FROM LEARNER BEHAVIOR TO REUSABLE SKILLS FOR EFFECTIVE AND EFFICIENT LEARNER SIMULATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Learner simulation aims to reproduce how a particular learner behaves on new tasks. Although Large Language Models (LLMs) can generate increasingly fine-grained learning behaviors, existing approaches often need to repeatedly process a growing interaction history to reconstruct the learner. This introduces additional context and inference costs and makes the acquired learner-specific simulation capability difficult to reuse across different LLMs. We therefore propose **Learner2Skill**, which externalizes the simulation capability acquired from historical interactions into a persistent and reusable **Simulation Skill**. The Skill captures the learner's current learning state and recurring response patterns, evolves as new real interactions arrive, and can be adapted to a new LLM through lightweight executor calibration without reconstructing the learner from scratch. Experiments show that Learner2Skill more faithfully reproduces fine-grained learner behavior while reducing overall token cost, and that the same constructed Skills can be effectively reused across different LLM executors.

1 INTRODUCTION

Learner simulation aims to reproduce how a particular learner behaves when facing new learning tasks (Zhao et al., 2023; Scarlatos et al., 2026). A useful learner simulator should capture more than whether the learner ultimately answers correctly: it should also reflect whether the learner attempts the task, how the solution process unfolds, what errors or corrections appear along the way, and what answer is produced (Asano et al., 2025; Koutcheme et al., 2026). Such fine-grained simulation provides a low-cost, controllable, and repeatable environment for developing and evaluating personalized teaching strategies without repeatedly involving real learners (Markel et al., 2023; Lu & Wang, 2024). Recent advances in Large Language Models (LLMs) have made this increasingly feasible, as LLM-based agents can generate rich, multi-step responses conditioned on learner information (Xu et al., 2024; Liu et al., 2024).

Existing LLM-based learner simulators, however, typically build their understanding of a learner by repeatedly processing the learner's interaction history, as illustrated in Figure 1(a). As new interactions accumulate, learner profiles, memories, or learning states are progressively constructed and updated before being used for future simulation (Gao et al., 2025; Duan et al., 2026b; Xu et al., 2025). This process becomes increasingly expensive as the history grows. More importantly, the learner-specific simulation capability acquired from that history is usually coupled to the current agent and its underlying executor. The historical interactions themselves can be preserved, but when the LLM or executor changes, the same history may still need to be processed again to reconstruct an effective representation of the learner. In other words, existing approaches preserve the learner's experience, but provide no explicit mechanism for preserving and reusing the *simulation capability acquired from that experience*.

This observation raises a natural question: *if historical interactions have already revealed what a learner can currently do and how that learner typically responds, can this acquired capability be preserved and directly reused in future simulation?* We address this question with **Learner2Skill**, illustrated in Figure 1(b), which organizes learner-specific simulation capability into a persistent and reusable **Simulation Skill**. Rather than repeatedly reconstructing the learner from the complete history, Learner2Skill distills the information most relevant to future simulation into the Skill, including the learner's current **learning state** and recurring **response patterns**, such as typical

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

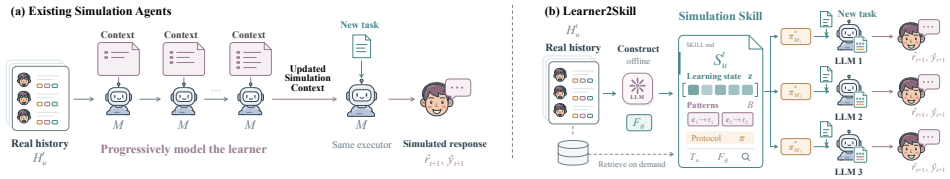

Figure 1: Comparison of (a) existing methods that progressively model learners from interaction histories and (b) Learner2Skill, which constructs reusable Simulation Skills for cross-executor simulation and online evolution.

solution strategies, characteristic errors, corrections, and stopping tendencies. The complete real interaction history is retained separately as external evidence and retrieved only when finer-grained information is needed. Future simulations can therefore begin from an already constructed Skill while preserving access to the underlying historical evidence.

Learner2Skill supports this reusable representation through a complete lifecycle. **Skill Construction** first extracts the learner's current learning state and repeatedly supported response patterns from real historical interactions, while keeping cognitive modeling and historical retrieval available as on-demand resources. Because different LLMs may interpret and use the same learner information differently, **Executor Calibration** then uses a small amount of real interaction feedback from multiple learners to adapt a shared execution protocol, without changing the constructed learner information itself. When the underlying LLM changes, the existing Skills can therefore be retained and only lightweight calibration needs to be repeated. During deployment, **Skill Evolution** further updates the learner's state and response patterns as new real interactions arrive, allowing the Skill to remain aligned with the learner over time. This design explicitly separates learner-specific simulation capability from the executor that uses it: the Skill preserves who the learner is and how they tend to behave, while the executor determines how that information is used for the current task.

We evaluate Learner2Skill on **LearnerTrace-2K**, a temporally ordered dataset containing 52,566 mathematics and physics interactions from 2,000 learners, including learners' main problem-solving ideas and intermediate steps. Following the interaction chronology, each learner's trajectory is used sequentially for Skill Construction, Executor Calibration, and final evaluation. Under the same LLM executor, Learner2Skill achieves the best performance on five of six fine-grained simulation metrics while reducing overall token cost. The same constructed Skills can also be directly reused across different LLM executors without repeating Skill Construction. Ablation studies further show that learning state, response patterns, executor calibration, and online evolution contribute complementary benefits to different aspects of simulation. Finally, learner responses generated by Learner2Skill improve several computerized adaptive testing methods when used as additional training data. These results suggest that learner-specific simulation capability can be externalized from a particular LLM executor, preserved as a reusable Skill, and carried forward across subsequent learner simulation.

2 Related Work

LLM-Based Learner Simulation Early LLM-based learner simulation mainly used predefined personas, knowledge levels, or learner attributes to generate heterogeneous student behaviors for applications such as teacher training, item evaluation, and intelligent tutoring (Markel et al., 2023; Lu & Wang, 2024; Benedetto et al., 2024; Liu et al., 2024; Nguyen et al., 2025; Jin et al., 2025). More recent studies construct learner agents from authentic learning data (Xu et al., 2024; Gao et al., 2025), covering classroom-level simulation (Xu et al., 2025; Zhang et al., 2025; Gao et al., 2026), tutoring dialogues (Duan et al., 2026b; Scarlatos et al., 2026; Zheng et al., 2025), and domain-specific settings such as programming (Zhan et al., 2025; Duan et al., 2026a; Koutcheme et al., 2026). Most closely related to our work is *individual learner response simulation*. Agent4Edu and Embracing Imperfection characterize individual learners from historical records and simulate their subsequent responses (Gao et al., 2025; Wu et al., 2025), while SMART and One LLM Does Not Simulate All Students study the alignment between simulated behavior and learner ability (Scarlatos et al., 2025; Que et al., 2026). Empirical studies further show that LLM-generated solutions can differ systematically from real learner behavior even when final answers are correct (Asano et al., 2025). Existing learner agents typically acquire learner-specific simulation capability through progressive processing of interaction histories, limiting both efficiency and reuse across executors. Learner2Skill instead preserves this capability as a persistent and reusable Simulation Skill.

Skills for LLM Agents Recent LLM-agent research increasingly studies how experience can be externalized into reusable knowledge, workflows, or skills. ExpeL, Agent Workflow Memory, and Voyager extract reusable guidance or executable procedures from previous trajectories (Zhao et al., 2024; Wang et al., 2024; 2023b), while SkillX, SAGE, SkillRL, and Skill1 further investigate construction, selection, transfer, and evolution of reusable skill libraries (Wang et al., 2026b;a; Xia et al., 2026; Shi et al., 2026). Related work also explores improving externally represented skills from execution experience and transferring them across executors or environments (Liu et al., 2026), as well as progressively internalizing external skills into model parameters (Lu et al., 2026). These developments establish reusable skills as an emerging abstraction for preserving capabilities acquired from experience. We apply this abstraction to *learner-specific simulation capability*: rather than preserving how an agent solves a task, a Simulation Skill preserves how a particular learner tends to respond, allowing the acquired capability to be maintained and reused across LLM executors.

3 The Proposed Learner2Skill Method

We focus on **individual learner response simulation**: given the previously observed real interactions of a particular learner, the goal is to simulate how the same learner would respond to a new learning task (Duan et al., 2026b; Scarlatos et al., 2026). For learner u, we denote the real interaction history observed up to time step t as

$$\mathcal{H}_u^t = \{(e_i, r_i, y_i)\}_{i=1}^t, \tag{1}$$

where e_i denotes a learning task, such as a mathematics problem, r_i denotes the learner's complete response behavior on that task, and y_i denotes the corresponding observable outcome. Specifically, we consider four simulation targets: (1) whether the learner attempts the task; (2) the response process, such as the solution strategy used, intermediate errors, corrections, or verification; (3) the final answer; and (4) the resulting performance, such as correctness or task score. Given a new task e_{t+1}, the objective is therefore to generate the corresponding learner response and outcome $(\hat{r}_{t+1}, \hat{y}_{t+1})$. Throughout simulation, only real interactions observed before the current task are available.

