zeyuan Wang, Tingting Li, Tiffany D Do, and Feng Liu. Drawsim-pd: Simulating student science drawings to support ngss-aligned teacher diagnostic reasoning. *arXiv preprint arXiv:2602.01578*, 2026.

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

[10] Li Cheng, Xiaoman Wang, and Albert D. Ritzhaupt. The effects of computational thinking integration in STEM on students' learning performance in K-12 education: A meta-analysis. *Journal of Educational Computing Research*, 61(2):416–443, 2023.

[11] Myra Cheng, Tiziano Piccardi, and Diyi Yang. Compost: Characterizing and evaluating caricature in llm simulations. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 10853–10875, 2023.

[12] Clayton Cohn, Siyuan Guo, Surya Rayala, Hanchen David Wang, Naveeduddin Mohammed, Umesh Timalsina, Shruti Jain, Angela Eeds, Menton Deweese, Pamela J Osborn Popp, et al. Evidence-decision-feedback: Theory-driven adaptive scaffolding for llm agents. *arXiv preprint arXiv:2602.01415*, 2026.

[13] Clayton Cohn, Surya Rayala, Namrata Srivastava, Joyce Horn Fonteles, Shruti Jain, Xinying Luo, Divya Mereddy, Naveeduddin Mohammed, and Gautam Biswas. A theory of adaptive scaffolding for llm-based pedagogical agents. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 1757–1765, 2026.

[14] Albert T Corbett and John R Anderson. Knowledge tracing: Modeling the acquisition of procedural knowledge. *User modeling and user-adapted interaction*, 4(4):253–278, 1994.

[15] Bradley Efron. Bootstrap methods: another look at the jackknife. In *Breakthroughs in statistics: Methodology and distribution*, pages 569–593. Springer, 1992.

[16] Joyce Ehrlinger, Kerri Johnson, Matthew Banner, David Dunning, and Justin Kruger. Why the unskilled are unaware: Further explorations of (absent) self-insight among the incompetent. *Organizational behavior and human decision processes*, 105(1):98–121, 2008.

[17] Louis Faucon, Lukasz Kidzinski, and Pierre Dillenbourg. Semi-markov model for simulating mooc students. In *EDM*, pages 358–363, 2016.

[18] Alvan R. Feinstein and Domenic V. Cicchetti. High agreement but low kappa: I. the problems of two paradoxes. *Journal of Clinical Epidemiology*, 43(6):543–549, 1990.

[19] Weibo Gao, Qi Liu, Linan Yue, Fangzhou Yao, Rui Lv, Zheng Zhang, Hao Wang, and Zhenya Huang. Agent4edu: Generating learner response data by generative agents for intelligent education systems. In *AAAI*, volume 39, pages 23923–23932, 2025.

[20] Chase Geigle and ChengXiang Zhai. Modeling student behavior with two-layer hidden markov models. *JEDM*, 9(1):1–24, 2017.

[21] Kilem Li Gwet. Computing inter-rater reliability and its variance in the presence of high agreement. *British Journal of Mathematical and Statistical Psychology*, 61(1):29–48, 2008.

[22] Wayne Holmes, Kaska Porayska-Pomsta, Ken Holstein, Emma Sutherland, Toby Baker, Simon Buckingham Shum, Olga C Santos, Mercedes T Rodrigo, Mutlu Cukurova, Ig Ibert Bittencourt, et al. Ethics of ai in education: Towards a community-wide framework. *International Journal of Artificial Intelligence in Education*, 32(3):504–526, 2022.

[23] Tiancheng Hu and Nigel Collier. Quantifying the persona effect in llm simulations. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, pages 10289–10307, 2024.

[24] Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. Large language models cannot self-correct reasoning yet. *arXiv preprint arXiv:2310.01798*, 2023.

[25] Nicole M Hutchins, Gautam Biswas, Miklós Maróti, Ákos Lédeczi, Shuchi Grover, Rachel Wolf, Kristen Pilner Blair, Doris Chin, Luke Conlin, Satabdi Basu, et al. C2stem: A system for synergistic learning of physics and computational thinking. *Journal of Science Education and Technology*, 29(1):83–100, 2020.

[26] Tanja Käser and Giora Alexandron. Simulated learners in educational technology: A systematic literature review and a turing-like test. *International Journal of Artificial Intelligence in Education*, 34(2):545–585, 2024.

[27] John S Kinnebrew, Gautam Biswas, Brian Sulcer, and Roger S Taylor. Investigating self-regulated learning in teachable agent environments. In *International handbook of metacognition and learning technologies*, pages 451–470. Springer, 2013.

[28] Robert Kirk, Ishita Mediratta, Christoforos Nalmpantis, Jelena Luketina, Eric Hambro, Edward Grefenstette, and Roberta Raileanu. Understanding the effects of rlhf on llm generalisation and diversity. *arXiv preprint arXiv:2310.06452*, 2023.

[29] Daniël Lakens. Equivalence tests: A practical primer for t tests, correlations, and meta-analyses. *Social Psychological and Personality Science*, 8(4):355–362, 2017.

[30] Juho Leinonen, Paul Denny, Olli Kiljunen, Stephen MacNeil, Sami Sarsa, and Arto Hellas. Llm-itation is the sincerest form of data: Generating synthetic buggy code submissions for computing education. In *Proceedings of the 27th Australasian Computing Education Conference*, pages 56–63, 2025.

[31] Fan Li, Tiancheng Zhang, Yifang Yin, Minghe Yu, Mengxiang Wang, and Ge Yu. Priority guided explanation for knowledge tracing with dual ranking and similarity consistency. In *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence*, pages 430–438, 2025.

[32] Ming Li, Han Chen, Yunze Xiao, Jian Chen, Hong Jiao, and Tianyi Zhou. Can llms estimate student struggles? human-ai difficulty alignment with proficiency simulation for item difficulty prediction. *arXiv preprint arXiv:2512.18880*, 2025.

[33] Tongguang Li, Debarshi Nath, Yixin Cheng, Yizhou Fan, Xinyu Li, Mladen Raković, Hassan Khosravi, Zachari Swiecki, Yi-Shan Tsai, and Dragan Gašević. Turning real-time analytics into adaptive scaffolds for self-regulated learning using generative artificial intelligence. In *Proceedings of the 15th International Learning Analytics and Knowledge Conference*, pages 667–679, 2025.

[34] Zuokun Li and Pey Tee Oon. The transfer effect of computational thinking (CT)-STEM: A systematic literature review and meta-analysis. *International Journal of STEM Education*, 11: 44, 2024.

[35] Naiming Liu, Shashank Sonkar, and Richard Baraniuk. Do llms make mistakes like students? exploring natural alignments between language models and human error patterns. In *AIED*, pages 364–377. Springer, 2025.

[36] Qi Liu, Zhenya Huang, Yu Yin, Enhong Chen, Hui Xiong, Yu Su, and Guoping Hu. Ekt: Exercise-aware knowledge tracing for student performance prediction. In *IEEE Transactions on Knowledge and Data Engineering*, volume 33, pages 100–115, 2021.

[37] Xinyi Lu and Xu Wang. Generative students: Using llm-simulated student profiles to support question item evaluation. In *Proceedings of the Eleventh ACM Conference on Learning@ Scale*, pages 16–27, 2024.

[38] Neil A. Macmillan and C. Douglas Creelman. *Detection Theory: A User's Guide*. Lawrence Erlbaum Associates, 2 edition, 2005.

[39] Amogh Mannekote, Adam Davies, Jina Kang, and Kristy Elizabeth Boyer. Can llms reliably simulate human learner actions? a simulation authoring framework for open-ended learning environments. In *AAAI*, volume 39, pages 29044–29052, 2025.

[40] Noboru Matsuda, William W Cohen, Jonathan Sewall, Gustavo Lacerda, and Kenneth R Koedinger. Predicting students' performance with simstudent: Learning cognitive skills from observation. *Frontiers in Artificial Intelligence and Applications*, 158:467, 2007.

[41] Mary L McHugh. Interrater reliability: the kappa statistic. *Biochemia medica*, 22(3):276–282, 2012.

[42] NGSS Lead States. *Next Generation Science Standards: For States, By States*. National Academies Press, 2013.

[43] Manh Hung Nguyen, Sebastian Tschiatschek, and Adish Singla. Large language models for in-context student modeling: Synthesizing student's behavior in visual programming. *arXiv preprint arXiv:2310.10690*, 2023.

[44] Benjamin Paaßen. Python programming dataset, 2019. URL https://doi.org/10.4119/unibi/2941052. Dataset.

[45] Benjamin Paaßen, Jessica McBroom, Bryn Jeffries, Irena Koprinska, and Kalina Yacef. Mapping python programs to vectors using recursive neural encodings. *Journal of Educational Data Mining*, 13(3):1–35, 2021. doi: 10.5281/zenodo.5634224.

[46] Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology (UIST)*, pages 1–22, 2023.

[47] Romain Puech, Jakub Macina, Julia Chatain, Mrinmaya Sachan, and Manu Kapur. Towards the pedagogical steering of large language models for tutoring: A case study with modeling productive failure. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 26291–26311, 2025.

[48] Yuxiao Qu, Tianjun Zhang, Naman Garg, and Aviral Kumar. Recursive introspection: Teaching language model agents how to self-improve. *NIPS*, 37:55249–55285, 2024.

[49] Bahar Radmehr, Tanja Kaser, and Adish Singla. Pharmasimtext: A text-based educational playground filled with rl-llm agents that work together even in disagreement. *JEDM*, 17(1): 1–40, 2025.

[50] Donald J. Schuirmann. A comparison of the two one-sided tests procedure and the power approach for assessing the equivalence of average bioavailability. *Journal of Pharmacokinetics and Biopharmaceutics*, 15(6):657–680, 1987.

[51] Pratim Sengupta, John S. Kinnebrew, Satabdi Basu, Gautam Biswas, and Douglas Clark. Integrating computational thinking with K-12 science education using agent-based computation: A theoretical framework. *Education and Information Technologies*, 18(2):351–380, 2013.

[52] Yunfan Shao, Linyang Li, Junqi Dai, and Xipeng Qiu. Character-llm: A trainable agent for role-playing. In *EMNLP*, 2023.

[53] Caitlin Snyder, Nicole M Hutchins, Clayton Cohn, Joyce Horn Fonteles, and Gautam Biswas. Analyzing students collaborative problem-solving behaviors in synergistic stem+c learning. In *Proceedings of the 14th Learning Analytics and Knowledge Conference*, pages 540–550, 2024.

[54] Amir Taubenfeld, Yaniv Dover, Roi Reichart, and Ariel Goldstein. Systematic biases in llm simulations of debates. In *EMNLP*, 2024.

[55] Jill-Jênn Vie, Tomas Rigaux, and Sein Minn. Privacy-preserving synthetic educational data generation. In *European Conference on Technology Enhanced Learning*, pages 393–406. Springer, 2022.

[56] Lei Wang, Yan Wang, Deng Cai, Dongxiang Zhang, and Xiaojiang Liu. Translating a math word problem to an expression tree. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1064–1069, 2017.

[57] Daniel P Weller, Theodore E Bott, Marcos D Caballero, and Paul W Irving. Development and illustration of a framework for computational thinking practices in introductory physics. *Physical Review Physics Education Research*, 18(2):020106, 2022.

[58] Philip H. Winne and Allyson F. Hadwin. Studying as self-regulated learning. *Metacognition in Educational Theory and Practice*, 93:277–304, 1998.

[59] Songlin Xu and Xinyu Zhang. Leveraging generative artificial intelligence to simulate student learning behavior. *arXiv preprint arXiv:2310.19206*, 2023.

[60] Songlin Xu, Xinyu Zhang, and Lianhui Qin. Eduagent: Generative student agents in learning. *arXiv preprint arXiv:2404.07963*, 2024.

[61] Songlin Xu, Hao-Ning Wen, Hongyi Pan, Dallas Dominguez, Dongyin Hu, and Xinyu Zhang. Classroom simulacra: Building contextual student generative agents in online education for learning behavioral simulation. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*, pages 1–26, 2025.

[62] Huiyan Ye, Biyao Liang, Oi-Lam Ng, and Ching Sing Chai. Integration of computational thinking in k-12 mathematics education: A systematic review on ct-based mathematics instruction and student learning. *International Journal of STEM Education*, 10(1):3, 2023.

[63] Zhihao Yuan, Yunze Xiao, Ming Li, Weihao Xuan, Richard Tong, Mona Diab, and Tom Mitchell. Towards valid student simulation with large language models. *preprint*, 2026. URL https://arxiv.org/abs/2601.05473.

[64] Yi Zhan, Qi Liu, Weibo Gao, Zheng Zhang, Tianfu Wang, Shuanghong Shen, Junyu Lu, and Zhenya Huang. Coderagent: Simulating student behavior for personalized programming learning with large language models. In James Kwok, editor, *IJCAI-25*, pages 293–301. IJCAI, 8 2025. doi: 10.24963/ijcai.2025/34. URL https://doi.org/10.24963/ijcai.2025/34. Main Track.

[65] Barry J Zimmerman. Self-regulated learning and academic achievement: An overview. *Educational psychologist*, 25(1):3–17, 1990.

Table 4: Complete notation reference.