Existing LLM-based learner agents typically process a learner's interaction history step by step to construct and update learner-specific simulation context (Gao et al., 2025; Duan et al., 2026b; Xu et al., 2025). As the trajectory grows, this requires continued processing of historical interactions, while the acquired learner-specific simulation capability is typically coupled to the current agent and its executor, making it difficult to directly reuse across different executors.

In contrast, the proposed Learner2Skill explicitly organizes the learner-specific simulation capability acquired from historical interactions into a persistent and reusable **Simulation Skill**:

$$\mathcal{S}_u^t = \text{Skill}\left(\mathcal{H}_u^t\right). \tag{2}$$

Given a new task e_{t+1}, an executor M uses the existing Skill to generate

$$(\hat{r}_{t+1}, \hat{y}_{t+1}) \sim q_{M,\pi}\left(r, y \mid e_{t+1}, \mathcal{S}_u^t\right), \tag{3}$$

where π denotes the execution protocol through which the executor uses the Skill for learner simulation. A formal comparison between the conventional learner-agent pipeline and the Learner2Skill pipeline is provided in Appendix A.

This formulation leads to three questions: *How can a reusable Simulation Skill be constructed from historical learner interactions? How can an existing Skill be adapted to different downstream executors? And how can the Skill remain aligned with the learner as new real interactions arrive?* Learner2Skill addresses these questions through **Skill Construction**, **Executor Calibration**, and **Skill Evolution**, respectively, as illustrated in Figure 2.

3.1 Constructing a Reusable Simulation Skill

Skill Construction is performed offline before downstream learner simulation, independently of the downstream LLM executor that will later use the Skill for simulation. Given the observed real interaction history $\mathcal{H}_u^t$ of learner u, our goal is to extract learner-specific information that is useful for future response simulation and organize it into a reusable Simulation Skill. Specifically, we

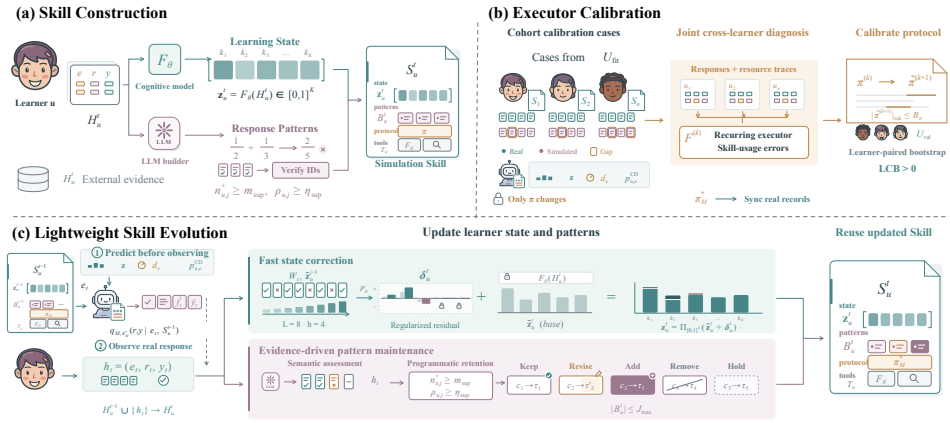

Figure 2: Overview of Learner2Skill. (a) Construct learner-specific Skills from real histories. (b) Calibrate a shared execution protocol through cross-learner diagnosis and held-out validation. (c) Update learner states and response patterns from new real interactions, with model parameters and the calibrated protocol fixed.

characterize two complementary aspects: **what the learner can currently do, represented by the learning state, and how the learner typically responds, represented by response patterns**.

Learning tasks typically involve one or more knowledge concepts, such as quadratic equations or function transformations in a mathematics problem (Wang et al., 2023a). We therefore represent the learner's **learning state** through knowledge-concept mastery. Because such mastery is latent and cannot be directly observed, we use a pretrained cognitive model F_θ to estimate the learner's state from the real interactions observed up to time t (Pandey & Karypis, 2019; Liu et al., 2019):

$$\mathbf{z}_u^t = F_\theta(\mathcal{H}_u^t) \in [0,1]^K, \tag{4}$$

where K denotes the number of knowledge concepts, and the k-th dimension of $\mathbf{z}_u^t$ represents the learner's estimated mastery of concept k.

The cognitive model is also retained as a **tool** associated with the Skill for subsequent simulation. For the current task, the tool provides the relevant dimensions of the learner's learning state, the task difficulty, and the predicted probability that the learner will correctly complete the task (Lord, 2012; Scarlatos et al., 2025). When additional cognitive information is needed, the executor can further query the cognitive model on demand. The construction, training, and use of the cognitive model are detailed in Appendix D.

The learning state captures the learner's current knowledge mastery, but similar learning states do not necessarily imply similar response behavior. Two learners with comparable knowledge states may still adopt different solution strategies, repeatedly make different types of errors, or exhibit different correction, verification, and stopping tendencies (Liu et al., 2024; Wu et al., 2025; Asano et al., 2025). We therefore use an offline LLM as the Skill builder to extract the learner's **response patterns** from the real interaction history (Duan et al., 2026b):

$$\mathcal{B}_u^t = \{(c_{u,j}, \tau_{u,j})\}_{j=1}^{J_u^t}, \tag{5}$$

where $c_{u,j}$ specifies the condition under which a pattern occurs, such as the relevant knowledge concepts, task difficulty, or other observable task characteristics, and $\tau_{u,j}$ describes the learner's typical response behavior under that condition. Such behavior may include using a particular solution strategy, making a characteristic error at a certain step, completing only part of a solution, checking or correcting a response, or stopping before completion.

The Skill builder used to construct these response patterns is separate from the downstream LLM executor that later performs learner simulation. A behavior is not retained as a response pattern based on a single occurrence; it enters $\mathcal{B}_u^t$ only when it is repeatedly supported by multiple real interactions, preventing isolated behavior from being treated as a persistent learner characteristic. The extraction, consolidation, and retention of response patterns are detailed in Appendix B.2.

Because the learning state and response patterns summarize historical behavior, they may not capture every detail needed for a new task. We therefore retain the complete real interaction history $\mathcal{H}_u^t$ as

external evidence in an interaction database and access it through retrieval rather than loading the full trajectory into every simulation context. When needed, the executor can retrieve the learner's most recent interactions, past interactions involving the current knowledge concepts, past interactions related to the current task content, and anonymized responses from other learners to the same task. The retrieval mechanism is detailed in Appendix B.5.

Based on the above information, we represent the Simulation Skill of learner u at time t as

$$\mathcal{S}_u^t = \left(\texttt{SKILL.md}, \mathbf{z}_u^t, \mathcal{B}_u^t, \mathcal{T}_u\right), \tag{6}$$

where $\mathbf{z}_u^t$ and $\mathcal{B}_u^t$ denote the learner's current learning state and response patterns, respectively, and $\mathcal{T}_u$ denotes the tool interfaces associated with the Skill, including the cognitive model F_θ and the retrieval tool for accessing the interaction database. The markdown file `SKILL.md` serves as the unified entry point and operational specification of the Skill. It describes how the learner-specific information and associated tools are organized and contains a shared execution protocol π that specifies how a downstream executor uses them during simulation. All learner Skills initially share the same base execution protocol, which is subsequently adapted to the current executor through Executor Calibration in Section 3.2. The organization of the Simulation Skill and the `SKILL.md` template are detailed in Appendix B.3.

To keep the Skill compact as the interaction history grows, Learner2Skill maintains only the learner's **current learning state** and currently supported response patterns rather than their historical versions. The complete real interaction trajectory remains in the external database, while the number of maintained response patterns is bounded by a fixed pattern budget, preventing the Skill size from growing linearly with the interaction history.

3.2 Calibrating a Skill to an Executor

After Skill Construction, the learner information and associated tools required for subsequent simulation have already been organized in the Simulation Skill. When a downstream LLM executor is equipped with a learner's Skill, it first reads the execution protocol in `SKILL.md` and, given the current task and available learner information, determines how to use the learner information already provided for the current task and whether additional resources, such as further cognitive-model queries or retrieval from the interaction database, should be invoked. Different LLMs or agent executors may vary in how they interpret the learning state and response patterns, when they invoke the associated tools, how they combine the available evidence, and how faithfully they reflect that evidence in the generated response. We therefore use a small amount of real interaction feedback from multiple learners to calibrate the shared execution protocol π in `SKILL.md` to the current executor M, while keeping the executor model parameters fixed. Specifically, we collect a small number of real interactions from multiple learners as the calibration set:

$$\mathcal{D}^{\text{cal}} = \{(u_i, e_i, r_i, y_i)\}_{i=1}^{N_c} \tag{7}$$

where each interaction contains a task together with the corresponding real learner response and outcome. These interactions are observed after Skill Construction and before final testing. Rather than learning a separate usage strategy for each learner, calibration aggregates feedback across learners to identify recurring errors made by the current executor when using the Skill. Because these errors reflect shared problems in how the executor follows the common execution protocol, only a small number of real interactions from each learner is needed to refine the protocol, reducing the cost of adapting existing learner Skills to a new executor.

The learners participating in calibration are divided into a protocol-fitting set $\mathcal{U}_{\text{fit}}$ and an independent validation set $\mathcal{U}_{\text{val}}$, with no learner shared between them. The former is used to identify Skill-usage errors that recur across learners and to propose protocol edits, whereas the latter is used only to determine whether a candidate protocol yields consistent improvements on learners not involved in protocol editing. The calibration data and learner split are detailed in Appendix E.1.

At calibration round k, the fixed executor M uses the constructed Skills of learners in $\mathcal{U}_{\text{fit}}$ together with the current execution protocol $\pi^{(k)}$ to simulate the corresponding tasks through the simulation interface described in Appendix B.6. Following the current protocol, the executor uses the learner information and current-task information already provided and, when needed, further queries the cognitive model or retrieves relevant historical evidence from the interaction database. The real

learner response and outcome are revealed only after simulation and are used as feedback. By comparing the simulated response with the real response and examining how the executor uses the information and tools already available through the Skill, we identify recurring **Skill-usage errors** in how it interprets, requests, combines, or reflects learner evidence.

Because the execution protocol is shared across all learner Skills, we update it only from Skill-usage errors that recur across different learners rather than from isolated errors in individual interactions. When calibration interactions are processed in multiple mixed-learner batches, similar errors identified across batches are first consolidated before protocol editing; details are provided in Appendix E.2. Let $\mathcal{F}^{(k)}$ denote the cross-learner Skill-usage errors identified at round k. A candidate protocol is obtained by

$$\tilde{\pi}^{(k+1)} = \mathrm{Edit}\left(\pi^{(k)}, \mathcal{F}^{(k)}\right), \qquad |\tilde{\pi}^{(k+1)}|_{\mathrm{tok}} \le B_\pi \tag{8}$$

where B_π constrains the protocol length so that the protocol remains compact across calibration rounds. The protocol-editing procedure is detailed in Appendix E.3.

Each candidate protocol is then compared with the current protocol using calibration interactions from $\mathcal{U}_{\mathrm{val}}$. Validation computes paired improvements at the learner level and further applies a one-sided paired bootstrap to assess whether the improvement is consistent. A candidate is accepted only when the lower confidence bound of the paired improvement is greater than zero. Calibration terminates when no further candidate is accepted or the maximum number of calibration rounds R_{cal} is reached, yielding an executor-specific shared execution protocol $\pi^\star_M$. All learner Skills under executor M then use this calibrated protocol. The validation procedure is detailed in Appendix E.4.

During calibration, the learning state, response patterns, and accessible real historical evidence in the Skill remain unchanged; only the execution protocol is updated. After the final $\pi^\star_M$ is selected, the real interactions observed during calibration are incorporated into the corresponding learners' real interaction histories in chronological order and used to synchronize their Skills, so that the learning state and response patterns available at the beginning of final testing are consistent with the latest observed real interactions. The calibrated protocol $\pi^\star_M$ remains fixed during this synchronization. These calibration interactions are excluded from the final evaluation, and subsequent updates from newly observed real interactions follow the Skill Evolution procedure in Section 3.3. Details are provided in Appendix E.5.

When the downstream LLM or agent executor changes from M to a new executor M', the constructed learner Skills and the real interaction histories stored in the database are retained. The same low-cost calibration procedure is then applied to obtain a new execution protocol $\pi^\star_{M'}$ for the new executor. In this way, the learner-specific simulation capability extracted and organized in Stage 1 can be reused across executors without reconstructing the learner Skill from the complete interaction history (Liu et al., 2026; Wang et al., 2026a). The theoretical analysis in Appendix G establishes when cross-executor Skill reuse gives Learner2Skill a lower total token cost than an existing LLM-based learner-simulation agents.

3.3 Test-Time Skill Evolution with Real Interactions

After Skill Construction and Executor Calibration, Learner2Skill proceeds to online simulation. As new real interactions arrive, the learner's knowledge state and response behavior may continue to change, so the existing Skill needs to incorporate this new evidence over time. We therefore introduce lightweight **test-time Skill Evolution**, which progressively updates the learner's learning state and response patterns from newly observed real interactions (Xia et al., 2026).

For learner u at time step t, simulation uses only the Skill available before the current real interaction:

$$(\hat{r}_t, \hat{y}_t) \sim q_{M,\pi^\star_M}\left(r, y \mid e_t, \mathcal{S}_u^{t-1}\right). \tag{9}$$

After simulation, the real response r_t and outcome y_t are observed and appended to the learner's real interaction history $\mathcal{H}_u^t$. Skill Evolution uses only real interactions as update evidence; simulated responses are never treated as learner observations.

For the **learning state**, the frozen cognitive model F_θ first produces a new base estimate $\tilde{\mathbf{z}}_u^t$ from the updated real interaction history. To make the state responsive to recent learner behavior, we further estimate a learner-specific residual correction $\boldsymbol{\delta}_u^t$ from the most recent L real interactions, yielding

$$\mathbf{z}_u^t = \Pi_{[0,1]^K}\left(\tilde{\mathbf{z}}_u^t + \boldsymbol{\delta}_u^t\right). \tag{10}$$

Table 1: Main results on LearnerTrace-2K. Effectiveness results are reported as mean ± standard deviation over three runs. All LLM-based methods use **Claude Sonnet 4.6** as the simulation executor. Higher is better for simulation effectiveness and cost-effectiveness, while lower is better for simulation cost. "–" indicates that the metric is not applicable. The best result is shown in **bold** and the second-best result is underlined.

	Simulation Effectiveness (↑)						Simulation Cost (↓)			Cost-Effectiveness (↑)
Method	**Attempt**	**Process**	**Final Ans.**	**Realized Corr.**	**Outcome Acc.**	**Consistency**	**Acq.**	**Test**	**Total**	**CE**
Supervised Learner-Performance Prediction										
KES	–	–	–	–	$52.92_{\pm 0.10}$	–	–	–	–	–
DKVMN	–	–	–	–	$61.16_{\pm 0.10}$	–	–	–	–	–
SAKT	–	–	–	–	$61.01_{\pm 0.09}$	–	–	–	–	–
DAISim	–	–	–	–	$62.63_{\pm 0.05}$	–	–	–	–	–
LLM-Based Learner Simulation										
LLM-FullHistory	$\underline{99.22}_{\pm 0.07}$	$50.66_{\pm 0.14}$	$29.00_{\pm 0.08}$	$46.88_{\pm 0.06}$	$61.12_{\pm 0.13}$	$60.48_{\pm 0.09}$	$\underline{227.170M}$	125.585M	$\underline{352.755M}$	$\underline{0.00171}$
EduAgent	$99.15_{\pm 0.09}$	$52.16_{\pm 0.11}$	$29.66_{\pm 0.09}$	$47.33_{\pm 0.12}$	$\underline{62.91}_{\pm 0.10}$	$\underline{66.40}_{\pm 0.11}$	312.730M	133.770M	446.500M	0.00149
Agent4Edu	$98.13_{\pm 0.07}$	$32.33_{\pm 0.13}$	$37.10_{\pm 0.10}$	$\mathbf{58.61}_{\pm 0.15}$	$57.26_{\pm 0.09}$	$55.26_{\pm 0.06}$	414.550M	118.455M	533.005M	0.00104
CoderAgent	$99.10_{\pm 0.13}$	$\underline{53.77}_{\pm 0.13}$	$33.24_{\pm 0.14}$	$52.37_{\pm 0.15}$	$59.25_{\pm 0.09}$	$57.30_{\pm 0.09}$	351.234M	121.007M	472.241M	0.00121
AAS	$98.23_{\pm 0.08}$	$53.70_{\pm 0.13}$	$\underline{38.24}_{\pm 0.13}$	$55.23_{\pm 0.12}$	$60.12_{\pm 0.07}$	$58.31_{\pm 0.10}$	403.277M	$\underline{117.532M}$	520.809M	0.00112
Learner2Skill	$\mathbf{99.64}_{\pm 0.06}$	$\mathbf{54.19}_{\pm 0.10}$	$\mathbf{40.13}_{\pm 0.13}$	$\underline{57.71}_{\pm 0.07}$	$\mathbf{63.43}_{\pm 0.09}$	$\mathbf{71.54}_{\pm 0.11}$	**182.590M**	**116.570M**	**299.160M**	**0.00239**

The residual correction is fitted to recent real performance while being regularized in magnitude and temporal variation, allowing the learner state to adapt to recent evidence without updating the cognitive-model parameters. The recent-interaction objective and optimization procedure are detailed in Appendix F. For **response patterns**, we adopt an evidence-driven adaptive update mechanism. Each new real interaction is compared with the existing patterns to determine whether it provides additional support for an existing pattern, conflicts with one, or provides evidence for a new candidate pattern. Existing patterns are retained, revised, or removed according to accumulated evidence, while a new pattern is added only when it receives repeated support from multiple real interactions and satisfies the same retention criterion used during Skill Construction.