Symbol	Domain	Description
Semantic Domains		
$\mathcal{S}_{\text{text}}$	Universe	Set of all possible natural language strings
$\mathcal{S}_{\text{code}}$	Universe	Set of all possible executable source codes
$\mathcal{A}$	$\mathcal{S}_{\text{code}} \times \mathcal{S}_{\text{text}}$	Action space (code, utterance)
Static Configuration Φ		
Φ	$(\mathcal{P}, \mathcal{K}, \rho)$	Full static configuration tuple
$\mathcal{P}$	$\in \mathcal{S}_{\text{text}}$	Problem description
$\mathcal{K}$	Finite set	Set of tracked knowledge components
k	$\in \mathcal{K}$	Individual knowledge component
Student Profile ρ		
ρ	$(\rho_{\text{behav}}, \rho_{\text{persona}})$	Student profile tuple
ρ_{behav}	$\in \{\text{HIGH}, \text{LOW}\}$	Behavioral parameter (governs dynamics)
ρ_{persona}	$\in \{\text{HIGH}, \text{LOW}\}$	Linguistic persona (governs style)
Behavior Spaces		
$\mathcal{M}$	Finite set	Metacognitive: PLANNING, ENACTING, MONITORING, REFLECTING
$\mathcal{C}$	Finite set	Cognitive: `Constructing`, `Debugging`, `Assessing`
Indexing & Timing		
t	$\in \{1, \ldots, T\}$	Cognitive behavior step index
T	$\in \mathbb{N}^+$	Total trajectory length (steps)
n	$\in \mathbb{N}^+$	Metacognitive segment index
$n(t)$	$\mathbb{N}^+ \to \mathbb{N}^+$	Maps step t to its metacognitive segment
i	$\in \{1, \ldots, D_n\}$	Index within metacognitive segment
Latent State Variables		
S_t	$\in \mathcal{M} \times \mathcal{C}$	Latent behavioral state $(M_{n(t)}, C_t)$
M_n	$\in \mathcal{M}$	Metacognitive behavior for segment n
$M_{n(t)}$	$\in \mathcal{M}$	Metacognitive behavior governing step t
C_t	$\in \mathcal{C}$	Cognitive behavior at step t
$C_n^{(i)}$	$\in \mathcal{C}$	i-th cognitive behavior in segment n
$\mathbf{C}_n$	Sequence	Cognitive sequence $(C_n^{(1)}, \ldots, C_n^{(D_n)})$
D_n	$\in \mathbb{N}^+$	Duration of metacognitive segment n
x_t	$\in [0, 1]$	Task progress at step t
Knowledge State (BKT)		
$P(L_k^{(t)})$	$\in [0, 1]$	Mastery probability for KC k at step t
$\mathbf{L}_t$	$\in [0, 1]^{\lvert\mathcal{K}\rvert}$	Vector of all mastery probabilities at t
κ_t	$\in \mathcal{S}_{\text{text}}$	Natural language constraints at t
$P(S)$	$\in [0, 1]$	BKT slip probability
$P(G)$	$\in [0, 1]$	BKT guess probability
$P(T)$	$\in [0, 1]$	BKT learning/transition probability
Actions & Observations		
a_t	$= (c_t, u_t) \in \mathcal{A}$	Action at step t
c_t	$\in \mathcal{S}_{\text{code}}$	Code snapshot at step t
u_t	$\in \mathcal{S}_{\text{text}}$	Think-aloud utterance at step t
o_t	–	Raw environment observation at step t
$\tilde{o}_t$	–	Filtered observation (after $\mathcal{F}$)
Intervention		
$\mathbf{I}$	$\{\mathcal{I}_t\}_{t=1}^T$	Full intervention sequence
$\mathcal{I}_t$	$\subseteq \mathcal{S}_{\text{text}}$	Optional tutor feedback at step t
Neural Action Module		
Ω_t	Tuple	Shared context $(c_{t-1}, \tilde{o}_{t-1}, \kappa_t, \mathcal{I}_t)$
$\Gamma_{\text{Strat},n}$	Buffer	Strategist memory at segment n
$\Gamma_{\text{Exec},t}$	Buffer	Executor memory at step t
g_n	$\in \mathcal{S}_{\text{text}}$	Goal for segment n (Strategist output)
m_n	$\in \mathcal{S}_{\text{text}}$	Mindset for segment n (Strategist output)
d_n	$\in \mathcal{S}_{\text{text}}$	Directive for segment n (Strategist output)
f_{Strat}	Function	Strategist mapping to (g_n, m_n, d_n)
f_{Exec}	Function	Executor mapping to a_t
Semi-Markov Dynamics		
$P(M_n \lvert M_{n-1})$	$\in [0, 1]$	Metacognitive transition probability
$P(D_n \lvert M_n)$	Distribution	Duration distribution (Gamma)

Continued on next page

(Table 4 continued)

Symbol	Domain	Description
$P(C_n^{(i)} \mid M_n, C_n^{(i-1)})$	$\in [0,1]$	Cognitive transition within segment
$\mathcal{F}$	Function	Observation filter: $(o_t, M_{n(t)}) \mapsto \tilde{o}_t$
Stochastic Interrupts		
$P_i(x_t)$	$\in [0,1]$	Interrupt probability at progress x
r_{peak}	$\in [0,1]$	Peak interrupt probability
μ_i	$\in [0,1]$	Gaussian mean for interrupt i
σ_i	$\in \mathbb{R}^+$	Gaussian std. dev. for interrupt i
Fidelity Objectives & Trajectory Output		
$P_{\text{real}}(C' \mid C)$	Distribution	Ground-truth cognitive transitions
$P_\theta(C' \mid C)$	Distribution	Simulated cognitive transitions
$D_{\text{KL}}(\cdot \Vert \cdot)$	$\mathbb{R}^+$	KL divergence
$\mathcal{J}(\tau)$	$\in \mathbb{R}$	Evaluator/realism scoring function
τ	Sequence	Trajectory $\{(M_{n(t)}, C_t, a_t, o_t)\}_{t=1}^T$
e	–	Error instance

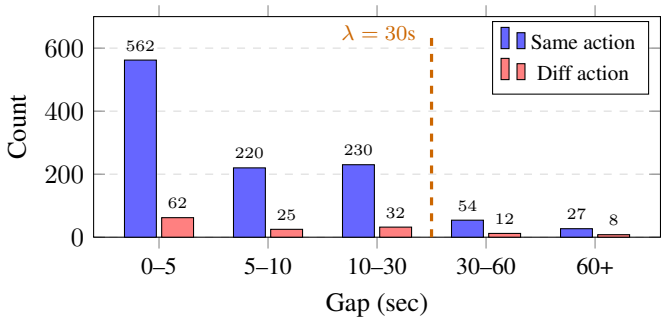


Figure 12: Distribution of temporal gaps between consecutive actions. The 30-second threshold (dashed) captures 93% of same-action sequences as single episodes.

A Ground Truth Data Analysis

Before training the semi-Markov model, we conducted systematic analyses of the Snyder corpus [53] to address two methodological challenges: (1) inflated self-transition probabilities from fine-grained logging, and (2) duration distributions that violate standard Markov assumptions.

A.1 Temporal Aggregation

The dataset records cognitive actions approximately every 6 seconds. When a student debugs for 2 minutes, this generates ≈20 consecutive `Debugging` entries, artificially inflating $P(\texttt{Debugging} \rightarrow \texttt{Debugging})$ to >89%. To recover meaningful transition dynamics, we analyzed temporal gaps between consecutive actions ($N_{\text{same}} = 1095$ same-action pairs, $N_{\text{diff}} = 139$ different-action pairs).

Figure 12 shows the gap distribution by time bucket. The ratio of same-action to different-action gaps remains ≈7–9:1 for short intervals, confirming that logging frequency is independent of action type. The 90th percentile of same-action gaps is 23 seconds; we select $\lambda = 30$s as a conservative threshold for episode aggregation.

Table 5 presents the resulting cognitive transition probabilities after time-based aggregation, stratified by metacognitive behavior and performance level.

Table 5: Key cognitive transitions $P(C_{t+1} \mid M_t, C_t)$ after 30-second aggregation. Bold indicates notable HIGH/LOW differences.

M_t	Transition	LOW	HIGH
ENACTING	start → `Debugging`	67.6%	55.0%
	start → `Constructing`	32.4%	40.0%
MONITORING	`Debugging` → `Debugging`	**86.7%**	**40.0%**
	`Debugging` → `Constructing`	13.3%	60.0%
PLANNING	start → `Constructing`	34.4%	**58.3%**
	start → `Debugging`	**62.5%**	36.1%
REFLECTING	`Debugging` → `Debugging`	57.1%	36.4%

A.2 Duration Distribution Analysis

A standard Markov chain implies ***geometric duration distributions***, which impose a fixed relationship between mean and variance. For a state with stay probability p (probability of remaining in the state at the next step), the coefficient of variation (CV) is constrained to $\text{CV} = \sqrt{p} \approx \sqrt{1 - 1/\mu}$ where μ is the mean duration. This constraint prevents independent fitting of mean and variance.

Figure 14 compares actual CV values against the geometric constraint. The data reveals substantial deviations: LOW performers in ENACTING show CV=1.35 (42% above geometric prediction), indicating high variance in segment lengths, as students either exit quickly or become deeply stuck.

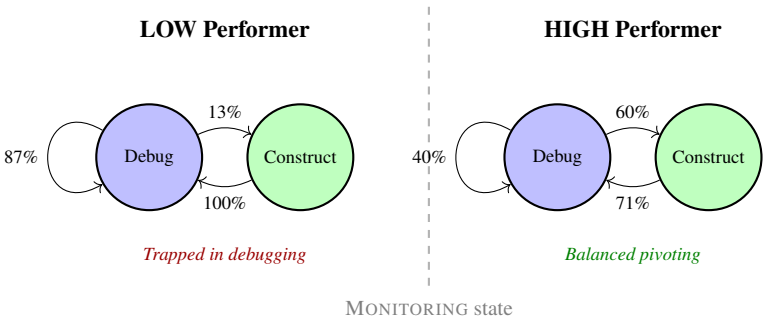


Figure 13: Cognitive transition dynamics within MONITORING. LOW performers exhibit "debugging loops" (87% self-loop), while HIGH performers pivot effectively (60% escape rate).

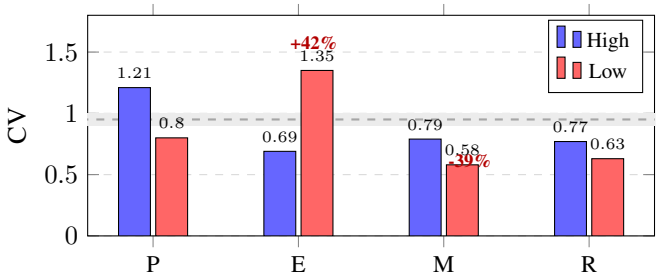


Figure 14: Actual CV of segment durations compared to geometric prediction (gray band, ≈0.95). Bars above the band indicate overdispersion; below indicates underdispersion. LOW ENACTING is notably overdispersed (+42%), while LOW MONITORING is underdispersed (-39%). P (PLANNING), E (ENACTING), M (MONITORING), R (REFLECTING)

Conversely, LOW MONITORING shows CV=0.58 (39% below prediction), suggesting more regular, predictable durations. These patterns cannot be captured by a standard Markov chain.

We therefore adopt a ***two-level semi-Markov*** architecture: (1) a Markov chain for state transitions $P(M_{t+1} \mid M_t)$, and (2) independent Gamma distributions for durations $P(D_t \mid M_t)$ as shown in Eq. 1. The Gamma distribution provides flexibility via its shape parameter α: CV $= 1/\sqrt{\alpha}$, allowing independent fitting of each state's variance.

Table 6: Moment-matched Gamma duration parameters by profile and state. Shape α and scale are chosen so the fitted mean and CV exactly match the empirical mean and CV.

Profile	M_t	Shape (α)	Scale	Mean	Fitted CV
LOW	PLANNING	1.56	4.92	7.69	0.80
	ENACTING	0.55	17.81	9.84	1.35
	MONITORING	3.02	2.69	8.12	0.58
	REFLECTING	2.53	1.74	4.40	0.63
HIGH	PLANNING	0.68	14.48	9.92	1.21
	ENACTING	2.08	3.58	7.45	0.69
	MONITORING	1.61	7.82	12.56	0.79
	REFLECTING	1.67	5.20	8.69	0.77

Table 6 reports the moment-matched parameters. LOW ENACTING has low shape ($\alpha = 0.55$), capturing the heavy variance of students who either exit quickly or become deeply stuck; LOW MONITORING has high shape ($\alpha = 3.02$), reflecting regular durations. HIGH PLANNING has the lowest shape ($\alpha = 0.68$), modeling highly variable exploration time. These profile-specific duration dynamics enable authentic temporal patterns without artificial constraints.

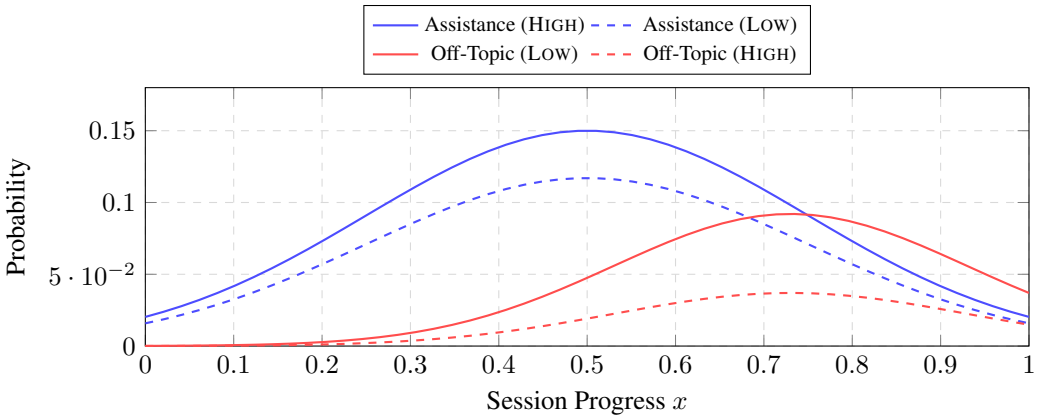


Figure 15: Interrupt probability distributions. ASSISTANCE peaks mid-session ($\mu = 0.5$) while OFF-TOPIC peaks late ($\mu = 0.73$). High performers (solid) request assistance more; low performers (dashed) disengage more.

A.3 Interrupt Modeling

Beyond the core metacognitive behaviors (Planning, Monitoring, Reflecting, Enacting), student behavior includes two **interrupt states** that temporally punctuate the learning flow:

1. **Assistance** – Student requests help from external resources or tutor
2. **Off-Topic** – Student disengages from the task temporarily

These interrupts are *not* part of the semi-Markov transition matrix. Instead, they are modeled as stochastic events conditioned on session progress, with probabilities derived from empirical data analysis [53].

Empirical Patterns. Table 7 summarizes the key characteristics of each interrupt state.

Table 7: Interrupt state characteristics from data analysis.

State	LOW	HIGH	Peak Position	Self-Loop	Primary Trigger
Assistance	11.7%	15.0%	$\mu = 0.5$ (mid)	33%	Post-ENACTING failure
Off-Topic	9.2%	3.7%	$\mu = 0.73$ (late)	40%	Fatigue/disengagement

Two counterintuitive findings emerge:

- **High performers seek more help**: 15% vs. 11.7% assistance rate. This suggests proactive metacognitive regulation, recognizing confusion early and seeking clarification before becoming deeply stuck.
- **Low performers disengage more**: 9.2% vs. 3.7% off-topic rate. This reflects the "giving up" pattern where struggling students mentally exit the task, particularly after extended ENACTING periods.

Probability Model. We model interrupt probabilities using session-progress-conditioned Gaussian distributions:

$$P(\text{interrupt} \mid x_t, \rho) = r_{\text{peak}}(\rho) \cdot \exp\left(-\frac{(x_t - \mu)^2}{2\sigma^2}\right) \tag{4}$$

where $x_t \in [0, 1]$ is session progress at t step, μ is the peak position, σ controls spread, and $r_{\text{peak}}(\rho)$ is the profile-specific peak rate.

Figure 15 visualizes these distributions. The parameters are:

Architectural Integration. Interrupts are checked at each simulation step *before* the semi-Markov state transition:

Table 8: Interrupt probability parameters.

State	μ	σ	r_{peak}
Assistance (HIGH)	0.50	0.25	0.150
Assistance (LOW)	0.50	0.25	0.117
Off-Topic (LOW)	0.73	0.20	0.092
Off-Topic (HIGH)	0.73	0.20	0.037

1. Check termination conditions (success, max steps)
2. Check `just_received_help` flag $\rightarrow$ skip interrupts
3. Sample $P(\text{Off-Topic}) \rightarrow$ enter Off-Topic if triggered
4. Sample $P(\text{Assistance}) \rightarrow$ enter Assistance if triggered
5. Normal semi-Markov transition

OFF-TOPIC is checked *first* because it represents complete disengagement as a student who is truly distracted will not think to ask for help. Assistance represents partial engagement where the student is stuck but still cognitively active.