After each update, the latest $\mathbf{z}_u^t$ and $\mathcal{B}_u^t$ are stored in the Simulation Skill and used for subsequent learner simulation, while the complete real interaction history remains available in the external database for retrieval. Throughout Skill Evolution, the executor M, the calibrated execution protocol $\pi_M^\star$, and the underlying model parameters remain fixed.

4 Experiments

Dataset. Existing educational datasets primarily record learner outcomes or final answers, while large-scale data with detailed problem-solving processes remain scarce. We therefore collaborate with a real educational platform for high-school students in China to construct **LearnerTrace-2K**, containing temporally ordered mathematics and physics interactions with learners' main problem-solving ideas and intermediate steps. It contains **2,000 learners, 52,566 learner–exercise interactions, 2,076 exercises, and 709 knowledge concepts**. For each learner, we preserve the original interaction order and chronologically split the trajectory into 60%/10%/30% for Skill Construction, Calibration, and Evolution, respectively. Dataset details are provided in Appendix H.1.

Baselines. We compare Learner2Skill with five LLM-based simulators: LLM-FullHistory, EduAgent (Xu et al., 2024), Agent4Edu (Gao et al., 2025), CoderAgent (Zhan et al., 2025), and AAS (Que et al., 2026); and four supervised predictors: KES (Liu et al., 2019), DKVMN (Zhang et al., 2017), SAKT (Pandey & Karypis, 2019), and DAISim (Zhao et al., 2023). All LLM-based methods use the same executor and output format. Details are provided in Appendix H.2.

Implementation Details. Executor Calibration uses the 10% calibration split, with learners divided 70%/30% for protocol editing and validation. The cognitive model is trained only on the Skill Construction split and then fixed. RQ1 uses Claude Sonnet 4.6 for all LLM-based methods. RQ2 reuses the same constructed Skills across Claude Sonnet 4.6, Gemini 3.8 Flash, Qwen3.8 Flash, and DeepSeek V4 Flash, repeating only Executor Calibration. Semantic evaluation uses an independent GPT-5.5, and all experiments are repeated three times. Full settings are provided in Appendix H.3. Code is available at `https://anonymous.4open.science/r/Learner2Skill/`.

Evaluation. We evaluate **effectiveness** and **efficiency**. For effectiveness, **Attempt**, **Process**, **Final-Answer**, and **Realized Correctness Fidelity** measure whether the simulation reproduces the learner's attempt behavior, problem-solving process, final answer, and realized success or failure, respectively.

Table 2: Cross-executor reuse of Simulation Skills on LearnerTrace-2K. **Initial** denotes the complete pipeline used to construct the Skills, whereas **Reuse** directly applies the constructed Skills to a new executor without repeating Skill Construction.

Configuration		Simulation Fidelity (↑)						Simulation Cost (↓)			
Setting	**Executor**	**Attempt**	**Process**	**Final Ans.**	**Realized Corr.**	**Outcome Acc.**	**Consistency**	**Build**	**Calib.**	**Test**	**Total**
Initial	Claude Sonnet 4.6	99.64	54.19	40.13	**57.71**	63.43	71.54	125.36M	57.23M	116.57M	299.16M
Reuse	Gemini 3.8 Flash	99.78	**54.53**	**43.74**	57.22	**63.73**	75.21	0	60.55M	132.13M	192.68M
Reuse	Qwen3.8 Flash	98.27	50.15	40.07	55.24	59.77	72.21	0	**53.21M**	103.12M	156.33M
Reuse	DeepSeek V4 Flash	**99.94**	48.93	43.11	57.42	60.86	**75.22**	0	60.17M	**80.60M**	**140.77M**

Outcome Prediction Accuracy evaluates the explicit prediction of learner performance, while **Prediction–Response Consistency** measures whether this prediction agrees with the outcome realized by the generated response. For efficiency, we report **Capability Acquisition Cost**, **Test-Time Cost**, their **Total Cost**, and **Consistency Efficiency (CE)**, defined as Prediction–Response Consistency per million generated tokens. Full contents are provided in Appendix H.4.

4.1 RQ1: Can We Simulate Learners More Effectively and Efficiently?

We first compare Learner2Skill with supervised learner-performance predictors and LLM simulators on LearnerTrace-2K, examining both fine-grained simulation effectiveness and generative cost. For a fair comparison, all LLM-based methods use the same **Claude Sonnet 4.6** simulation executor and are evaluated on the same test interactions.

Simulation Effectiveness. As shown in Table 1, the base LLM already exhibits meaningful learner-simulation capability: LLM-FullHistory, which directly uses the complete learner history, reaches 50.66 in Process Fidelity and 61.12 in Outcome Prediction Accuracy, but remains limited in Final-Answer Fidelity (29.00) and Prediction–Response Consistency (60.48). While different learner-modeling methods improve particular behavioral dimensions, Learner2Skill provides more consistent gains, achieving the best results in Attempt, Process, Final-Answer, Outcome Prediction, and Prediction–Response Consistency. Compared with LLM-FullHistory, it improves Final-Answer Fidelity from 29.00 to 40.13 and Prediction–Response Consistency from 60.48 to 71.54; its Outcome Prediction Accuracy (63.43) also exceeds the strongest supervised predictor, DAISim (62.63). These results show that the Simulation Skill preserves strong outcome prediction while more faithfully reproducing the learner's problem-solving process, final answer, and the consistency between predicted and generated behavior.

Simulation Cost and Cost-Effectiveness. Learner2Skill improves simulation effectiveness while also reducing generative cost, achieving the lowest Capability Acquisition Cost (182.590M), Test-Time Cost (116.570M), and Total Cost (299.160M) among all LLM-based methods. Its total cost is approximately **15.2%** lower than the next-lowest method, LLM-FullHistory (352.755M). This efficiency gain reflects the reusable nature of the Simulation Skill: although LLM-FullHistory repeatedly processes the complete learner history during simulation, Learner2Skill acquires the learner-specific simulation capability once and subsequently reuses the resulting Skill across interactions. Combining effectiveness and efficiency, Learner2Skill achieves the highest CE of **0.00239**, approximately **39.8%** above the second-best result. Together, these results show that organizing learner-specific simulation capability as a reusable Simulation Skill improves fine-grained behavioral simulation while reducing the cost of acquiring and repeatedly using that capability.

4.2 RQ2: Can Simulation Skills Be Reused Across Different LLM Executors?

We next examine whether a Simulation Skill constructed once can be reused across different LLM executors. We first run the complete pipeline with **Claude Sonnet 4.6** to construct the initial Simulation Skills, and then directly reuse these Skills with **Gemini 3.8 Flash**, **Qwen3.8 Flash**, and **DeepSeek V4 Flash**. For each reuse setting, Skill Construction is not repeated; only executor-specific calibration is performed before final testing.

As shown in Table 2, the same Simulation Skills remain effective across all tested executors. Gemini 3.8 Flash achieves the highest Process Fidelity (54.53), Final-Answer Fidelity (43.74), and Outcome Prediction Accuracy (63.73), while DeepSeek V4 Flash achieves the highest Prediction–Response

Table 3: Ablation study of Learner2Skill on LearnerTrace-2K. Colored numbers indicate changes relative to the full model.

Method	Attempt ↑	Process ↑	Final Ans. ↑	Realized Corr. ↑	Outcome Acc. ↑	Consistency ↑
Full Learner2Skill	**99.64**	**54.19**	**40.13**	**57.71**	**63.43**	**71.54**
w/o Learning State	99.13 -0.51	56.80 +2.61	40.03 -0.10	53.80 -3.91	62.03 -1.40	70.51 -1.03
w/o Response Patterns	99.98 +0.34	35.05 -19.14	39.64 -0.49	58.50 +0.79	62.60 -0.83	70.18 -1.36
w/o Executor Calibration	99.63 -0.01	53.46 -0.73	39.75 -0.38	57.21 -0.50	62.57 -0.86	71.40 -0.14
w/o Skill Evolution	99.53 -0.11	52.83 -1.36	39.38 -0.75	56.86 -0.85	62.51 -0.92	71.37 -0.17

Consistency (75.22). Importantly, changing the executor does not require repeating the 125.36M-token Skill Construction stage. Instead, the previously constructed Skills are retained and only executor-specific calibration is performed. The resulting total costs are 192.68M, 156.33M, and 140.77M tokens for Gemini, Qwen, and DeepSeek, respectively, compared with 299.16M tokens for the initial pipeline. These results show that learner-specific simulation capability can be retained across different executors, allowing a new LLM to reuse the existing Simulation Skill rather than reconstructing it from the learner's history.