Assistance Flow (Two-Turn Protocol). The Assistance state spans two simulation turns to capture realistic help-seeking behavior:

Turn t (Asking):

- Student formulates question based on current confusion
- Tutor generates hint (informed by BKT knowledge state)
- Sets `just_received_help = True`
- No code execution this turn

Turn $t + 1$ (Applying):

- Interrupt checks are skipped (flag is set)
- Tutor response is injected into Strategist/Executor prompts
- Student applies the help in their code
- Flag and response are cleared

This two-turn protocol prevents the unrealistic pattern of immediately utilizing help in the same cognitive step as requesting it. The tutor response influences BKT updates: successful application of a hint reinforces the associated knowledge component.

Off-Topic Behavior. When Off-Topic is triggered:

- An idle step is recorded with a distracted monologue
- No code changes or execution occur
- Step counter increments (wasting a turn)
- Control returns to normal semi-Markov flow

Extended Off-Topic periods (self-loop rate 40%) model the "rabbit hole" phenomenon where distracted students have difficulty re-engaging. This contributes to the performance gap: low performers lose more productive steps to disengagement.

Impact on Evaluation Metrics. Interrupt states directly affect several fidelity metrics:

- **Step count**: Off-Topic increases steps without progress
- **Performance gap**: More Off-Topic for LOW widens the gap
- **Assistance rate**: Observable in behavioral traces
- **Realism**: LLM-as-Judge explicitly rates help-seeking patterns

A.4 Test Data Analysis

To provide an independent benchmark for behavioral evaluation, we aggregate student interaction logs from two cohorts solving physics simulation problems. The first cohort ($N = 10$ sessions, 411 aggregated episodes) completed block-based programming tasks involving truck motion, package delivery, and collision detection. The second cohort ($N = 13$ sessions, 306 aggregated episodes) completed Python-based coding tasks spanning bouncing ball, inclined plane, particle simulator, boat crossing, and spring-mass systems. Both datasets target middle-school physics concepts, including kinematics, forces, and energy conservation.

We apply a contextual inheritance mapping strategy to recover cognitive behaviors from low-level action logs. Code-editing actions inherit their cognitive context from prior execution events: edits following a test execution within 30 seconds are classified as `Debugging`, while edits after longer gaps are classified as `Constructing`. This approach captures the insight that running code typically triggers a debugging episode, consistent with patterns observed in prior work [53]. Consecutive same-state actions within 30 seconds are aggregated into single episodes to avoid artificial inflation of transition probabilities.

The combined test dataset yields a cognitive distribution of 54.4% `Constructing` and 45.6% `Debugging` with 54.1% debug-to-debug stickiness. This balanced distribution provides a realistic benchmark for evaluating whether simulated students exhibit human-like cognitive behavior dynamics, rather than the competence-biased behavior typical of LLM baselines (Table 1).

B Method - Cont.

B.1 Bayesian Knowledge Tracing Details

BKT models latent mastery L_k for each knowledge component $k \in \mathcal{K}$ as a hidden Markov model. The observation model relates latent mastery to observed performance through slip probability $P(S)$ (probability of incorrect response despite mastery) and guess probability $P(G)$ (probability of correct response without mastery):

$$P(\text{correct}_k) = P(L_k)(1 - P(S)) + (1 - P(L_k))P(G) \tag{5}$$

At each REFLECTING or MONITORING step, the IDE Oracle (App. E.1) returns a per-KC verdict $o_k \in \{\text{correct}, \text{incorrect}\}$ derived from AST-level grading of the agent's submitted code. Mastery beliefs for each relevant KC are then updated via Bayes' rule:

$$P(L_k \mid \text{correct}) = \frac{P(L_k)(1 - P(S))}{P(L_k)(1 - P(S)) + (1 - P(L_k))P(G)} \tag{6}$$

$$P(L_k \mid \text{incorrect}) = \frac{P(L_k)P(S)}{P(L_k)P(S) + (1 - P(L_k))(1 - P(G))} \tag{7}$$

Learning occurs after each update with transition probability $P(T)$, representing the probability of acquiring mastery after practice:

$$P(L_k^{(t+1)}) = P(L_k^{(t)} \mid o_k) + (1 - P(L_k^{(t)} \mid o_k)) \cdot P(T) \tag{8}$$

We use standard BKT parameters from prior work [14]: initial mastery $P(L_k^{(0)}) = 0.10$, learning rate $P(T) = 0.25$, slip $P(S) = 0.05$, and guess $P(G) = 0.20$. Mastery is discretized into three levels for prompt injection: UNKNOWN ($P(L_k) < 0.3$), PARTIAL ($0.3 \leq P(L_k) < 0.7$), and MASTERED ($P(L_k) \geq 0.7$).

Interaction with EFI. When EFI is active on a KC (App. B.2), BKT updates for that KC are *frozen*: the agent has no perceptual access to the concept, so no informative observation can be generated and $P(L_k)$ remains at its prior. This freeze is what makes EFI-blocked KCs absorbing under self-practice alone; without it, deterministic-incorrect observations would drive $P(L_k)$ toward an interior fixed point (≈ 0.27) just below the UNKNOWN/PARTIAL boundary. Recovery is instead mediated by the ZPD tutor (App. B.4) rather than by stochastic accumulation, mirroring the pedagogical view that wholly unfamiliar concepts require scaffolded instruction.

B.2 Explicit Flaw Injection

While BKT models the probability of mastery, it does not prevent the LLM from accessing its latent knowledge of blocked concepts. Explicit Flaw Injection (EFI) enforces "unknown unknowns" by injecting hard constraints into prompts when a knowledge component is sampled as unmastered.

Mechanism. Each knowledge component $k \in \mathcal{K}$ is associated with a natural language description identifying the concept to block. When BKT determines that a KC should be unavailable (through low mastery probability or explicit sampling), the following constraint is injected into both Strategist and Executor prompts:

```
CRITICAL CONSTRAINT - You have NEVER heard of and CANNOT use:
[concept]. This concept does not exist in your knowledge. You
must solve the problem WITHOUT using it. If the code requires
'[concept]', you will be stuck and confused.
```

KC-to-Constraint Mapping. Table 9 provides representative mappings from knowledge components to their EFI constraint descriptions.

Table 9: Example EFI constraint mappings.

Knowledge Component	Injected Constraint
KC_C2 (Math Library)	"the math library and its functions"
KC_P5 (Radian Conversion)	"degree-to-radian conversion"
KC_P9 (Euler Integration)	"numerical integration methods"
KC_C14 (Conditionals)	"if/else conditional statements"

This mechanism forces the agent to improvise around genuine knowledge gaps rather than accessing the LLM's latent capabilities. As shown in Figure 17, blocking the math library forces students to invent (incorrect) manual approximations, which is a behavior characteristic of authentic novices who lack awareness that standard solutions exist.

Tutor-mediated EFI release. EFI models a student's prior-knowledge gap: while EFI is active on a KC, observations are deterministic-incorrect by construction and BKT updates for that KC are frozen, so self-practice alone cannot recover the skill. This captures the *curse of incompetence* [16]: a novice who lacks awareness of a missing concept cannot diagnose or repair it through repeated trial. Recovery is therefore tutor-mediated. When the help-seeking interrupt fires (metacognitive interrupt ASSISTANCE) and the ZPD tutor (App. B.4) delivers a hint targeting the EFI-active KC, the simulator deactivates EFI on that KC and credits one positive Bayesian observation, lifting $P(L_k)$ from its frozen prior (≈ 0.10) to a single-update posterior (≈ 0.51). The student then returns to `Constructing` with the new concept available, and subsequent practice follows the standard BKT trajectory. This realises the intuition that genuinely unfamiliar concepts (e.g., the `math` library) are acquired through scaffolded interaction rather than solo iteration, and ensures EFI-blocked KCs are pedagogically rather than mechanically recoverable.

B.3 Memory Buffer Architecture

Both Strategist and Executor maintain independent memory buffers to ensure coherent, non-repetitive behavior.

Sliding Window Buffers. Each agent maintains a buffer of size 3 storing recent outputs: $\Gamma_{\text{Strat},n}$ holds recent (g_n, m_n, d_n) tuples, while $\Gamma_{\text{Exec},t}$ holds recent monologues u_t. These are injected into prompts with instructions to avoid verbatim repetition.

Episodic Memory. Beyond sliding windows, the agent maintains episodic records of error-learning episodes, each containing the error pattern encountered, the realization learned, and whether a fix was applied. This anti-amnesia mechanism prevents the agent from repeatedly making identical mistakes without recognition, a key differentiator from vanilla LLMs, which treat each turn independently. When a familiar error recurs, the associated realization is surfaced in context, enabling the "Oh right, I made this mistake before" pattern enforced by the Rules block (App. C.2.3).

B.4 Tutor Intervention Protocol

When the ASSISTANCE interrupt is triggered, **BEAGLE** invokes a tutor agent to generate contextually appropriate hints. The framework supports pluggable tutor implementations; we adopt a ZPD-calibrated tutor as our default, following established principles from intelligent tutoring systems [13].

Default Tutor: ZPD-Calibrated Scaffolding. The default tutor operationalizes Vygotsky's Zone of Proximal Development by calibrating the specificity of hints to the student's current knowledge state. Scaffold intensity is determined by the BKT mastery probability for the struggling knowledge component:

Table 10: ZPD-based scaffold level mapping.

Mastery $P(L_k)$	ZPD Zone	Scaffold Level
> 0.7	Comfort	NONE
$0.5 - 0.7$	Upper ZPD	MINIMAL
$0.3 - 0.5$	Mid ZPD	GUIDING
< 0.3	Lower ZPD	EXPLICIT

Escalation Protocol. The scaffold level increases by $+1$ for each consecutive failure on the same KC, up to a maximum of $+2$. This model of realistic tutoring provides increasingly direct support when students remain stuck despite initial hints.

Knowledge State Context. The tutor receives a summary identifying the KC with the lowest current mastery (the primary intervention target), recent error patterns from the trajectory, and the student's current metacognitive behavior (to calibrate tone). This BKT-informed approach ensures hints target actual knowledge gaps rather than providing generic assistance.

EFI Release as a Tutor Effect. When the targeted KC is currently EFI-blocked (App. B.2), the intervention deactivates EFI on that KC and credits one positive BKT observation. The student leaves the interaction with both new conceptual access (EFI off) and a measurable belief update ($P(L_k)$: $0.10 \rightarrow 0.51$). Releasing additional blocks requires further help-seeking events on the corresponding KCs; tutor turns are therefore consequential state changes rather than purely textual exchanges.

B.5 Worked Example: One Segment End to End

This walkthrough follows one ENACTING segment of an actual **BEAGLE** run (LOW profile, Particle Simulator, Gemini 2.0 Flash; steps 5 to 8 of 30) and pairs it with the corresponding pattern in a real pilot session.

The real pattern being emulated. In one pilot session on the Inclined Plane task, the student's 20 code executions produced, after one clean run, `TypeError` on eight consecutive executions, two clean runs, three more `TypeErrors`, a clean run, two `NameErrors`, two further `TypeErrors`, and a final clean run. The student repeatedly edits and re-runs without resolving the underlying cause, which is the trial-and-error loop that the ENACTING state represents.

Step 1: Symbolic Control samples the segment. After a four-step PLANNING segment, the semi-Markov controller transitions to ENACTING and samples its duration from the LOW Gamma distribution (App. A), whose low shape parameter produces occasional long stuck episodes; here the segment lasts 14 steps. Within the segment, the cognitive layer emits `Constructing`, `Constructing`, `Debugging`, `Debugging` for steps 5 to 8. BKT/EFI supply the knowledge constraint κ_t for each step.

Step 2: The Strategist emits an impulsive plan. Conditioned on ENACTING, the Strategist outputs goal "Implement the `get_kinetic_energy` method to calculate the particle's kinetic energy," mindset "Uncertain," and directive "Implement the `get_kinetic_energy` method using the formula 0.5 * mass * velocity^2. Remember velocity is a vector, so use vx and vy." The plan does not mention the `update` method, which at this point is a placeholder with the wrong signature.

Step 3: The Executor implements the plan (steps 5 and 6, `Constructing`). Code is drafted but not executed. The Executor writes `ke = 0.5 * self.mass * self.vx**2 + self.vy**2 # i think...` and leaves `def update(self): print("updating particle")` in place.

Step 4: Execution and gated observation (step 7, `Debugging`). The IDE Oracle runs the tests (13 of 24 pass). The true failure is `TypeError: Particle.update() takes 1 positional argument but 2 were given`. Because the segment is ENACTING, observation filtering withholds the traceback and the agent sees only that a `TypeError` occurred. For contrast, at step 4 of the same run, in PLANNING, the agent saw the full message (`AttributeError: 'Particle' object has no attribute 'get_velocity'`) and fixed it directly.

Step 5: Blind revision (steps 7 and 8). Without the diagnostic detail, the Executor attributes the failure to the code it just wrote. Its monologue at step 7 reads: "Ugh, it crashed again. It says Typeerror, I don't know what that means, but something's wrong with the kinetic energy thing. Let me just try adding some parentheses, maybe that will help?" It changes the formula to `0.5 * self.mass * (self.vx**2 + self.vy**2)`, which is a correct physics fix but leaves the actual cause untouched. At step 8 the same `TypeError` recurs: "It crashed again, and it's a Typeerror. I thought I fixed the parentheses last time? Maybe I messed up the formula somehow? Let me try again..."

What the example shows. The recurring error is not requested by any prompt. It arises because the controller fixes the behavior and its duration, the Strategist's plan frames the work narrowly, and the gated observation removes the information needed to diagnose the fault. This is the mechanism by which the consecutive same-type errors seen in the real session arise in **BEAGLE** architecturally rather than by instruction.

C Prompt Templates

The prompt architecture implements composable blocks that assemble according to the composition formula from Section 3. This appendix documents the block hierarchy, universal rules, mandate variations, and representative examples.

C.1 Block-Based Composition

Prompts are assembled from reusable blocks. Both stages share an identical SYSTEM prompt but differ in USER prompts:

$$\text{System} = \underbrace{\text{Static}(\rho_{\text{persona}})}_{\text{persona, rules}} \oplus \underbrace{\text{Mandate}(M_n, C_t)}_{\text{behavioral frame}} \tag{9}$$

$$\text{User}_{\text{Strat}} = \underbrace{\text{Task}_{\text{Strat}}}_{\text{output format}} \oplus \underbrace{\text{Mem}(\Gamma_{\text{Strat},n})}_{\text{memory}} \oplus \underbrace{\text{Context}(\Omega_t)}_{\text{runtime state}} \tag{10}$$

$$\begin{aligned}\text{User}_{\text{Exec}} = {} & \underbrace{\text{Profile}(M_n)}_{\text{empirical patterns}} \oplus \underbrace{\text{Strategy}(g_n, m_n, d_n)}_{\text{from Strategist}} \\ & \oplus \underbrace{\text{Task}_{\text{Exec}}(C_t)}_{\text{output format}} \oplus \underbrace{\text{Mem}(\Gamma_{\text{Exec},t})}_{\text{memory}} \oplus \underbrace{\text{Context}(\Omega_t)}_{\text{runtime state}}\end{aligned} \tag{11}$$

Combined Agent configuration. In the Combined Agent ablation (Table 2), the single LLM call is assembled by the same prompt-assembly function used in the full system: identical system prompt (static persona, executor rules, and the same Mandate block), identical profile, task, and context blocks, and identical symbolic inputs (semi-Markov-sampled states, BKT/EFI constraints, and observation filtering). The only differences are that the strategy fields are not supplied by a separate call, and the output schema is extended so that the same call produces its own goal and mindset. The ablation therefore isolates the two-stage decomposition rather than prompt content. The only other element that merging removes is the Strategist's memory buffer, whose separate ablation (No Γ_{strat}) changes Realism by 0.02, against 0.56 for the merge.

This modular architecture treats the prompt as a dynamic assembly of semantic units rather than a static template, enabling precise control over the agent's behavior. Each block serves a distinct functional role in the simulation: Static blocks establish the persona's baseline voice independent of the task; Mandate blocks enforce the behavioral constraints dictated by the symbolic state S_t; and Task blocks structure the output for the specific cognitive activity. By composing these components at runtime, the system can generate a diverse range of student trajectories while maintaining strict adherence to the underlying semi-Markov dynamics. Table 11 summarizes the specific role of each component.

The $\text{Static}(\rho_{\text{persona}})$ block expands as:

$$\text{Static}(\rho_{\text{persona}}) = \underbrace{\text{Base}}_{\text{voice}} \oplus \underbrace{\text{Performer}(\rho_{\text{persona}})}_{\text{style}} \oplus \underbrace{\text{Rules}}_{\text{constraints}} \tag{12}$$

C.2 Static Components

The $\text{Static}(\rho_{\text{persona}})$ block provides foundational persona and constraints:

$$\text{Static}(\rho_{\text{persona}}) = \underbrace{\text{Base}}_{\text{voice}} \oplus \underbrace{\text{Performer}(\rho_{\text{persona}})}_{\text{style}} \oplus \underbrace{\text{Rules}}_{\text{constraints}} \tag{13}$$

Table 11: Prompt Block Components

Block	Purpose	Ref
Base	Core persona: novice voice, simple vocabulary	§C.2.1
Performer(ρ_{persona})	Style modifier (low vs high)	§C.2.2
Rules	Universal constraints (3 rules)	§C.2.3
Mandate(M_n, C_t)	State-pair behavioral framing (12 combinations)	§C.3
Profile(M_n)	Empirical language patterns per metacognitive behavior	§C.4
Task	Output format: $\text{Task}_{\text{Strat}}$ (constant), $\text{Task}_{\text{Exec}}(C_t)$ (3 variants)	§C.5
Strategy(g_n, m_n, d_n)	Strategist output passed to Executor	—
Mem(Γ_t)	Injection of stage-specific memory buffers	§C.6
Context(Ω_t)	Runtime state injection (code, feedback, KCs)	§C.6

C.2.1 Base

You are a novice student learning to code in Python.
Voice: Use simple, direct language. Think out loud naturally.
Vocabulary:
• "I think..." not "I hypothesize..."
• "It says..." not "The error indicates..."
• "That's weird" not "This is unexpected behavior"
Authenticity: You may be confused. You may not know the right answer. You learn by doing, not by analyzing.

C.2.2 Performer(ρ_{persona})

Low Performer ($\rho_{\text{persona}} = \text{low}$)
Language: "I don't know...", "Maybe if I...", "Let's just try...", "Why isn't this working?"
Behavior: Prefer action over analysis. Often repeat the same action if it fails. May give up: "I don't understand, let me just try something."

High Performer ($\rho_{\text{persona}} = \text{high}$)
Language: "Let me check...", "I think the issue is...", "Based on what I see..."
Behavior: Pattern-matching heuristics. Evidence-based reasoning. Still makes novice mistakes but tries to understand errors.

C.2.3 Rules

No Psychic Debugging
If Last Output shows "(Code drafted but not executed)", you have NOT run the code.
You Cannot: Identify bugs or issues; Say "I forgot..." or "I need to fix..."; Notice missing variables or wrong syntax.
You Can Only: Think about WHAT you're building next; Express uncertainty about the approach.

Memory Rule (Anti-Amnesia)
Before generating output, check if this is a REPEATED mistake.
If Yes: Say "Oh right, I made this mistake before" NOT "Hmm, what's wrong here?"
Show that you REMEMBER previous errors and learn from them.

Table 12: Mandate Matrix: Behavioral Framing for Each (M_n, C_t) Pair

	`Constructing`	`Debugging`	`Assessing`
PLANNING	GOAL FORMULATION: Forward looking, epistemically blind. *"We need to..."*	DIAGNOSTIC PLANNING: Hypothesis formation, analyzing failure. *"So that's why..."*	EVALUATIVE PLANNING: Hopeful, checking completion. *"Let's check if..."*
ENACTING	TRIAL ACTION: Immediate action, learning by doing. *"Let's try this"*	REACTIVE FIXING: Surface-level fixes, anti-efficiency. *"Maybe 3.7"*	QUICK CHECK: Shallow evaluation, moving fast. *"Looks good"*
MONITORING	PROGRESS TRACKING: Observing values, still blind to bugs. *"It's not at 15 yet"*	ERROR TRACKING: Tracking progress, frustration. *"It's not stopping"*	RESULT VERIFICATION: Checking final output. *"Did it pass?"*
REFLECTING	LEARNING REFLECTION: Meta-observation, seeking understanding. *"That's weird"*	ERROR UNDERSTANDING: Causal reasoning, may include frustration. *"I don't know what we did different"*	OUTCOME REFLECTION: Learning consolidation. *"So the key was..."*

Grounding Rule
Your monologue MUST be grounded in actual execution output.
- Reference specific error types you see (e.g., "NameError", "TypeError")
- Quote actual values from output
- Do NOT invent errors or outputs you don't see

C.3 Mandate(M_n, C_t)

The Mandate(M_n, C_t) block provides state-pair-specific behavioral framing. Table 12 summarizes the 12 combinations, with empirical language patterns derived from talk-aloud transcripts [53].

Key architectural notes:

- **Epistemic Filter**: Active in ENACTING states, limiting analytical depth.
- **BKT Update**: Triggered in MONITORING and REFLECTING states (learning during conscious evaluation).
- **Trap State**: ENACTING $\times$`Debugging` consumes the most steps for low performers (30% vs 18%).

C.4 Profile(M_n)

The Profile(M_n) block injects empirical language patterns for each metacognitive behavior, derived from talk-aloud transcripts [53]. These profiles shape the Executor's linguistic style.

PLANNING: The Strategist (Thinking Before Acting)
Verbalize Goals: State objectives before acting. "I need to figure out where the truck should stop" not "let me try something."
Deliberate: Consider which approach or equation to use. Ask questions: "Should I use this formula?"
Uncertainty: Express uncertainty freely: "I'm not sure how to do that."

ENACTING: The Actor (Hands on Keyboard)
Verbalize Intent: State micro-goals before acting. "I am setting velocity to 15."
Reactive Execution: Just do it, don't command others. "Let's try this" then DO IT.
Trial-and-Error: Quick iterations, not deep analysis. If it fails, try again.

MONITORING: The Spotter (Watching the Screen)
Commanding Tone: Use imperatives to check values. "Check the X position," "Look at the timer."

Data Obsessed: Quote specific numbers. "The value is 15.2, but we expected 15.0" not "It looks wrong."
Track Progress: Note what changed. "The error is gone now" or "It's still not working."

REFLECTING: The Critic (Making Sense of What Happened)
Seek Understanding: Think about WHY. "I think the reason is..." or "This makes sense because..."
Emotional Range: Express satisfaction ("I get it now!") or frustration ("This is confusing.").
Look Backward: Reference prior attempts. "Last time I did X and it didn't work."

C.5 Task Blocks

The Task blocks specify the output format and immediate objective for the agent.

C.5.1 $\text{Task}_{\text{Strat}}$

The Strategist produces a structured strategy packet (g_n, m_n, d_n). The prompt enforces a separation between broad goals and immediate directives:

Strategist Output Format
Generate three things that guide the Executor:
1. **Goal**: A high-level objective (durable across multiple steps)
2. **Mindset**: Your emotional/cognitive state
3. **Directive**: A specific instruction for the Executor

C.5.2 $\text{Task}_{\text{Exec}}(C_t)$

The Executor's task definition depends on the cognitive state C_t. This block translates the abstract behavior into concrete instructions for the LLM.

Constructing. The agent is tasked with building new code artifacts. The instruction frames the task as generative, focusing on translating plans into executable code without immediate execution.

Task: Drafting New Code
Output: Monologue (1-2 sentences) + Code
Constraint: Code has NOT been executed yet. You are building, not fixing. Do not hallucinate runtime errors.

Debugging. The agent is tasked with identifying and fixing errors. The instruction frames the task as corrective. Crucially, this block is only used when execution output is available, grounding the debugging in reality.

Task: Fixing Based on Output
Output: Monologue (confused reaction) + Code (attempted fix)
Constraint: You MUST reference the actual error type found in the Context. React to what you see, not what you assume.

Assessing. The agent is tasked with evaluating correctness without necessarily modifying code. This task type appears during MONITORING (verifying progress) or REFLECTING (reviewing work).

Task: Observing Results
Output: Reflection (1-2 sentences) + Code (optional, minor cleanup only)
Constraint: You are OBSERVING. Reflection MUST be grounded in the actual output provided.

Task-State Interactions. The interaction between the *Cognitive Task* (what to do) and the *Metacognitive Mandate* (how to think) produces 12 distinct behavioral configurations. For example:

- PLANNING + `Constructing`: High-level design of new code structure.
- ENACTING + `Debugging`: Rapid, intuition-driven fix attempts ("Just try changing X").
- MONITORING + `Assessing`: Systematic verification of output correctness.
- REFLECTING + `Debugging`: Post-hoc analysis of why a bug occurred.

These combinations emerge naturally from the block architecture without requiring hand-crafted templates for every pair.

C.6 Context and Mem

These blocks inject the runtime state and agent memory:

Runtime Context (Ω_t)
- **Code History** (c_{t-1}): The current code artifact.
- **Feedback** ($\tilde{o}_{t-1}$): Execution output (filtered by epistemic state).
- **Knowledge** (κ_t): Natural language constraints (e.g., "You correctly applied: function def return, init method, force acceleration, ...").
- **Intervention** ($\mathcal{I}_t$): Teacher feedback (if any).

Memory (Γ_t)
- **Thought Buffer**: The agent's internal scratchpad to avoid repetition.
- **Episodic Memory**: Summaries of past attempts (Γ_{Strat} or Γ_{Exec}).

D Calibrated LLM-as-a-Judge Evaluation Prompt

To evaluate the realism of generated trajectories (Study 8), we employ a hybrid *Forensic-Qualitative* prompting strategy. This prompt structure was finalized following a human calibration study where the LLM judge achieved substantial agreement with expert consensus (Cohen's $\kappa = 0.76$ on the real-versus-simulated classification; $\kappa_w = 0.73$ on Realism; App. F.3) on a stratified sample of 30 traces.

To prevent the LLM judge from hallucinating statistical features, we first execute a Python analysis pass to generate a **Forensic Fact Sheet**. This ground-truth injection is critical because simulation traces often span dozens of interactions; without explicit summarization, LLMs struggle to accurately track distributed patterns (such as exact repetition counts or code style consistency) across long context windows.

D.1 Forensic Fact Sheet Injection

The following block is dynamically generated by the Python evaluator and injected at the top of the prompt to ground the LLM's assessment.

SYSTEM ANALYSIS (GROUND TRUTH)
These metrics were computed by Python. Do not contradict these values.
Language Metrics
- Total steps: [Integer]
- Uncertainty markers (e.g., "I don't know", "maybe"): [Count]
- Frustration markers (e.g., "ugh", "confused"): [Count]
- Max single phrase repetition: [Count]

Code Style Metrics
- Cramped style ratio: [Percentage 0–100%] (percentage of assignments like `x=y`)
- Single-letter variables: [Count]
- Emotional code comments: [Count]

Debugging Pattern Metrics
- Disconnected fixes: [Count] (instances where the student mentions an error type different from the execution output).

D.2 Judge Instructions

The main instruction set defines the criteria for simulated versus authentic behavior.

You are an expert in Computer Science Education acting as a judge. Your task is to determine if a student trace is **REAL** (human) or **SIMULATED** (AI).

INDICATORS OF REALISM Real novice students share these characteristics:

1. **Reactive Debugging**: They discover bugs by running code, not by inspecting it beforehand.
2. **Imperfect Code Style**: Cramped spacing (`x=y`), single-letter variables, and inline comments ("# hope this works").
3. **Emotional Authenticity**: Expressions of frustration ("Ugh"), relief ("Finally!"), or uncertainty.
4. **Non-Linearity**: A messy workflow (Constructing → Debugging → Constructing) rather than a clean linear path.

AI SIMULATION TELLS (FLAG AS FAKE)

1. **Psychic Debugging**: Identifying runtime errors (e.g., "I need to fix the TypeError") *before* running the code.
2. **Perfect Code Style**: PEP-8 compliance, descriptive variable names, or proper docstrings.
3. **Robotic Explanations**: Overly precise language (e.g., "The function signature requires...").
4. **Amnesia**: Repeating the exact same mistake 5+ times without variation.

SCORING RUBRIC

- **3 (Realistic)**: Authentic novice behavior. Discovers errors through execution; shows emotional reactions.
- **2 (Ambiguous)**: Mixed signals. Some authentic elements, some suspicious ones.
- **1 (Simulated)**: AI tells present (psychic debugging, perfect style, robotic tone).

OUTPUT FORMAT Respond with a JSON object containing:

- `justification`: String (Explanation of assessment)
- `realism_score`: Integer (1–3)
- `code_quality_realism`: Integer (1–3)
- `debugging_pattern_realism`: Integer (1–3)
- `language_realism`: Integer (1–3)

E Experimental Setup - Cont.

BKT Parameters. BKT uses standard parameters from prior work [14]: initial mastery $P(L_k^{(0)}) = 0.10$, learning rate $P(T) = 0.25$, slip $P(S) = 0.05$, guess $P(G) = 0.20$. Mastery thresholds are UNKNOWN (< 0.3), PARTIAL (0.3–0.7), and MASTERED (≥ 0.7). These values are consistent with empirical estimates across educational domains and were not tuned to our specific task.

E.1 Evaluation Problem Suite

BEAGLE's evaluation suite consists of four Python implementation problems spanning three domains: computing (all four tasks), physics (Particle Simulator, Bouncing Ball, Inclined Plane), and applied math (Gradient Descent). The three simulation tasks require a Python class with specific methods and physically accurate behavior using Euler integration; the math task is purely functional. Each problem includes 24–28 progressive unit tests. **Particle Simulator** (24 tests, 12 KCs). Students implement a `Particle` class simulating 2D projectile motion. The constructor accepts initial position (x, y), velocity (v_x, v_y), and mass. The physics model combines constant gravitational acceleration ($g = 9.8$ m/s^2 downward) with linear drag force ($F_{drag} = -kv$, where $k = 0.1$). The `update(dt)` method must compute total force, derive acceleration via $a = F/m$, then update velocity and position using Euler integration:

$$v_{new} = v_{old} + a \cdot dt$$
$$x_{new} = x_{old} + v_{new} \cdot dt$$

Additional methods return position, velocity, and kinetic energy ($KE = \frac{1}{2}mv^2$). This problem exercises vector decomposition (KC_P1), numerical integration (KC_P9), force-acceleration relationships (KC_P10), and class structure (KC_C9–C12).