4.3 RQ3: What Makes Learner2Skill Effective?

We ablate the four key components of Learner2Skill: Learning State and Response Patterns in Skill Construction, Executor Calibration, and Skill Evolution. As shown in Table 3, Response Patterns are particularly important for Process Fidelity, while Learning State mainly improves outcome-related metrics. This is consistent with their roles: Response Patterns characterize how a learner typically behaves during problem solving, whereas Learning State reflects the learner's current capability. Executor Calibration and Skill Evolution provide further consistent gains by improving Skill use and keeping learner-specific information aligned with newly observed behavior. Together, these results confirm the complementary roles of the four components.

4.4 RQ4: Can Learner2Skill Benefit Real Educational Applications?

We evaluate whether learner responses generated by Learner2Skill can improve Computerized Adaptive Testing (CAT), a practical application that estimates learner proficiency using a limited number of adaptively selected exercises. We consider three CAT strategies: FSI (Lord, 2012), KLI (Chang & Ying, 1996), and MAAT (Bi et al., 2020), with test lengths of 5 and 10 exercises. For each strategy, we compare training on the original real learner data with training on the same data augmented by simulated learner responses. The CAT strategy remains unchanged, and both settings are evaluated on the same held-out real learners. Figure 3 shows higher F1 scores with simulated-data augmentation across all three strategies and both test lengths, with gains ranging from 0.48 to 2.88. These results suggest that Learner2Skill-generated responses provide useful additional supervision for CAT, improving performance on real learners without changing the adaptive testing strategy.

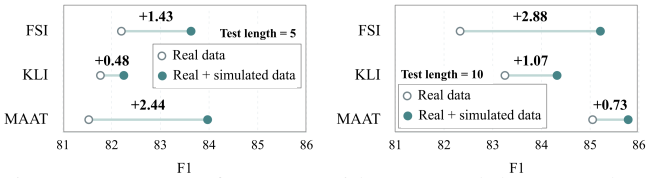

Figure 3: CAT performance with generated data. Numbers above the lines denote F1 improvements after augmentation.

5 Conclusion

We introduced **Learner2Skill**, which organizes learner-specific simulation capability acquired from historical interactions into persistent and reusable **Simulation Skills**, avoiding repeated reconstruction of the learner in subsequent simulation. Through Skill Construction, Executor Calibration, and Skill Evolution, Learner2Skill supports the construction, cross-executor reuse, and continual updating of learner-specific information. Experiments show that it more faithfully reproduces fine-grained learner behavior while reducing overall token cost, and that constructed Skills can be directly reused across different LLM executors. The generated learner responses also provide useful supervision for practical educational applications. Overall, Learner2Skill provides a way to decouple learner-specific simulation capability from a particular LLM and preserve it for continued reuse.

AI USE STATEMENT

Large language models were used in both the experiments and language checking of this work. In the experiments, we accessed large language models through model APIs to construct and run LLM-based learner simulation agents, including Simulation Skill construction, learner response simulation with different LLM executors, and related LLM-based baseline experiments. For several process-level simulation quality metrics, we additionally used an independent large language model as an automatic evaluator. The models used, their respective roles, and the corresponding experimental settings are specified in the paper. ChatGPT with GPT-5.6 Sol was also used to assist with checking English grammar, wording, and language fluency. The resulting edits were manually reviewed by the authors.

REFERENCES

Yuya Asano, Diane Litman, and Erin Walker. Can llms simulate the same correct solutions to free-response math problems as real students? In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 16347–16376, 2025.

Luca Benedetto, Giovanni Aradelli, Antonia Donvito, Alberto Lucchetti, Andrea Cappelli, and Paula Buttery. Using llms to simulate students' responses to exam questions. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 11351–11368, 2024.

Haoyang Bi, Haiping Ma, Zhenya Huang, Yu Yin, Qi Liu, Enhong Chen, Yu Su, and Shijin Wang. Quality meets diversity: A model-agnostic framework for computerized adaptive testing. In *2020 IEEE International Conference on Data Mining (ICDM)*, pp. 42–51. IEEE, 2020.

Hua-Hua Chang and Zhiliang Ying. A global information approach to computerized adaptive testing. *Applied Psychological Measurement*, 20(3):213–229, 1996.

Zhangqi Duan, Nigel Fernandez, and Andrew Lan. Kaser: Knowledge-aligned student error simulator for open-ended coding tasks. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 39988–40006, 2026a.

Zhangqi Duan, Shuyan Huang, Alexander Scarlatos, Jaewook Lee, Simon Woodhead, and Andrew Lan. Who am i? history-aware profiles for student simulation in tutoring dialogues. *arXiv preprint arXiv:2605.30051*, 2026b.

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 *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 23923–23932, 2025.

Weibo Gao, Qi Liu, Linan Yue, Zheng Zhang, Yichao Du, Fangzhou Yao, Ao Yu, Zhenya Huang, and Shijin Wang. Edu-theater: A data-efficient agent framework for scalable learner behavior simulation through staging roll-call. *arXiv preprint arXiv:2606.15225*, 2026.

Hyoungwook Jin, Minju Yoo, Jeongeon Park, Yokyung Lee, Xu Wang, and Juho Kim. Teachtune: Reviewing pedagogical agents against diverse student profiles with simulated students. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*, pp. 1–28, 2025.

Charles Koutcheme, Juho Leinonen, and Arto Hellas. Teaching language models how to code like learners: Conversational serialization for student simulation. *arXiv preprint arXiv:2604.10720*, 2026.

Qi Liu, Shiwei Tong, Chuanren Liu, Hongke Zhao, Enhong Chen, Haiping Ma, and Shijin Wang. Exploiting cognitive structure for adaptive learning. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 627–635, 2019.

Yuxuan Liu, Zhaochen Su, Lingyun Xie, Yuhao Zhang, Qing Zong, Jiahe Guo, Zhongwei Xie, Yiyan Ji, Yauwai Yim, Hongyu Luo, et al. Skillrevise: Improving llm-authored agent skills via trace-conditioned skill revision. *arXiv preprint arXiv:2606.01139*, 2026.

Zhengyuan Liu, Stella Xin Yin, Geyu Lin, and Nancy Chen. Personality-aware student simulation for conversational intelligent tutoring systems. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 626–642, 2024.

Frederic M Lord. *Applications of item response theory to practical testing problems*. Routledge, 2012.

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*, pp. 16–27, 2024.

Zhengxi Lu, Zhiyuan Yao, Jinyang Wu, Chengcheng Han, Qi Gu, Xunliang Cai, Weiming Lu, Jun Xiao, Yueting Zhuang, and Yongliang Shen. Skill0: In-context agentic reinforcement learning for skill internalization. *arXiv preprint arXiv:2604.02268*, 2026.

Julia M Markel, Steven G Opferman, James A Landay, and Chris Piech. Gpteach: Interactive ta training with gpt-based students. In *Proceedings of the tenth acm conference on learning@ scale*, pp. 226–236, 2023.

Bang Nguyen, Tingting Du, Mengxia Yu, Lawrence Angrave, and Meng Jiang. Qg-sms: enhancing test item analysis via student modeling and simulation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 26152–26168, 2025.

Shalini Pandey and George Karypis. A self-attentive model for knowledge tracing. *arXiv preprint arXiv:1907.06837*, 2019.

Huixing Que, Qi Liu, Weibo Gao, and Zhenya Huang. One llm does not simulate all students: Ability-aware student simulation via cognitive diagnosis guided llm assignment. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 6067–6084, 2026.

Alexander Scarlatos, Nigel Fernandez, Christopher Ormerod, Susan Lottridge, and Andrew Lan. Smart: Simulated students aligned with item response theory for question difficulty prediction. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 25082–25105, 2025.

Alexander Scarlatos, Jaewook Lee, Simon Woodhead, and Andrew Lan. Simulated students in tutoring dialogues: Substance or illusion? In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 42349–42385, 2026.

Yaorui Shi, Yuxin Chen, Zhengxi Lu, Yuchun Miao, Shugui Liu, Qi Gu, Xunliang Cai, Xiang Wang, and An Zhang. Skill1: Unified evolution of skill-augmented agents via reinforcement learning. *arXiv preprint arXiv:2605.06130*, 2026.

Chenxi Wang, Zhuoyun Yu, Xin Xie, Wuguannan Yao, Runnan Fang, Shuofei Qiao, Kexin Cao, Guozhou Zheng, Xiang Qi, Peng Zhang, et al. Skillx: Automatically constructing skill knowledge bases for agents. *arXiv preprint arXiv:2604.04804*, 2026a.