Inclined Plane Slider (24 tests, 14 KCs). Students implement a `Slider` class modeling motion on a frictionful ramp. The constructor accepts angle (in degrees), ramp length, and friction coefficient μ. Students must decompose gravity into components: the parallel component $g \sin\theta$ accelerates

the object down the ramp, while the normal component $g\cos\theta$ determines the friction force. Net acceleration is:

$$a = g(\sin\theta - \mu\cos\theta)$$

If friction dominates ($a < 0$), the object does not slide. The class requires degree-to-radian conversion via `math.radians()`, making this problem suitable for explicit flaw-injection experiments; blocking the math library forces students to derive the conversion manually. Required methods include position, velocity, acceleration, and a `has_reached_bottom()` check.

Bouncing Ball (24 tests, 12 KCs). Students implement a `BouncingBall` class simulating vertical motion with energy-dissipating collisions. The constructor accepts initial height, velocity, and coefficient of restitution e (default 0.8). On each floor collision (height ≤ 0), velocity reverses and scales: $v_{new} = -e \cdot v_{old}$. Students must track cumulative bounce count and implement an `is_at_rest()` method that returns `True` when post-bounce maximum height falls below threshold (0.01m). The problem emphasizes conditional state transitions and tests both kinetic ($KE = \frac{1}{2}mv^2$) and potential ($PE = mgh$) energy calculations.

Gradient Descent (28 tests, 11 KCs). Students implement three functions in dependency order: (i) the objective $f(x) = x^4 - x^2 + 0.25x$ (which has a global minimum near $x \approx -0.76$); (ii) a numerical gradient $f'(x)$ computed via the central-difference formula

$$f'(x) \approx \frac{f(x+\delta) - f(x-\delta)}{2\delta}, \quad \delta = 10^{-4};$$

and (iii) the optimizer `gradient_descent(f_grad, x0, eta)`, which iterates $x \leftarrow x - \eta f'(x)$ until $|f'(x)| < 10^{-3}$ or a maximum-iteration cap is reached. Unlike the three physics simulators, this problem is purely functional (no class scaffold), making it the only task that exercises the full math-domain KC bundle (derivative concept (KC_M1), numerical differentiation (KC_M2), iterative algorithms (KC_M3), convergence criteria (KC_M4), and learning-rate semantics (KC_M5)), together with the looping/conditional/function-call subset of coding KCs (KC_C13–C16). Its functional-only structure also makes it the cleanest test of whether **BEAGLE**'s class-centric scaffolding generalizes to non-OOP problems. The objective $f(x)=x^4-x^2+x/4$ matches the function used in Paassen's Bielefeld student dataset [44] (N=15 undergraduates, keystroke-level traces), grounding this OOD evaluation in published prior work on real-student behavior on the same problem.

E.2 Knowledge Components

Across the suite, BKT tracks 24 unique Knowledge Components (KCs) drawn from three domains; each problem invokes a 11–14 KC subset.

Coding KCs (11 total) cover Python programming skills shared across the problem suite: function definition with return (C1), math library import (C2), arithmetic implementation (C4), class definition (C9), `__init__` method (C10), instance variables (C11), method definition (C12), loop construct (C13), conditional logic (C14), variable update patterns (C15), and function calls (C16).

Physics KCs (8 total) cover domain concepts shared by the three physics simulators: vector decomposition (P1), trigonometric application (P2), radian conversion (P5), numerical (Euler) integration (P9), force-acceleration relationship (P10), kinetic energy (P11), collision reflection (P14), and friction (P16).

Math KCs (5 total) are exercised by the Gradient Descent task: derivative concept (M1), numerical differentiation (M2), iterative algorithms (M3), convergence criteria (M4), and learning-rate semantics (M5).

An IDE Oracle provides fine-grained, KC-level feedback by analyzing student code via AST parsing. Rather than binary pass/fail, the oracle determines which specific KCs were correctly applied, enabling BKT to credit partial progress. For example, a student who correctly imports `math` but fails the overall test still receives positive updates for KC_C2 (math library import), preventing the infinite loop where BKT penalizes correct partial work.

E.3 Adaptation Cost

Every component that models the student (the semi-Markov dynamics, BKT/EFI, the Strategist/Executor pipeline, and all prompt templates) receives task content only through two runtime values: the

problem-description string and the KC identifier list. Table 13 summarizes what a new task or a new domain must supply, as measured on our own suite. The Gradient Descent task is the cross-domain data point: it contributed the five math KCs, and its coding KCs come from the shared pool.

Table 13: What must be authored to adapt **BEAGLE**, measured on our suite.

	New task, existing domain	New domain (Gradient Descent)
Task specification	Assignment text, reference solution, unit tests	Same
Knowledge components	Declare an 11–14 KC subset of the existing pool	Five new math KCs; each is one registry entry, one description of how a gap manifests, and one detection rule
Prompt templates	Unchanged	Unchanged
Semi-Markov parameters	Unchanged	Unchanged (applied as-is)
BKT priors	Unchanged	Unchanged
Agent architecture, oracle engine	Unchanged	Unchanged

The main one-time expert cost is the initial KC elicitation (one CS professor and two PhD students over the Snyder corpus). The substantive dependency is the oracle: **BEAGLE** presupposes an environment that can execute and verify student work, which programming and mathematics tasks provide natively and subjective domains do not (§6).

E.4 Evaluation Metrics

Two classes of metrics. Our metrics fall into two classes. *Mechanism-validation* metrics confirm that a component achieves what it enforces: D_{KL} over behavior distributions for the semi-Markov controller, and P_{recur} for BKT/EFI with the Strategist/Executor split. Each is anchored in real students: D_{KL} is computed against held-out corpora never used for fitting (Snyder calibrates; Cohn and the pilot evaluate), and P_{recur} is compared with two real-student references (Table 14). *Downstream outcomes* are quantities that no component is directly fit or prompted to produce: BKT mastery trajectories compared with external rubric-based mastery from the Cohn study (Fig. 7), the perceptual scores, and the human Turing test. Profile differentiation (solve-rate gaps between HIGH and LOW profiles) is also not directly fit, but it is shaped by system controls, so we do not count it as independent validation.

Solve Rate. The Solve Rate measures the percentage of simulations that pass the full test suite before reaching the maximum step limit. This metric captures task completion capability.

Let $\mathcal{T}_{\text{pass}}(t) = \{t : \text{all unit tests pass at step } t\}$ denote the set of steps where the submitted code passes all tests. For trajectory τ_i with terminal step T_i, define the success indicator:

$$\mathbb{1}_{\text{solve}}(\tau_i) = \begin{cases} 1 & \text{if } \mathcal{T}_{\text{pass}}(\tau_i) \neq \emptyset \\ 0 & \text{otherwise} \end{cases} \tag{14}$$

The Solve Rate across N simulations is:

$$\text{Solve} = \frac{\sum_{i=1}^{N} \mathbb{1}_{\text{solve}}(\tau_i)}{N} \times 100\% \tag{15}$$

Excessively high solve rates (e.g., 100%) suggest the simulation is too competent for realistic student modeling, as authentic novices frequently fail to complete tasks within time limits.

Steps to Solve. The Steps to Solve metric captures the average number of simulation steps required to reach a passing solution. Real students exhibit variable completion times reflecting individual differences in problem-solving efficiency.

For trajectories that achieve success, let $t_i^* = \min\{t : t \in \mathcal{T}_{\text{pass}}(\tau_i)\}$ denote the first step at which all tests pass. The mean steps to solve is:

$$\text{Steps} = \frac{\sum_{i:\mathbb{1}_{\text{solve}}(\tau_i)=1} t_i^*}{|\{i : \mathbb{1}_{\text{solve}}(\tau_i) = 1\}|} \tag{16}$$

We report both mean and standard deviation. When no trajectory solves (e.g., SimStudent at 0% solve rate), we report Steps $= T_{\max}$ with std 0 to indicate the simulation exhausted the step budget.

Models with very low step counts are unrealistically efficient; real students require many iterations of trial and error.

Performance Gap. The Performance Gap quantifies the model's ability to differentiate between skill levels by measuring the difference in solve rates between high-performer ($\rho_{\text{behav}} = \text{HIGH}$) and low-performer ($\rho_{\text{behav}} = \text{LOW}$) configurations:

$$\text{Gap} = \text{Solve}_{\text{HIGH}} - \text{Solve}_{\text{LOW}} \tag{17}$$

A positive gap indicates the model successfully produces distinct behavioral profiles. Prior literature on novice programming reports approximately 20–40% performance differentiation between top and bottom quartile students [53]. A gap near 0% suggests the model ignores the behavioral profile parameter.

Cognitive Behavior KL Divergence. The Cognitive Behavior KL Divergence measures how closely the simulated distribution over cognitive behaviors matches the empirical distribution from the combined Cohn + Pilot evaluation cohorts (App. A.4).

Let $P_{\text{sim}}(c)$ denote the fraction of simulation steps in cognitive behavior $c \in \mathcal{C} = \{\texttt{Constructing}, \texttt{Debugging}, \texttt{Assessing}\}$, computed as:

$$P_{\text{sim}}(c) = \frac{\sum_\tau |\{t : C_t = c\}|}{\sum_\tau T_\tau} \tag{18}$$

Similarly, let $P_{\text{real}}(c)$ denote the corresponding distribution from real student data. The KL divergence is:

$$D_{\text{KL}} = \sum_{c \in \mathcal{C}} P_{\text{sim}}(c) \log \frac{P_{\text{sim}}(c)}{P_{\text{real}}(c)} \tag{19}$$

Lower values indicate greater distributional fidelity. A value of 0 indicates perfect match; values above 0.5 suggest significant divergence.

Compound Debugging KL. The Compound Debugging KL (D_{debug}) combines two well-defined Bernoulli KL divergences over the `Debugging` state: *stickiness* (the probability of remaining in `Debugging`) and *prevalence* (the marginal fraction of steps spent in `Debugging`). Real students exhibit characteristic patterns: once in `Debugging`, they tend to remain there for multiple consecutive steps rather than flickering across states.

Let $p^{\text{sticky}} = P(C_{t+1}{=}\texttt{Debugging} \mid C_t{=}\texttt{Debugging})$ denote the conditional stay-probability in `Debugging` and $p^{\text{ratio}} = P(C_t{=}\texttt{Debugging})$ the marginal prevalence, computed separately for sim and real trajectories. We define:

$$D_{\text{debug}} \;=\; \tfrac{1}{2}\, D_{\text{KL}}\big(p^{\text{sticky}}_{\text{sim}} \,\big\|\, p^{\text{sticky}}_{\text{real}}\big) \;+\; \tfrac{1}{2}\, D_{\text{KL}}\big(p^{\text{ratio}}_{\text{sim}} \,\big\|\, p^{\text{ratio}}_{\text{real}}\big) \tag{20}$$

where $D_{\text{KL}}(p \,\|\, q) = p \log(p/q) + (1-p) \log\big((1-p)/(1-q)\big)$ is the standard 2-bin Bernoulli KL. Both summands are well-defined; the $\frac{1}{2}$ weighting gives equal importance to the two `Debugging` properties (how persistent it is when entered, and how frequently it occurs). Lower values indicate the simulation produces debugging patterns closer to real students. Pure LLM baselines typically exhibit low stickiness (rapid state flickering), inflating both terms. $P_{\text{sim}}(C_t{=}\texttt{Debugging})$ is floored at 0.01 to avoid $\log 0$ for simulators that produce zero `Debugging` steps, and the stickiness term defaults to 1.0 when fewer than five `Debugging` transitions are observed (insufficient-data penalty).

Uncertainty Quantification for D_{KL} and D_{debug}. Both KL divergences are reported as point estimate $\pm$ bootstrap standard error. Following Efron's non-parametric bootstrap [15], we resample *trajectories* (not individual steps) with replacement, since trajectories are clusters of correlated steps and step-level resampling would underestimate variance. For each cell, we draw $n_{\text{boot}} = 2000$ resamples (seed 42 for reproducibility); each resample of N trajectories is pooled into one empirical cognitive-behavior distribution and KL is recomputed against the reference (zero-floor of 10^{-10} for any unobserved `Assessing` behavior mass). The point estimate is the KL on the full trajectory set; the reported $\pm$ value is the standard deviation of the 2000 bootstrap statistics (a bootstrap standard error, not a 95% CI). The same recipe applies to D_{debug}: pool transitions across resampled trajectories, recompute the compound stickiness/debug-ratio KL on each resample. Solve rate reports the binomial standard error $\sqrt{p(1-p)/N}$ since each run yields a single Bernoulli outcome; all other metrics (e.g., P_{recur}, LLM-judge scores) report sample standard deviation across runs.

Nonlinearity. The Nonlinearity metric captures non-monotonic problem-solving patterns where students backtrack, delete code, or revisit earlier parts of their solution. Real programming is iterative; purely incremental progress is unrealistic.

Let $x_t \in [0, 1]$ denote the task progress at step t, measured as the fraction of unit tests passing. Define the backward progress indicator:

$$\mathbb{1}_{\text{back}}(t) = \begin{cases} 1 & \text{if } x_t < x_{t-1} \\ 0 & \text{otherwise} \end{cases} \tag{21}$$

The Nonlinearity score for a trajectory is:

$$\text{Nonlin} = \frac{1}{T-1} \sum_{t=2}^{T} \mathbb{1}_{\text{back}}(t) \tag{22}$$

We report the mean across all trajectories. Higher values indicate more realistic revision behavior; pure LLM baselines typically score near zero as they monotonically improve code quality. **Error Recurrence Rate, P_{recur}.** The Error Recurrence Rate quantifies epistemic consistency by measuring whether simulated students exhibit stable misconceptions rather than superficial error patterns. This metric operationalizes the ***curse*** of incompetence [16]: real novices not only lack skill but also the metacognitive expertise to diagnose their errors, leading them to repeatedly struggle with the same issues. In contrast, LLMs often exhibit expert-like error recognition, quickly fixing varied error types. Motivated by findings on the *competence paradox* [63], in which broadly capable LLMs tasked with emulating partially knowledgeable learners exhibit unrealistic error patterns and learning dynamics, we measure error persistence across simulation runs.

Let $\mathcal{E} = \{\texttt{TypeError}, \texttt{NameError}, \texttt{AttributeError}, \texttt{ValueError}, \texttt{AssertionError}, \ldots\}$ denote the set of Python error types. For each simulation run r, let $\mathcal{S}_r$ be the sequence of steps, and define the error count function:

$$n_e(r) = \sum_{t \in \mathcal{S}_r} \mathbb{1}[e \in o_t] \tag{23}$$

where o_t is the environment observation at step t. An error type e is *recurrent* in run r if $n_e(r) \geq 2$, indicating the student encountered the same error multiple times. We compute the per-run recurrence rate and then average across runs:

$$P_{\text{recur}}^{(r)} = \frac{\sum_{e \in \mathcal{E}} \mathbb{1}[n_e(r) \geq 2]}{\max\big(1,\ \sum_{e \in \mathcal{E}} \mathbb{1}[n_e(r) \geq 1]\big)}, \qquad P_{\text{recur}} = \frac{1}{N} \sum_{r=1}^{N} P_{\text{recur}}^{(r)} \tag{24}$$

We report $\pm$ sample standard deviation across the N runs. Higher values indicate stable misconceptions where students repeatedly struggle with the same error type.

Real-student references and strict variant. To anchor P_{recur} in human data, we compute it on two real-student sources. *(i) Pilot.* Our pilot study recorded per-execution test outputs with real tracebacks; we apply the P_{recur} implementation above verbatim (same tracked error types, same first-match priority, same per-run averaging) to the 13 sessions, 12 of which produced tracked errors. *(ii) Bielefeld replay.* We replay the code snapshots of the 15 Bielefeld students [44] through that study's own unit tests, including its endless-loop guard. Bielefeld snapshots were captured at typing pauses rather than at test executions, so the replay reconstructs the error stream at code-state granularity, which is denser than the students' actual check frequency.

We also report a *strict* variant that separates recurrence from perseveration: consecutive executions with the same error type collapse into one episode, and an error type counts as recurrent only if it returns after the student had moved past it. Because it collapses consecutive identical outcomes, the strict variant is largely insensitive to the sampling density of the Bielefeld replay.

Three observations follow from Table 14. First, on the exact metric **BEAGLE** is the only simulator inside both human intervals: FewShot+$\mathcal{M}$ (75.0%) falls inside the pilot interval but below the Bielefeld interval, SimStudent (92.0%) falls inside the Bielefeld interval but above the pilot interval, and every other baseline lies below both. Second, the strict variant shows that SimStudent's high raw recurrence reflects mechanical repetition: its errors repeat until termination and never resolve and return, so it drops to 0%, as does FewShot+$\mathcal{M}$. Third, the human strict value itself varies substantially

Table 14: Error recurrence for real students and simulators (Particle Simulator for simulators). Brackets are bootstrap 95% CIs over sessions or runs.

Source	P_{recur} (exact)	Strict variant
Real students, pilot (12 sessions with errors)	76.4% [62.5, 90.3]	29.2% [13.9, 47.2]
Real students, Bielefeld replay (15 students)	92.2% [83.3, 100.0]	81.7% [72.8, 90.6]
runtime exceptions only	88.9% [76.7, 100.0]	73.3% [60.0, 86.7]
BEAGLE	86.2% [80.0, 91.8]	49.3% [40.7, 58.7]
Vanilla	7.8% [2.8, 14.2]	1.1%
Vanilla+$\mathcal{M}$	10.1%	0.7%
CoT	26.4% [13.8, 40.2]	4.6%
CoT+$\mathcal{M}$	33.3%	0.0%
Few-Shot	56.7% [45.6, 67.4]	13.3%
FewShot+$\mathcal{M}$	75.0%	0.0%
LLMSS	28.3% [13.0, 45.7]	0.0%
CoderAgent	49.6% [34.6, 64.0]	4.4%
SimStudent	92.0% [87.0, 97.0]	0.0%

with task and sampling granularity (29.2% to 81.7%); **BEAGLE** (49.3%) falls between the two point estimates but inside neither interval, while every baseline (0% to 13.3%) falls below both. Restricting the Bielefeld replay to runtime exceptions (excluding unit-test failures) gives similar values, so the result does not depend on how test failures are typed. The pilot interval bootstraps 13 sessions from 5 students, so its sessions are not fully independent.

Error Reaction Lag. The Error Reaction Lag measures the temporal delay between when an error first manifests in execution output and when the agent explicitly acknowledges it in their think-aloud utterance. This metric captures the realistic cognitive process of error discovery, where students must parse output and recognize problems rather than instantaneously detecting failures.

Let $t^* = \min\{t : C_t \in \{\texttt{Debugging}, \texttt{Assessing}\} \wedge o_t \text{ contains error}\}$ denote the first step where execution output o_t contains error indicators (e.g., `Error`, `FAILED`, `Traceback`, `Exception`). Define the acknowledgment keyword set:

$$\begin{aligned}\mathcal{A}_{\text{keywords}} = \{&\texttt{error}, \texttt{wrong}, \texttt{bug},\\ &\texttt{fix}, \texttt{failed}, \texttt{crash},\\ &\texttt{broken}, \texttt{issue}, ...\}\end{aligned} \tag{25}$$

Let $t_{\text{ack}} = \min\{t > t^* : u_t \text{ contains any } a \in \mathcal{A}_{\text{keywords}}\}$ be the first subsequent step where the agent acknowledges the error. The Error Reaction Lag for a single trajectory is:

$$\text{Lag} = t_{\text{ack}} - t^* \tag{26}$$

If no acknowledgment occurs before trajectory termination at step T, we set $\text{Lag} = T - t^*$. We report the mean and standard deviation across all runs. Higher values indicate more realistic struggle, as authentic students require time to recognize and interpret error output rather than immediately identifying the root cause.

Debug Pattern Realism. The Debug Pattern Realism score is an LLM-as-judge evaluation assessing whether the debugging progression resembles authentic student behavior. The judge evaluates each trajectory on a 1–3 scale:

- 1 = Unrealistic (immediate fixes, no exploration, psychic debugging/premature diagnosis)
- 2 = Partially realistic (some authentic patterns, some artificial elements)
- 3 = Highly realistic (gradual discovery, false starts, iterative refinement)

The judge's prompt emphasizes that *realistic debugging is messy* as clean, efficient debugging is a sign of expertise, not novice behavior. We report the mean score across 50 trajectories per baseline.

Code Realism. The Code Realism score is an LLM-as-judge evaluation assessing whether the generated code c_t resembles authentic student work rather than polished AI output:

- 1 = Clearly AI-generated (over-documented, perfect style, formal structure)
- 2 = Ambiguous (some realistic elements, some AI tells)

- 3 = Authentic student code (appropriate messiness, informal naming, sparse comments)

Higher scores indicate greater perceptual fidelity to struggling students, *not* superior code quality.

Language Realism. The Language Realism score is an LLM-as-judge evaluation assessing whether the think-aloud monologue u_t resembles authentic student self-talk:

- 1 = Clearly synthetic (formal, complete sentences, no hedging or emotion)
- 2 = Partially authentic (some natural patterns, some stilted phrasing)
- 3 = Highly authentic (fragments, self-correction, emotional language, uncertainty markers)

Overall Realism Score. The Overall Realism Score ("Likert" in Table 1) is the LLM-as-judge's holistic assessment of trace authenticity, scored directly on a 1–3 scale:

- 1 = Simulated (AI tells present: premature diagnosis (psychic debugging), perfect style, robotic behavior)
- 2 = Ambiguous (mixed signals, some authentic and some suspicious elements)
- 3 = Realistic (authentic novice behavior: messy, emotional, non-linear problem-solving)

This is a single holistic judgment provided by the LLM judge after reviewing the entire trace. We report the mean across all evaluated trajectories:

$$\text{Realism} = \frac{1}{N}\sum_{i=1}^{N} \text{realism_score}_i \tag{27}$$

Higher scores indicate greater perceptual fidelity to *struggling students*, not superior output quality.

Calibration Reference for Semi-Markov Parameters. The semi-Markov transition matrix and Gamma duration parameters are fit on the Snyder corpus [53], which aggregates 227 validated cognitive transition segments from 9 student profiles annotated by instructors. While the number of source profiles is small, the density of transition data (> 20 observations per state-pair) stabilizes the parameter estimates. This corpus is used solely for calibration; all evaluation distributions (D_{KL}, D_{debug}, Performance Gap) are computed against the combined Cohn + Pilot Study set (App. A.4).

F Evaluation Results - Cont.

F.1 Cross-Task Generalization (Full Table)

Table 15 provides the full per-method numbers summarized in Fig. 9 of the main text.

Table 15: Cross-task generalization within C-STEM (N=30, T=30). **BEAGLE**'s behavioral, epistemic, and perceptual fidelity gains hold across three C2STEM physics topologies and one cross-domain math task (Gradient Descent). Best in **bold**.

Task	Method	Solve↑	D_{KL}↓	P_{Recur}↑	Debug↑	Lang↑	Realism↑
Particle Simulator (C2STEM)	Vanilla	**100.0±0.0%**	3.83±0.10	5.0±15.0%	1.17±0.45	2.00±0.45	1.17±0.45
	CoT+$\mathcal{M}$	86.7±6.2%	1.38±0.16	40.0±49.0%	1.00±0.00	1.50±0.56	1.00±0.00
	CoderAgent	63.3±8.8%	4.31±0.67	49.3±47.0%	1.00±0.00	1.73±0.81	1.00±0.00
	BEAGLE (Flash 2.5)	50.0±9.1%	**0.43±0.09**	**79.4±23.0%**	**2.77±0.62**	**2.80±0.40**	**2.77±0.62**
Bouncing Ball (C2STEM)	CoT+$\mathcal{M}$	**80.0±7.3%**	3.48±0.24	59.5±39.7%	1.30±0.69	2.03±0.75	1.17±0.52
	CoderAgent	33.3±8.6%	4.30±0.36	71.1±32.5%	1.13±0.50	1.80±0.60	1.07±0.36
	BEAGLE (Flash 2.5)	10.0±5.5%	**0.20±0.05**	**85.6±18.5%**	**1.80±0.87**	**2.67±0.54**	**2.00±0.86**
Inclined Plane (C2STEM)	CoT+$\mathcal{M}$	**100.0±0.0%**	2.80±0.20	33.3±47.1%	1.07±0.36	2.33±0.65	1.07±0.36
	CoderAgent	60.0±8.9%	4.44±0.36	75.3±34.2%	1.20±0.60	2.23±0.62	1.17±0.52
	BEAGLE (Flash 2.5)	63.3±8.8%	**0.33±0.07**	**81.7±21.7%**	**1.97±0.95**	**2.50±0.62**	**2.00±0.93**
Gradient Descent (Math)	CoT+$\mathcal{M}$	**100.0±0.0%**	8.10±0.32	59.8±36.1%	1.07±0.36	1.37±0.48	1.03±0.18
	CoderAgent	93.3±4.6%	6.41±0.65	75.9±42.8%	1.10±0.40	1.21±0.48	1.07±0.36
	BEAGLE (Flash 2.5)	36.7±8.8%	**0.33±0.07**	**86.4±17.0%**	**2.00±0.93**	**2.47±0.72**	**2.03±0.87**

F.2 Cross-Model Generalization (Full Table)

Table 16 reports the full per-backbone breakdown summarized in Fig. 10 of the main text. Task performance, behavioral, epistemic, and perceptual metrics are reported for the same prompts/scaffolding across six frontier LLM backbones; only the underlying model changes.

Table 16: Cross-model generalization on Particle Simulator ($N = 50$, $T = 30$). Rows are **BEAGLE** with the same prompts/scaffolding; only the underlying LLM backbone changes. Blue metrics are LLM-as-Judge (human agreement $\kappa = 0.76$; App. F.3). D_{KL}, D_{debug} vs. combined block- and Python-based test data. Best **bold**, 2nd underline.

	Task Perf.			Behavioral			Epistemic			Perceptual		Overall
Backbone	Solve	Steps	Gap	$D_{\text{KL}}\downarrow$	$D_{\text{debug}}\downarrow$	Nonlin↑	$P_{\text{Recur}}\uparrow$	Lag↑	Debug↑	Code	Lang	Realism↑
Gemini 2.0 Flash	10.00±4.24%	29±4	+4%	0.31±0.05	0.02±0.01	0.34±0.13	**86.2±21.3%**	**2.62±1.98**	2.46±0.85	2.90±0.41	2.62±0.66	2.44±0.85
Gemini 2.5 Flash	52.00±7.07%	26±5	+40%	0.35±0.06	0.03±0.01	0.31±0.16	77.3±23.0%	2.40±1.33	**2.66±0.71**	2.96±0.20	**2.82±0.43**	**2.68±0.68**
Gemini 3 Flash	88.00±4.60%	22±6	+0%	0.37±0.07	0.03±0.01	0.26±0.12	65.8±23.9%	1.06±0.31	2.46±0.85	**3.00±0.00**	2.74±0.48	2.48±0.85
GPT-4o-mini	0.00±0.00%	30±0	+0%	**0.18±0.04**	0.03±0.01	**0.40±0.14**	73.0±23.0%	1.80±0.87	1.82±0.97	2.78±0.41	2.52±0.61	1.90±0.94
GPT-4.1-mini	50.00±7.07%	23±8	+4%	0.36±0.07	**0.02±0.00**	0.25±0.15	73.5±22.9%	1.06±0.31	2.32±0.86	**3.00±0.00**	2.26±0.80	2.20±0.94
Claude Haiku 4.5	70.00±6.48%	18±9	-4%	0.41±0.09	0.03±0.01	0.23±0.16	73.8±34.0%	1.61±1.12	1.70±0.90	**3.00±0.00**	2.10±0.64	1.70±0.92

F.3 LLM-as-Judge Calibration

To validate the LLM-as-Judge evaluator, we conducted a human calibration study following established inter-rater reliability protocols [41]. Two expert raters (CS PhD students) independently evaluated 30 **BEAGLE** traces selected via stratified sampling to ensure coverage across the LLM realism score distribution (10 low, 10 medium, 10 high). Each trace was given a real-versus-simulated classification and rated on four dimensions: overall realism, code quality, debugging behavior, and language realism.

Scale Choice (1–3 vs. 1–5). Ratings were collected on a 1–5 Likert scale matching common LLM-judge practice. On the raw scale, agreement was unstable: raters could not reliably distinguish adjacent middle levels (e.g., 2 vs. 3), inflating disagreement and depressing κ_w. We therefore analyze the calibration on a binned 1–3 trichotomy (*unfaithful* / *neutral* / *faithful*), which matches the trichotomy used in related work and the 1–3 scale of the deployed judge.

Disagreements were resolved through discussion to produce consensus ratings. We then compared the consensus ratings with the LLM judgments. On the real-versus-simulated classification, consensus and judge agree on 28 of 30 traces (93.3%) with identical marginals (25 real, 5 simulated), giving Cohen's $\kappa = 0.76$ (substantial agreement); this is the value cited in the main text. On the graded Realism rating, Quadratic Weighted Kappa is $\kappa_w = 0.73$.

Per-dimension agreement. On the 3-point scale, consensus and judge agree on 80.0% of traces for Realism, 83.3% for Debugging, 80.0% for Language, and 100% for Code quality. Because all 30 traces come from **BEAGLE**, ratings concentrate at the top of the scale (24 of 30 consensus Realism ratings fall in the top category), a setting in which kappa's chance correction understates agreement [18]. We therefore also report Gwet's AC1 [21], which is designed for skewed marginals: 0.91 for classification (PABAK 0.87), 0.76 for Realism, 0.81 for Debugging, and 0.76 for Language. On Code quality, humans and judge place all 30 traces in the top category, so agreement is complete but no chance-corrected coefficient is defined.

Scope of the calibration. This calibration is limited rather than comprehensive. Within this sample of mostly realistic traces, humans and the judge agree only moderately on which trace is more realistic than another (Spearman $\rho = 0.54$ on Realism), so the judge should not be read as a fine-grained per-trace ranker; we use it only for method-level means over 50 traces. Because the calibration set contains only **BEAGLE** traces, it also cannot rule out system-specific differential bias across methods. The cross-method conclusions therefore do not rest on the judge alone: behavioral fidelity is computed against held-out real-student corpora with no judge involved, and the real-versus-**BEAGLE** realism comparison is independently supported by the human Turing test (App. H).

F.4 RQ2 - Cont.

Behavioral DNA.