Fei Wang, Zhenya Huang, Qi Liu, Enhong Chen, Yu Yin, Jianhui Ma, and Shijin Wang. Dynamic cognitive diagnosis: An educational priors-enhanced deep knowledge tracing perspective. *IEEE Transactions on Learning Technologies*, 16(3):306–323, 2023a.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023b.

Jiongxiao Wang, Qiaojing Yan, Yawei Wang, Yijun Tian, Soumya Smruti Mishra, Zhichao Xu, Megha Gandhi, Panpan Xu, and Lin Lee Cheong. Reinforcement learning for self-improving agent with skill library. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1529–1550, 2026b.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. *arXiv preprint arXiv:2409.07429*, 2024.

Tao Wu, Jingyuan Chen, Wang Lin, Mengze Li, Yumeng Zhu, Ang Li, Kun Kuang, and Fei Wu. Embracing imperfection: Simulating students with diverse cognitive levels using llm-based agents. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9887–9908, 2025.

Peng Xia, Jianwen Chen, Hanyang Wang, Jiaqi Liu, Kaide Zeng, Yu Wang, Siwei Han, Yiyang Zhou, Xujiang Zhao, Haifeng Chen, et al. Skillrl: Evolving agents via recursive skill-augmented reinforcement learning. *arXiv preprint arXiv:2602.08234*, 2026.

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

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*, pp. 1–26, 2025.

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. *arXiv preprint arXiv:2505.20642*, 2025.

Jiani Zhang, Xingjian Shi, Irwin King, and Dit-Yan Yeung. Dynamic key-value memory networks for knowledge tracing. In *Proceedings of the 26th international conference on World Wide Web*, pp. 765–774, 2017.

Zheyuan Zhang, Daniel Zhang-Li, Jifan Yu, Linlu Gong, Jinchang Zhou, Zhanxin Hao, Jianxiao Jiang, Jie Cao, Huiqin Liu, Zhiyuan Liu, et al. Simulating classroom education with llm-empowered agents. In *Proceedings of the 2025 conference of the nations of the americas chapter of the association for computational linguistics: Human language technologies (volume 1: Long papers)*, pp. 10364–10379, 2025.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19632–19642, 2024.

Guanhao Zhao, Zhenya Huang, Yan Zhuang, Jiayu Liu, Qi Liu, Zhiding Liu, Jinze Wu, and Enhong Chen. Simulating student interactions with two-stage imitation learning for intelligent educational systems. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*, pp. 3423–3432, 2023.

Longwei Zheng, Anna He, Changyong Qi, Haomin Zhang, and Xiaoqing Gu. Cognitive echo: Enhancing think-aloud protocols with llm-based simulated students. *British Journal of Educational Technology*, 56(5):2019–2042, 2025.

A Formal Comparison of Learner Simulation Formulations

This section formalizes the difference between conventional LLM-based learner agents and Learner2Skill in how learner-specific simulation capability is acquired from historical interactions and subsequently used for response simulation. Both formulations consider the same observed interaction history and the same prediction target; the distinction lies in the representation through which information extracted from the history is carried into future simulation.

A.1 Conventional LLM-Based Learner Agents

For learner u, let

$$\mathcal{H}_u^t = \{(e_i, r_i, y_i)\}_{i=1}^t \tag{11}$$

denote the real interaction history observed up to time step t.

Existing LLM-based learner agents commonly process these interactions sequentially to construct or update a learner-specific simulation context. Let M denote a learner agent, $\mathbf{c}_{u,M}^i$ denote the context

maintained by M after observing the first i interactions, and $\mathcal{U}_M$ denote its context-update procedure. The context acquisition process can be expressed as

$$\mathbf{c}_{u,M}^{i} = \mathcal{U}_M\left(\mathbf{c}_{u,M}^{i-1}, (e_i, r_i, y_i)\right), \qquad i = 1, \ldots, t. \tag{12}$$

Given a new task e_{t+1}, the learner agent then generates the simulated response according to

$$(\hat{r}_{t+1}, \hat{y}_{t+1}) \sim q_M\left(r, y \mid e_{t+1}, \mathbf{c}_{u,M}^{t}\right). \tag{13}$$

The specific form of $\mathbf{c}_{u,M}^{t}$ varies across learner agents. It may include historical interactions supplied directly to the LLM, a progressively maintained learner profile or memory, an estimated learner state, or a combination of these representations. Eqs 12–13 therefore provide a general abstraction of this class of methods: learner-specific simulation capability is acquired through an agent-specific context construction process and is subsequently used by the same agent for response generation.

Under this formulation, the resulting context $\mathbf{c}_{u,M}^{t}$ depends on the construction procedure associated with M. As the observed history grows, maintaining this context requires continued processing of learner interactions. When the underlying executor changes, the historical interactions may consequently need to be processed again under the new agent in order to construct a compatible simulation context.

A.2 Learner2Skill Formulation

Learner2Skill separates the learner-specific simulation capability acquired from historical interactions from the downstream executor that subsequently uses it. Given the same observed history $\mathcal{H}_u^t$, we first construct a persistent Simulation Skill:

$$\mathcal{S}_u^t = \text{Skill}\left(\mathcal{H}_u^t\right). \tag{14}$$

For a new task e_{t+1}, a downstream executor M uses the existing Skill under an execution protocol π to generate

$$(\hat{r}_{t+1}, \hat{y}_{t+1}) \sim q_{M,\pi}\left(r, y \mid e_{t+1}, \mathcal{S}_u^t\right). \tag{15}$$

Here, $\mathcal{S}_u^t$ represents the learner-specific simulation capability acquired from the observed history, whereas M and π determine how that capability is used for response generation. This formulation makes the learner-specific representation persistent across downstream executors: when the executor changes, the previously constructed $\mathcal{S}_u^t$ can be retained, while executor-specific adaptation is handled separately.

A.3 Comparison

The two formulations differ primarily in the dependency of the learner representation on the downstream simulator. In conventional learner agents, the context used for simulation is constructed as $\mathbf{c}_{u,M}^{t}$ through an update procedure associated with the current agent M. The acquired learner-specific simulation capability is therefore represented jointly by the maintained context and the agent that constructs and interprets it.

Learner2Skill instead represents the capability acquired from $\mathcal{H}_u^t$ explicitly as $\mathcal{S}_u^t$, and conditions downstream simulation on both this persistent learner-specific representation and the current executor:

$$q_M\left(r, y \mid e_{t+1}, \mathbf{c}_{u,M}^{t}\right) \qquad \text{vs.} \qquad q_{M,\pi}\left(r, y \mid e_{t+1}, \mathcal{S}_u^t\right). \tag{16}$$

This separation allows the learner-specific simulation capability encoded in $\mathcal{S}_u^t$ to remain available when the downstream executor changes. The subsequent sections describe how the Skill is constructed, adapted to a particular executor, and updated from newly observed real interactions.

B Simulation Skill Construction

Simulation Skill Construction organizes the observed real interaction history

$$\mathcal{H}_u^t = \{(e_i, r_i, y_i)\}_{i=1}^{t} \tag{17}$$

into the learner-specific information defined in Section 3.1. Specifically, Learner2Skill constructs the learner's **learning state** $\mathbf{z}_u^t$ and **response patterns** $\mathcal{B}_u^t$. The former represents the learner's knowledge mastery estimated from the real interactions observed up to time t, whereas the latter describes how the learner tends to respond under particular conditions. Although both are derived from the same real interaction history, they are constructed differently: $\mathbf{z}_u^t$ is estimated by the cognitive model, whereas $\mathcal{B}_u^t$ is extracted from real interactions by an LLM and retained only when supported by repeated real evidence.

The LLM used in this section serves as an offline **Skill builder** during Skill Construction and is separated from the downstream LLM executor that subsequently uses the Skill for simulation. Once constructed, $\mathbf{z}_u^t$ and $\mathcal{B}_u^t$ are stored in the Simulation Skill and can be reused when the downstream executor changes. Executor-specific adaptation is handled separately through Executor Calibration in Section 3.2.

B.1 Initializing the Learning State and Task Information

At Skill Construction time, we process all real interactions observed up to time t in their original chronological order and use the trained cognitive model F_θ to estimate the learner's learning state:

$$\mathbf{z}_u^t = F_\theta(\mathcal{H}_u^t) \tag{18}$$

where $\mathbf{z}_u^t \in [0,1]^K$, and its k-th dimension represents the learner's estimated mastery of knowledge concept k. This estimate is stored as the current learning state in the Simulation Skill. The cognitive-model instantiation, training procedure, and learning-state estimation are detailed in Appendix D.

The same cognitive model can also provide task-specific information. For a task e_i, the model can estimate its task difficulty d_{e_i}; many cognitive models can additionally provide the predicted probability that the learner correctly completes the task. During Skill Construction, task difficulty can be used to describe the conditions under which response patterns occur. During subsequent simulation, the runtime uses F_θ to provide information for the current task by default, including the relevant dimensions of the learner's current learning state, the task difficulty, and the predicted probability of correct completion. The cognitive model also remains available as a resource associated with the Skill, allowing the executor to issue additional cognitive-model queries when information beyond the current-task information is needed. These task-specific quantities do not introduce an additional learner state.

B.2 Constructing Response Patterns from Real Interactions

In addition to the learning state, Learner2Skill constructs the response patterns defined in the main text:

$$\mathcal{B}_u^t = \{(c_{u,j}, \tau_{u,j})\}_{j=1}^{J_u^t} \tag{19}$$

where $c_{u,j}$ denotes the condition under which the j-th pattern occurs and $\tau_{u,j}$ denotes the response behavior exhibited by the learner under that condition.

A single real interaction describes the learner's behavior on one particular task and is insufficient by itself to establish a response pattern. We therefore construct $\mathcal{B}_u^t$ by first extracting candidate patterns from the real interaction history, consolidating semantically equivalent candidates across the history, and finally retaining only those supported by sufficient repeated real evidence.

When the construction history is long, we preserve its chronological order and partition it into contiguous segments, each containing at most ℓ_{blk} real interactions. The offline Skill builder processes each segment separately. This segmentation only limits the amount of history processed in a single LLM call; a pattern extracted from one segment remains a candidate until its evidence is consolidated across the complete construction history.

The condition $c_{u,j}$ can include relevant knowledge concepts, task difficulty, and other observable task characteristics that remain meaningful for future tasks. Task difficulty is obtained from the cognitive model in Appendix B.1. To make difficulty-related conditions comparable across tasks, the construction-time difficulty estimates are partitioned into G_{diff} fixed regions, which remain unchanged during subsequent calibration and testing.

The corresponding $\tau_{u,j}$ describes observable response behavior under $c_{u,j}$, such as whether and how the learner attempts a task, typical solution strategies, characteristic errors or omissions, partial

progress, verification or self-correction, and continuation or stopping behavior. These are possible manifestations of $\tau_{u,j}$ rather than additional learner representations.

We do not include the numerical learning-state value $\mathbf{z}_u^t$ directly in $c_{u,j}$. The learning state changes as new real interactions are observed, whereas response patterns represent behavior supported across multiple interactions. Keeping the two forms of learner-specific information separate prevents a response pattern from being tied to a particular mastery value at one time point.

The Skill builder uses the following prompt to extract candidate response patterns:

```
# Response-Pattern Extraction

You are constructing reusable response patterns for one learner.

## Input
A chronological segment of the learner's REAL interactions:
{{RECORDS}}

Each interaction may contain:
- the learning task and observable task information;
- relevant knowledge concepts and task difficulty when available;
- the learner's complete observed response behavior;
- the corresponding observable outcome.

## Goal
Identify candidate patterns describing how this learner tends to
    respond
under identifiable task conditions.

## Rules
1. Use only observable evidence contained in the supplied real
    interactions.
2. A condition may refer to relevant knowledge concepts, task-
    difficulty
   regions, or other observable task characteristics that can
    generalize to
   future tasks.
3. Response behavior may include solution strategies, characteristic
    errors
   or omissions, partial progress, correction, verification, and
    stopping
   behavior.
4. Do not infer unobserved psychological traits, motivations, or
    abilities.
5. A behavior observed only once remains a candidate and must not be
    treated
   as a retained response pattern.
6. Attach the IDs of real interactions that support or conflict with
    each
   candidate. Do not invent interaction IDs.

## Output
For each candidate, return:
- condition
- response_behavior
- support_ids
- conflict_ids

Return an empty list if the supplied records do not support a useful
candidate.
```

Candidates extracted from different chronological segments may describe the same response pattern. Before making the final retention decision, we therefore consolidate candidates whose conditions are identical or semantically equivalent and whose observable response behaviors are the same. Similar behavior under different conditions, or different behavior under the same condition, remains represented separately.

The Skill builder is used only to determine whether natural-language candidate descriptions are semantically equivalent. It does not count evidence or decide whether a candidate should ultimately enter $\mathcal{B}_u^t$. The consolidation prompt is:

```
# Response-Pattern Consolidation

You are consolidating candidate response patterns extracted from
    different
chronological segments of the SAME learner.

## Input
Candidate patterns:
{{CANDIDATE_PATTERNS}}

Each candidate contains:
- condition
- response_behavior
- support_ids
- conflict_ids

## Rules
1. Merge two candidates only when their conditions are identical or
   semantically equivalent AND they describe the same observable
    response
   behavior.
2. Keep candidates separate when similar behavior occurs under
    different
   conditions or when different behavior occurs under the same
    condition.
3. Preserve all attached support_ids and conflict_ids.
4. Do not introduce new learner behavior or new interaction evidence.
5. Do not count evidence and do not make the final retention decision.

## Output
For each consolidated candidate, return:
- condition
- response_behavior
- support_ids
- conflict_ids
```

After consolidation, the final retention decision is made programmatically from the attached real interaction IDs. For candidate pattern j of learner u, let $n_{u,j}^+$ denote the number of distinct real interactions supporting the pattern and $n_{u,j}^-$ the number of distinct real interactions under comparable conditions that conflict with it. We define

$$\rho_{u,j} = \frac{n_{u,j}^+}{n_{u,j}^+ + n_{u,j}^-} \tag{20}$$

and retain a candidate in $\mathcal{B}_u^t$ only if

$$n_{u,j}^+ \geq m_{\text{sup}}, \qquad \rho_{u,j} \geq \eta_{\text{sup}} \tag{21}$$

The first condition requires support from multiple distinct real interactions, while the second prevents behavior with substantial conflicting evidence from being retained. The quantities $n_{u,j}^+$, $n_{u,j}^-$, and $\rho_{u,j}$ are computed directly from the real interaction IDs rather than generated by the LLM.

The final $\mathcal{B}_u^t$ stored in the Simulation Skill contains only the $(c_{u,j}, \tau_{u,j})$ pairs defined in the main text. Supporting and conflicting interaction IDs and their statistics are retained only as backend evidence for later maintenance of the response patterns and are not introduced as additional learner information in the simulation context.

B.3 Organizing the Simulation Skill

After constructing the learning state and response patterns, Learner2Skill organizes them together with the associated resource interfaces as

$$\mathcal{S}_u^t = \left(\texttt{SKILL.md}, \mathbf{z}_u^t, \mathcal{B}_u^t, \mathcal{T}_u\right) \tag{22}$$

where $\mathcal{T}_u$ denotes the resource interfaces associated with the Skill, including access to the cognitive model F_θ and the retrieval tool for the real interaction history stored in the interaction database.

Conceptually, the resulting organization is:

```
Learner2Skill
|
+-- Learner Skills
|   |
|   `-- learner <uid>
|       |
|       +-- SKILL.md
|       |
|       `-- references/
|           +-- knowledge-model.json   [current learning state]
|           `-- response-model.md      [response patterns]
|
+-- Shared Tools
|   |
|   +-- cognitive-model/               [F_theta]
|   `-- retrieval/                     [database access]
|
`-- Interaction Database
    |
    +-- learner <uid-1>                 [real interactions]
    +-- learner <uid-2>                 [real interactions]
    `-- ...
```

The physical storage layout may vary across implementations. Learner2Skill only assumes that each learner Skill can reference its stored learning state and response patterns, invoke the shared tools, and access a queryable interaction database containing the complete real interaction history.

For learner u, the interaction database stores

$$\mathcal{H}_u^t = \{(e_i, r_i, y_i)\}_{i=1}^t \tag{23}$$

with each interaction associated with its original learner. Retrieval of the target learner's own history accesses only the corresponding $\mathcal{H}_u^t$. When permitted by the experimental setting, the database can also provide real responses from other learners to the same task, with learner identity removed before the response is returned to the executor.

This organization separates directly reusable learner information from the complete real interaction history. The Simulation Skill stores the current $\mathbf{z}_u^t$ and $\mathcal{B}_u^t$, while finer-grained historical evidence remains available from the interaction database. To keep the Skill compact and the learner-specific input to each LLM inference concise, we constrain

$$|\mathcal{B}_u^t| \le J_{\max} \tag{24}$$

and retain only currently supported response patterns within this budget. Historical details not explicitly maintained in $\mathcal{B}_u^t$ remain available in the interaction database.