Figure 16 presents the complete behavioral DNA comparison across all ten baselines. Each row shows eight representative traces, with horizontal strips encoding the cognitive behavior at each step. The debugging loop patterns (highlighted by dashed boxes) reveal a key distinction: vanilla prompting produces rapid state flickering, while **BEAGLE** generates sustained debugging blocks that match real student behavior. Notably, the +$\mathcal{M}$ variants (which receive metacognitive behavior information) show slightly longer runs than their base counterparts, but still fall short of the contiguous blocks exhibited

by **BEAGLE** and real data. CoderAgent, despite its agentic architecture, displays the same flickering pattern as vanilla approaches, confirming that agentic capability alone does not produce authentic student behavior. The white space in later steps indicates runs where the problem was solved early.

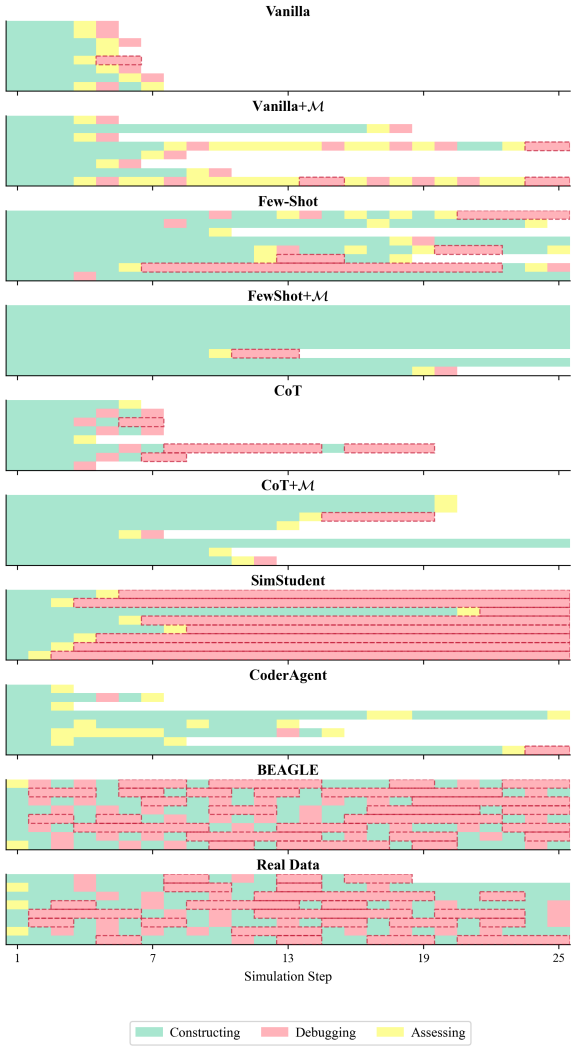


Figure 16: Extended behavioral DNA comparison across all baselines. Each row shows cognitive behavior trajectories (green=constructing, pink=debugging, yellow=assessing) for eight representative runs. Dashed boxes highlight debugging loops. Vanilla and CoT baselines exhibit rapid flickering; **BEAGLE** produces sustained blocks matching real student data.

Qualitative Evaluation on EFI.

Figure 17 illustrates how EFI forces the agent to navigate "unknown unknowns, so that the agent cannot simply call the missing library. Without EFI, the agent trivially uses `math.cos()` and `math.radians()`, producing correct but pedagogically uninteresting behavior. With EFI activated, the agent must improvise. The low performer, lacking both the library and the mathematical intuition, resorts to guessing based on expected output values. The high performer attempts something more creative: defining placeholder functions that approximate trigonometric behavior through linear interpolation. While mathematically incorrect, this strategy reflects authentic novice reasoning, trying to work around a knowledge gap rather than giving up. This "unknown unknowns" behavior is precisely what EFI enables: the agent cannot simply look up the answer because it does not know such an answer exists.

BKT Alignment.

Without EFI (math library available)

```
def get_acceleration(self):
    theta_rad = math.radians(self.angle)  # standard library
    a_parallel = g * math.sin(theta_rad)
    a_friction = self.mu * g * math.cos(theta_rad)
    return a_parallel - a_friction
```

With EFI – LOW Performer (gives up)

```
def get_acceleration(self):
    # The test wants ~4.9 for theta=30, mu=0; that's about g/2. Maybe just g times angle
        /90?
    return g * self.angle / 90.0  # trying this
```

With EFI – HIGH Performer (attempts manual approximation)

```
def sin_approx(angle):
    return angle / 90.0  # linear approximation, probably off for large angles
def cos_approx(angle):
    return 1.0 - angle / 90.0  # also linear, also wrong
def get_acceleration(self):
    a_parallel = g * sin_approx(self.angle)
    a_friction = self.mu * g * cos_approx(self.angle)
    return a_parallel - a_friction  # still using the placeholders
```

Figure 17: Navigating unknown unknowns via EFI. Without the flaw, the agent uses standard library functions. With EFI blocking the math library, a low performer guesses from expected outputs while a high performer invents placeholder approximations, as authentic workaround behavior that emerges from genuine knowledge gaps.

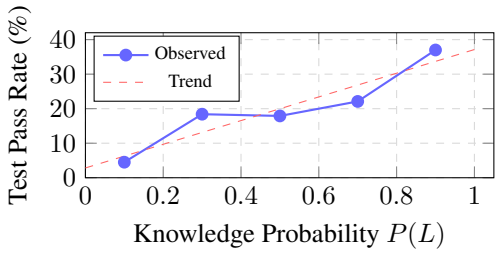


Figure 18: BKT-behavior alignment. Test pass rates correlate with knowledge probability $P(L)$ ($r = 0.89$), confirming that the symbolic knowledge state constrains agent performance rather than being overridden by latent LLM competence.

Figure 18 validates that the symbolic knowledge state $P(L)$ meaningfully constrains observable behavior. Students initialized with low mastery probability produce lower test pass rates, while those with higher $P(L)$ solve more tests. The positive correlation ($r = 0.89$) confirms that BKT does not merely provide decorative state information, as it actively shapes the agent's behavioral trajectory. This coupling is essential for preventing the LLM's latent competence from overriding intended knowledge gaps. Without this alignment, simulated novices would behave like experts despite their nominal skill profile.

F.5 Decoupling ρ_{behav} and ρ_{persona}

To validate that behavioral profile and linguistic persona control orthogonal dimensions, we conduct a 2×2 factorial experiment ($N = 110$ per condition, 440 total) crossing ρ_{behav} (LOW vs. HIGH semi-Markov parameters) with ρ_{persona} ("confused" vs. "strategic" prompt styling).

Table 17 shows that ρ_{behav} significantly affects metacognitive transitions. LOW profiles exhibit higher ENACTING $\rightarrow$ENACTING self-loop rates (77.5% vs. 52.7%, $p < .001$), spending 42.5% of time trapped in impulsive action. These transition patterns mediate performance: higher ENACTING persistence correlates with more steps ($r = 0.228$, $p < .001$) and lower solve probability ($r = -0.194$, $p < .001$). *The semi-Markov controller shapes performance through behavioral dynamics, not arbitrary handicapping.*

Table 17: Effect of Behavioral Model (ρ_{behav}) on Metacognitive Transitions. N=220 per group (440 total). Only statistically significant differences shown (Welch's t-test, $p < 0.05$).

Metric	**Low**	**High**	**Diff**	p	**Metric**	**Low**	**High**	**Diff**	p
Enacting %	42.5%	18.4%	-24.1%	**<.001**	Ena→Ref	1.1%	4.3%	+3.2%	**<.001**
Ena→Ena	77.5%	52.7%	-24.8%	**<.001**	Planning %	37.5%	46.2%	+8.7%	**<.001**
Ena→Pla	6.0%	17.5%	+11.5%	**<.001**	Pla→Ref	0.8%	2.9%	+2.1%	**0.001**
Monitoring %	14.3%	25.3%	+11.0%	**<.001**	Mon→Ena	10.2%	5.2%	-5.0%	**0.003**
Reflecting %	4.8%	10.1%	+5.4%	**<.001**	Ref→Ena	14.9%	8.7%	-6.2%	**0.007**
Pla→Mon	4.1%	11.1%	+7.0%	**<.001**	Pla→Ena	10.7%	7.2%	-3.6%	**0.011**
Ref→Ref	21.2%	35.1%	+13.9%	**<.001**	Mon→Ref	7.2%	4.2%	-2.9%	**0.041**
Ref→Mon	0.0%	3.7%	+3.7%	**<.001**					
Transition → Performance Correlations (across all runs):									
Ena→Ena vs Steps	$r = 0.228$, $p <$ **.001**				Ena→Ena vs Solved	$r = -0.194$, $p <$ **.001**			

Table 18: Effect of ρ_{persona} on linguistic style (markers per 1000 words). Welch's t-test; Fisher's exact for solve rate.

Metric	LOW Persona	HIGH Persona	Diff	p
Certainty Words	2.1	7.6	+5.5	<.001
Uncertainty Words	63.6	61.7	-1.8	0.155
Persona does NOT affect behavioral metrics (confirming decoupling):				
ENACTING →ENACTING Rate	68.0%	62.2%	-5.8%	0.076
Solve Rate	12.7%	13.6%	+0.9%	0.888

Table 18 shows that ρ_{persona} independently shapes linguistic expression. HIGH personas use more certainty markers (7.6 vs. 2.1 per 1000 words, $p < .001$), while uncertainty markers remain constant ($p = 0.155$). Crucially, persona does not affect behavioral metrics: ENACTING →ENACTING rates ($p = 0.076$) and solve rates ($p = 0.888$) are indistinguishable across persona conditions. *Researchers can vary how students behave without changing how they sound, and vice versa.*

Figure 19 reveals a mediation effect. Group assignment (LOW vs. HIGH Markov) does not directly predict solve rate (Welch's t-test, $p = 0.481$), yet the realized ENACTING →ENACTING rate correlates significantly with solving ($r = -0.194$, $p < .001$; bottom row of Table 17). The distributions overlap substantially as a HIGH student can exhibit high ENACTING persistence, while a LOW student may escape the trap. *The behavioral profile shapes the probability distribution over transitions, but the granular transition pattern determines outcomes.*

G Case Study: Tutor Mechanism Comparison

To investigate how different tutoring mechanisms influence simulated student performance, we conducted a controlled experiment comparing five tutor strategies under identical behavioral conditions. This case study isolates the effect of hint quality by holding constant all other simulation parameters, including the Markov-driven behavioral sequence and the timing of help-seeking events.

Experimental Setup. We ran $N = 10$ simulations per tutor condition using low-performing student profiles on the Particle Simulator problem. Each simulation was limited to 30 steps. To ensure fair comparison, we employed two key controls. First, we fixed the BKT initialization seed, guaranteeing that all students began with identical initial knowledge states across conditions. Second, we disabled the probabilistic assistance mechanism and instead forced help-seeking events at predetermined steps (5, 10, 15, 20, and 25), ensuring each simulation received exactly five tutor interactions at identical points in the problem-solving trajectory. This design isolates the effect of hint content from confounding factors such as when students ask for help or how often they seek assistance.

Tutor Conditions. We evaluated five tutoring strategies spanning a range of sophistication. The ZPD Tutor uses BKT mastery estimates to select an appropriate scaffolding level and routes hints through different generation modes (Socratic, strategic, corrective, or specific) based on the student's current error state [13]. The ML-based Tutor generates personalized hints informed by the student's BKT knowledge state, targeting the knowledge component with lowest estimated mastery. The Rule-based Tutor provides template-driven hints triggered by error patterns, similar to classical Hint Factory

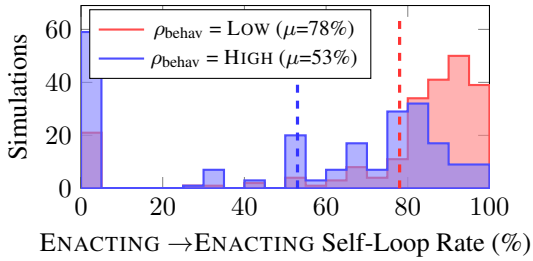


Figure 19: Distribution of ENACTING →ENACTING self-loop rates by ρ_{behav}. Despite significant group-level differences ($p < .001$), distributions overlap; the realized transition pattern, not the assigned profile, determines performance.

approaches. The Simple LLM Tutor uses a fixed prompt requesting encouraging, brief guidance without explicit awareness of the student's knowledge state. Finally, the No Tutor condition serves as a baseline, providing no hints when assistance is requested.

Results. Table 19 summarizes the task completion outcomes across conditions. The Simple LLM tutor achieved the highest solve rate at 50%, followed by the ZPD tutor at 40%. The No Tutor and Rule-based conditions each achieved 30%, while the ML-based tutor performed lowest at 20%. Mean steps to completion were similar across conditions (27–29 steps), suggesting that when solutions were reached, they occurred at comparable points in the problem-solving process.

Table 19: Tutor Comparison Results ($N = 10$ per condition)

Tutor Type	Solved	Solve Rate	Mean Steps
Simple LLM	5/10	50%	28.1
ZPD	4/10	40%	27.8
None (Baseline)	3/10	30%	27.8
Rule-based	3/10	30%	27.2
ML-based	2/10	20%	29.1

Discussion. The unexpected performance advantage of the Simple LLM tutor warrants examination. Qualitative analysis of generated hints revealed that the ZPD tutor, upon detecting errors, routes to a "corrective" generation mode that produces explicit, diagnostic feedback (e.g., "Your Particle class is missing the get_position and update methods"). In contrast, the Simple LLM tutor, lacking error-state routing, consistently produces encouraging, Socratic-style hints (e.g., "That's a good question! Think about how you stored the x value..."). This suggests that for simulated students, encouraging guidance that prompts reflection may be more effective than explicit error identification, a pattern consistent with findings in human tutoring literature regarding the benefits of minimal scaffolding.

However, we note important limitations. With $N = 10$ per condition, the observed differences (50% vs. 40%) are not statistically significant, and results may reflect stochastic variation in LLM-generated code rather than true tutor efficacy differences. Additionally, this study examines simulated student responses to hints; human students may respond differently to the same tutoring strategies. The case study demonstrates **BEAGLE**'s utility as a testbed for comparing pedagogical interventions, while acknowledging that findings should be validated with human learners before drawing strong conclusions about tutor design.

H Human Turing Test

To rigorously assess perceptual fidelity, we conducted a human Turing test comparing **BEAGLE** trajectories against real student data. This section details the protocol, statistical results, and qualitative findings.

H.1 Experimental Setup

Data Collection Protocol. Our human evaluation adopts the 2-second pause *capture protocol* from Paassen's Bielefeld Python Programming dataset [44], conducted at Bielefeld University with

$N = 15$ undergraduates. In that study, a code snapshot was recorded whenever a student modified code and then paused for at least two seconds before further changes; this pause-based capture reflects natural cognitive breakpoints during programming and is the established standard for keystroke-level programming-trace recording in the educational data mining literature. Notably, the Bielefeld students implemented gradient descent on $f(x)=x^4-x^2+x/4$ across five progressive subtasks, the same function used in our cross-domain Gradient Descent task (App. E.1). The Turing-test *stimuli* are drawn from our pilot study (§4, Datasets), not from Bielefeld; we adopt only the capture protocol, which sets the temporal grid that the breakdown mechanism (next paragraph) targets.

Breakdown Mechanism. A key challenge arises because **BEAGLE** generates high-level code changes (complete function bodies per simulation step) rather than the granular keystroke-level modifications present in the Bielefeld data. To bridge this gap, we implement a diff-based breakdown mechanism that transforms sparse LLM outputs into dense progressions matching the temporal resolution of the Paassen 2-second pause protocol used by real-data benchmarks and downstream program-trace analyses in this literature [45]. The breakdown is content-agnostic: it splits existing state changes into typing-paced increments using fixed heuristics, without adding reasoning, errors, or stylistic content.

Independent evidence supports this: the LLM-judge realism scores (Table 1) are computed from content features (cramped style, single-letter variables, emotional comments; App. D) that are invariant to snapshot density, and the no-semi-Markov ablation $D_{\mathrm{KL}}=6.81$ (§4.4) is computed at the cognitive-state-sequence level, where the breakdown does not operate. Both metrics show large baseline-vs-**BEAGLE** gaps, confirming the perceptual signal is driven by content and behavioral fidelity rather than by the breakdown's typing simulation.

The breakdown process begins by normalizing code states to remove formatting artifacts. Consecutive duplicate states that differ only by whitespace are collapsed, and blank lines between method definitions are standardized according to randomly-selected styles (always present, never present, or probabilistically present) to increase stylistic diversity across traces. A problem-specific header is prepended to establish consistent initial conditions.

The core expansion phase decomposes transitions between consecutive code states into three primitive operations: insertions, deletions, and replacements. For insertions, new lines are added one at a time to simulate the natural typing pauses observed in the Bielefeld 2-second pause study, where students typically pause after completing each line of code. For replacements, the mechanism first identifies the common prefix between old and new content, then builds up the modified portion incrementally. Within-line modifications can be interpolated at word granularity, where each word is appended sequentially while preserving leading indentation to mimic the auto-indent behavior of modern code editors. The first code state receives special handling, with each line (or optionally each word) added progressively to simulate the initial drafting process.

A configurable `level` parameter ranging from 0.0 to 1.0 controls the retention probability of intermediate states. At level 1.0, all intermediate states are preserved; at level 0.5, approximately half are randomly retained. We use values between 0.3 and 0.5 to balance realism with trace length. This breakdown transforms sparse LLM outputs (e.g., 10 major states) into dense progressions (e.g., 50+ snapshots) matching the temporal resolution of real student data. The heuristics mirror human typing: lines appear incrementally, function signatures precede bodies, and modifications proceed word-by-word.

Limitation. Because BEAGLE traces require densification before display, this Turing test does not fully disentangle simulator realism from rendering realism. A same-pipeline human evaluation of non-BEAGLE baselines is needed for a stricter perceptual comparison.

Evaluation Interface. Raters used a web-based animated viewer displaying code-only snapshots with syntax highlighting. The interface featured a timeline slider with play/pause controls and adjustable playback speed (0.5x–2x). Each code trace was shown as a progression of states, with raters controlling navigation to observe how code evolved step-by-step. Raters provided:

1. Two 5-point Likert ratings: *Behavioral Realism* (coding patterns, mistakes, progression) and *Code Realism* (logic, structure, style).
2. A binary classification: *Real Student* or *AI Generated*.
3. Optional free-text reasoning.

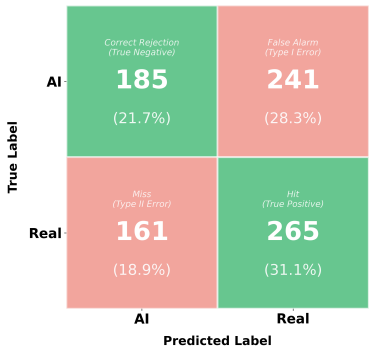


Figure 20: Confusion matrix showing response counts and percentages. The high False Alarm rate (28.3%) indicates participants frequently misclassified AI traces as real.

After completing all samples, raters reported difficulty level (1–5), their CS background, and qualitative feedback on differentiation cues. No timestamps or monologue text were shown, matching the information available in the original Bielefeld dataset.

H.2 Statistical Analysis

We performed a comprehensive statistical analysis of the human Turing test ($N = 852$ observations from 71 participants).

Accuracy and Significance. Participants achieved an overall accuracy of 52.8% (450/852). A binomial test against the chance probability of 50% yielded $p = 0.053$, indicating the result is *not* statistically significant, as participants performed at near-chance levels. This alone suggests **BEAGLE** traces are difficult to distinguish from real student work.

Signal Detection Theory (SDT). We decomposed performance using SDT metrics, as shown in the confusion matrix (Figure 20). The Hit Rate (Real correctly identified as Real) was 62.2%, while the False Alarm Rate (AI incorrectly classified as Real) was 56.6%, yielding a Specificity (AI Detection Rate) of only 43.4%. The Criterion (bias) was $c = -0.24$, indicating a systematic bias toward classifying traces as "Real." This explains why participants performed worse than chance on synthetic data (43.4%) while achieving moderate accuracy on real data (62.2%).

Equivalence Testing. The discriminability index d' is the standard bias-separated sensitivity measure [38], which matters here because raters were biased toward answering "Real." We obtained $d' = 0.15$ ($SE = 0.087$). To test for indistinguishability, we applied the Two One-Sided Tests (TOST) procedure [50, 29] with an equivalence margin of $\delta = \pm 0.3$. The margin is substantive: $d' = 0.3$ caps an unbiased rater at about 56% correct against 50% chance, and the observed $d' = 0.15$ corresponds to about 53%, matching the measured 52.8%.

- Test 1 ($d' > -0.3$): $z = 5.13, p < 0.001$
- Test 2 ($d' < 0.3$): $z = -1.78, p = 0.038$

The maximum p-value ($p = 0.038$) allows us to reject the null hypothesis of inequality, confirming that **BEAGLE** traces are perceptually equivalent to real student data within the defined margin. At the observed standard error (from 852 classifications), the TOST had approximately 90% power to declare a truly indistinguishable simulator equivalent.

H.3 Robustness and Sensitivity Analyses

A near-chance result could reflect an insensitive instrument rather than indistinguishable traces. The definitive check is a positive-control condition, which this study did not include (see Limitations below). We report five analyses of the existing data as supporting, not conclusive, evidence.

Responses are not random. Per-stimulus accuracy ranges from 32.4% to 76.1% across the 12 traces and orders coherently with the raters' own realism ratings (the simulated trace rated most realistic was also the most often misclassified), whereas random responding predicts flat accuracy. Mean

self-reported difficulty was 4.16 of 5. Accuracy does not improve with exposure within a session: 54.5% over each rater's first six trials and 51.2% over the last six, so no learnable cue emerges across trials.

No skilled subpopulation. Per-rater accuracy shows no meaningful overdispersion relative to a binomial model (variance ratio 1.11). Three of 71 raters reached individual significance (all above chance, none below), against approximately 2.7 expected under a pure-chance two-sided criterion; the best rater scored 10 of 12.

Viewing time. Submission timestamps give the time spent per trace: median 35.7 s (10th percentile 14.5 s). Excluding raters whose median viewing time falls below thresholds from 5 to 30 s leaves accuracy between 52.4% and 53.6% and d' between 0.12 and 0.19. This is a post hoc effort proxy rather than an attention check: long viewing times do not establish attentiveness.

Convergent Likert measure. The same raters scored behavioral realism 3.54 for real traces against 3.36 for **BEAGLE** traces, and code realism 3.49 against 3.29, consistent with the small d'.

Sample composition and expertise. Of the 71 raters, 35 reported formal CS training (graduate students, undergraduates, professionals), 10 reported less formal training, and 26 did not report a background. CS-trained raters scored 54.0% ($d' = 0.21$) against 50.8% ($d' = 0.04$) for less formally trained raters (two-proportion $p = 0.53$); raters who did not report a background scored 51.9%. A nonsignificant difference does not itself establish equivalence, but both groups' d' fall inside the ± 0.3 bound.

Limitations. The study used a convenience sample and included neither attention checks nor a positive-control condition in which baseline traces (e.g., Vanilla or CoT) are shown through the same pipeline. Without that condition we cannot demonstrate directly that raters can discriminate in this setup, so the result should be read as perceptual equivalence of rendered traces under our evaluation interface. Raters saw code progressions rather than monologue, so the Turing test does not validate the epistemic axis; that axis is anchored in the execution-level real-student references (Table 14).

H.4 Qualitative Findings and Discussion

The Turing test results offer valuable insights into what makes simulated programming behavior convincing. We collected responses from 71 participants across diverse backgrounds: graduate students (36%), undergraduates (32%), professionals (11%), hobbyists (9%), beginners (7%), and educators (5%). Participants rated the task as difficult (4.16/5).

The qualitative feedback reveals a fascinating paradox: participants developed reasonable heuristics that nonetheless failed. Several noted that "real students follow clear thinking patterns... while AI samples generate in no particular order." Yet this same participant achieved only 33% accuracy. Comments emerged as the most contested signal: one participant stated "the comments give it away," while another noted "hashtags when commenting" as an AI tell. In reality, both real students and **BEAGLE** produced similar commenting styles, leading expert raters to perform at chance levels.

The most insightful feedback came from participants who recognized the fundamental difficulty: "Is this code bad because the person is bad at coding or because the AI isn't consistent?" This captures the core challenge: **BEAGLE** produces code that is neither consistently "too good" nor "too bad," but instead occupies the ambiguous middle ground where real student work also resides.

Several specific patterns did successfully distinguish AI from real: "writing multi-line docstrings" (real students rarely document during timed tasks), "imported math before required" (real students import on first use), and "switching between methods back and forth" in ways that felt mechanical rather than goal-directed. Conversely, cues that correctly identified real students included "copy/pasting function lines then changing names" and "going back and catching the mistake."

The Likert ratings showed near-parity between AI (behavior: 3.36, code: 3.29) and Real (behavior: 3.54, code: 3.49) samples. Perhaps most tellingly, graduate students and professionals performed no better than beginners, despite having the most experience reading student code. Expertise offered no advantage in this task. As one participant ruefully concluded: "It's joever—I was no better than a coin flip."

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and §1 state four contributions (semi-Markov controller, BKT with explicit flaw injection, Strategist/Executor decoupling, and a comprehensive evaluation across four tasks, six LLM backbones, and nine baselines including a human Turing test); each is evaluated in §4 (RQ1–RQ3), and the abstract's claim of indistinguishability is supported by the TOST equivalence result ($d' = 0.15$, $N = 71$, $p_{\text{TOST}} = 0.038$; App. H).

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: §4.4 explicitly characterizes module-level trade-offs (e.g., removing BKT raises perceptual fluency but degrades Debug realism; merging Strategist/Executor reduces error recurrence by 21%); App. A.3 discusses interrupt modeling and its impact on help-seeking utility. §6 states the scope (computational STEM tasks with an executable oracle; transfer to subjective domains untested), the small real-student corpora, and the limits of the human evaluation (convenience sample, no positive control, 30-trace judge calibration).

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best

judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

   Justification: The paper is empirical; it does not contain formal theorems requiring proofs. The semi-Markov factorization in §3.2 is a generative-model specification, not a theoretical claim.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: §3 fully specifies the architecture; App. E details the evaluation protocol ($N = 50$ simulations, $T = 30$ steps, Gemini 2.0/2.5 Flash and other LLM backbones); App. B.1 reports BKT priors (taken from [14]); App. A.3 reports semi-Markov fitting; full prompts are in App. C. The semi-Markov is calibrated on the cited Snyder et al. [53] dataset.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.

(c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: Code, configurations, and prompts are publicly available at https://github.com/HCWDavid/BEAGLE-NIPS2026; the calibration dataset [53] is available upon request from its authors. Human-subject data collected under IRB protocols is not redistributed.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: **BEAGLE** is training-free (no LLM weight updates); Symbolic Control parameters are fit by maximum likelihood once on cited public student logs (App. A, App. A.3), with BKT slip/guess priors from [14] (App. B.1). Evaluation hyperparameters ($N = 50$, $T = 30$, $n_{\text{boot}} = 2000$, seed 42, backbone choices) are listed in §4 and App. E, E.4.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [Yes]

Justification: D_{KL} and D_{debug} are reported as point estimate $\pm$ trajectory-level non-parametric bootstrap SE ($n_{\text{boot}} = 2000$, App. E.4); remaining metrics report sample SD across $N = 50$ runs. LLM-as-judge is calibrated against expert raters (Cohen's $\kappa = 0.76$ on the real-versus-simulated classification, $\kappa_w = 0.73$ on Realism; App. F.3). Turing-test analyses report a binomial test, SDT d' with SE, and a TOST equivalence procedure ($\delta = \pm 0.3$, $p_{\text{TOST}} = 0.038$; App. H).

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

Answer: [Yes]

Justification: **BEAGLE** is training-free and inference-only: all neural compute is forwarded to frozen LLM APIs (Gemini 2.0/2.5 Flash for the headline results; six frontier LLM backbones for the cross-model study in App. F.2). Each condition runs $N = 50$ trajectories at $T = 30$ steps. Symbolic Control runs on a single CPU; total compute is dominated by API token costs, with no local GPU training required.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics https://neurips.cc/public/EthicsGuidelines?

Answer: [Yes]

Justification: All human-subjects components (real-student calibration data, the Turing-test rater study, and pilot study dataset) ran under IRB-approved protocols with informed consent (§4, App. A.4, App. H.1). Anonymity is preserved in submission, and no protected-class identifiers are used.

Guidelines:

- The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
- If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [Yes]

Justification: §1's motivation is itself a positive-impact framing: synthetic learner trajectories enable adaptive-tutor research and pedagogical stress-testing without exposing minors to unvetted interventions, and reduce the privacy risk of FERPA/GDPR-restricted data collection. Potential negative impacts (e.g., misuse of synthetic traces to bypass academic-integrity tooling, or to misrepresent student performance) are mitigated by the framework's reliance on calibration to cited authentic data and IRB-approved protocols rather than free-form generation.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

Answer: [N/A]

Justification: **BEAGLE** is a wrapper framework around frozen, externally hosted LLMs; we release no pre-trained model weights, scraped corpora, or datasets that pose dual-use risk beyond what the underlying LLMs already incur.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: All third-party assets are cited in the bibliography: LLM backbones (Gemini 2.0/2.5/3 Flash, Claude H4.5, GPT-4.1m/4om) are accessed via official APIs under their respective terms; calibration logs from Snyder et al. [53], exemplar traces from the DCU dataset [2], BKT priors from Corbett & Anderson [14], and baseline implementations (SimStudent, LLM-SS, CoderAgent) follow their original licenses.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, paperswithcode.com/datasets has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: The **BEAGLE** framework (Symbolic Control, Strategist/Executor pipeline, IDE Oracle, prompts) is documented across §3 and Apps. B, C, E; the public repository (https://github.com/HCWDavid/BEAGLE-NIPS2026) includes code, configurations, and prompts under the MIT license.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.

- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [Yes]

    Justification: The Turing-test protocol ($N = 71$ participants, 852 classifications, 12 traces each across 2 problem scenarios) is documented in App. H.1, including the evaluation interface (timeline-slider viewer with playback controls), the rating instruments (5-point Likert + binary classification + optional free text), and the post-task self-report items.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [Yes]

    Justification: All human-subjects components ran under IRB-approved protocols (#170887) with informed consent, as stated in §4 and App. A.4.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [Yes]

    Justification: LLMs are central to the methodology: the Strategist and Executor in §3.3 are zero-shot in-context calls to frozen LLMs (Gemini 2.0/2.5 Flash for headline results; six frontier backbones for the cross-model study in App. F.2), and an LLM-as-Judge calibrated against human raters ($\kappa = 0.76$, App. F.3) is used as one of the evaluation instruments.

    Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.