`SKILL.md` template. All learner Skills share the same base `SKILL.md` template. It serves as the unified entry point through which the downstream executor accesses the learner information, current-task information, associated resources, execution protocol, and output requirement.

```
# Learner Simulation Skill

## Objective
{{SIMULATION_OBJECTIVE}}

## Learner Information
Current learning state: {{LEARNING_STATE}}
Response patterns: {{RESPONSE_PATTERNS}}

## Current-Task Information
Relevant learning-state dimensions: {{RELEVANT_LEARNING_STATE}}
Task difficulty: {{TASK_DIFFICULTY}}
Predicted correctness probability: {{PREDICTED_CORRECTNESS}}

## Available Resources
Cognitive model: {{COGNITIVE_MODEL}}
Retrieval tool: {{RETRIEVAL_TOOL}}

## Execution Protocol
<PROTOCOL_START>
{{EXECUTION_PROTOCOL}}
<PROTOCOL_END>

## Output
{{OUTPUT_REQUIREMENT}}
```

The learner information in the Skill remains persistent across tasks, whereas the `Current-Task Information` block is populated at runtime for the current task. It contains the dimensions of $\mathbf{z}_u^t$ associated with the task's knowledge concepts, together with the task difficulty and predicted correctness probability produced by the cognitive model. These quantities are provided before the executor begins the current simulation and do not constitute additional persistent learner-specific information.

The base execution protocol π specifies how the executor uses the Skill without prescribing a fixed sequence of additional resource calls. Given the current task, the learner information, and the current-task information already provided, the executor itself determines whether further information is needed, which associated resource should be queried, and whether additional resource calls are necessary.

A base protocol can be instantiated as follows:

```
<PROTOCOL_START>

1. Read the current task, the learner information stored in the Skill,
   and the current-task information already provided.

2. Determine whether the available information is sufficient to
    simulate the
   learner's likely response.

3. If additional information is needed, decide which associated
    resource is
   most useful:
   - query the cognitive model for additional cognitive information
    beyond
     the current-task information already provided;
```

```
   - invoke retrieval for finer-grained historical behavioral evidence
    .

4. After each additional resource call, reassess whether further
    evidence
   is needed.

5. Generate the response most consistent with the available learner
    evidence,
   including supported errors, partial progress, corrections,
    verification,
   and stopping behavior, rather than producing an ideal solution.

6. Return the required structured simulation output.

<PROTOCOL_END>
```

These decisions are made by the downstream LLM or agent executor. The runtime loads the Skill, provides the current-task information, exposes the available resources, and executes any additional resource calls requested by the executor. The shared base protocol is subsequently calibrated to the current executor in Section 3.2.

B.4 Illustrative Instantiation of a Simulation Skill

To make the representation in Section B.3 concrete, we provide an illustrative instantiation of a Simulation Skill. The example follows exactly the Skill organization and simulation interface used throughout Learner2Skill.

Learner-specific Skill. Suppose learner u has completed the real interactions available before the current task. Skill Construction produces the learner-specific information

```
# Learner Simulation Skill

## Learner Information

Current learning state:
{{CONCEPT-LEVEL MASTERY ESTIMATES}}

Response patterns:

Pattern 1
Condition:
{{CONDITION_1}}

Response tendency:
{{RESPONSE_BEHAVIOR_1}}

Pattern 2
Condition:
{{CONDITION_2}}

Response tendency:
{{RESPONSE_BEHAVIOR_2}}

## Available Resources

Cognitive model:
Provides task-specific cognitive information and supports additional
cognitive queries when needed.
```

```
Retrieval tool:
Provides access to real historical interactions when finer-grained
behavioral evidence is needed.
```

The current learning state corresponds to $\mathbf{z}_u^t$, while the retained response patterns instantiate $\mathcal{B}_u^t$. Together with the common `SKILL.md` interface and the associated resource interfaces $\mathcal{T}_u$, they form the Simulation Skill $\mathcal{S}_u^t$. The complete real interaction history is not stored in the Skill itself, but remains available through the external interaction database.

Current-task instantiation. When a new task e_{t+1} arrives, the runtime uses the cognitive model to provide the task-specific information required by the common simulation interface:

```
## Current Task
{{TASK_CONTENT}}

## Current-Task Information

Relevant learning-state dimensions:
{{RELEVANT_LEARNING_STATE}}

Task difficulty:
{{TASK_DIFFICULTY}}

Predicted correctness probability:
{{PREDICTED_CORRECTNESS}}
```

These quantities are generated for the current task and are therefore separated from the persistent learner information stored in the Skill. The executor begins simulation from the current task, the learner-specific Skill, and this current-task information. If they are insufficient, the executor may additionally query the cognitive model or retrieve real historical evidence from the interaction database.

Skill-guided simulation. After sufficient evidence is available, the executor generates the structured simulation output

```
{
  "attempted": {{TRUE_OR_FALSE}},
  "response_process": {{SIMULATED_RESPONSE_PROCESS}},
  "final_answer": {{SIMULATED_FINAL_ANSWER}},
  "predicted_outcome": {{PREDICTED_OUTCOME}}
}
```

The first three fields constitute the simulated learner response $\hat{r}_{t+1}$, while `predicted_outcome` corresponds to $\hat{y}_{t+1}$. Importantly, all information available to this simulation precedes the current real interaction. Only after the simulation is completed is the real response and outcome observed and appended to the interaction history; the resulting real evidence can then be used for subsequent Skill Evolution. Simulated responses are never treated as learner evidence.

B.5 Retrieval Tool

The learning state and response patterns preserve learner-specific information expected to remain useful for future tasks, but they do not explicitly retain every detail in $\mathcal{H}_u^t$. Learner2Skill therefore stores the complete real interaction history in the interaction database and allows the executor to access it through the retrieval tool in $\mathcal{T}_u$ when additional evidence is needed.

The retrieval tool can provide four types of evidence: the learner's recent interactions, past interactions involving the current knowledge concepts, past interactions related to the current task content, and, when permitted by the experimental setting, anonymized responses from other learners to the same task.

For a historical interaction (e_i, r_i, y_i) and the current task e, the retrieval tool measures relevance from three complementary aspects: knowledge-concept relevance, semantic relevance, and recency. Let $\mathcal{K}(e_i)$ and $\mathcal{K}(e)$ denote the knowledge-concept sets associated with the historical and current tasks. Knowledge-concept relevance is

$$s_{\text{concept}}((e_i, r_i, y_i), e) = \frac{|\mathcal{K}(e_i) \cap \mathcal{K}(e)|}{|\mathcal{K}(e_i) \cup \mathcal{K}(e)|} \tag{25}$$

Let $\bar{\mathbf{v}}_{e_i}$ and $\bar{\mathbf{v}}_e$ denote the L_2-normalized representations of the two tasks. Semantic relevance is

$$s_{\text{semantic}}((e_i, r_i, y_i), e) = \frac{1 + \bar{\mathbf{v}}_{e_i}^\top \bar{\mathbf{v}}_e}{2} \tag{26}$$

Because the real interaction history follows chronological order, recency is defined as

$$s_{\text{recency}}((e_i, r_i, y_i), e) = \frac{i}{t}, \qquad 1 \le i \le t \tag{27}$$

We use a fixed equal-weight combination:

$$s((e_i, r_i, y_i), e) = s_{\text{concept}}((e_i, r_i, y_i), e) + s_{\text{semantic}}((e_i, r_i, y_i), e) + s_{\text{recency}}((e_i, r_i, y_i), e) \tag{28}$$

The executor specifies the information it needs, and the retrieval tool first selects candidate records consistent with that request. When multiple candidate records remain, they are ranked according to Eq. 28. The ranking function determines the ordering of relevant records rather than whether retrieval should occur.

Whether additional historical evidence is needed, what type of evidence to request, how many records to request, and whether another retrieval is necessary after inspecting the returned evidence are all determined by the executor.

When the target learner's current learning state, response patterns, current-task information, and own real interaction history remain insufficient for the current simulation, the executor may request anonymized real responses from other learners to the same task if such access is permitted by the experimental setting. Evidence from the target learner has higher priority, and responses from other learners are used only after the executor determines that learner-specific evidence remains insufficient.

All retrieval is subject to the current prediction time. Suppose the target learner's observed real interaction history is

$$\mathcal{H}_u^t = \{(e_i, r_i, y_i)\}_{i=1}^t \tag{29}$$

When simulating the response to e_{t+1}, the retrieval tool can access only $\mathcal{H}_u^t$. The not-yet-observed interaction $(e_{t+1}, r_{t+1}, y_{t+1})$ and all subsequent interactions are unavailable to the current simulation. The same temporal principle applies to cross-learner retrieval according to the corresponding evaluation protocol.

Learner2Skill does not use a fixed number of retrieval actions or a fixed number of returned records as method hyperparameters. The executor adaptively determines whether retrieval is needed, which evidence to request, and how much evidence is useful according to the current task and the information already available. In our implementation, approximately 3–5 retrieval actions can be provided in `SKILL.md` as non-binding execution guidance when additional historical evidence is needed. This range is not tuned as a method hyperparameter. The executor may use fewer retrievals, perform no retrieval at all, or continue requesting evidence when additional information remains useful.

B.6 Equipping an LLM Executor with a Simulation Skill

The constructed Simulation Skill is used through a common simulation interface. When a downstream LLM executor is equipped with learner u's Skill, the runtime loads the corresponding `SKILL.md`, exposes the stored learner information and associated resources, and provides the current task e_{t+1}.

Before the executor begins simulation, the runtime also uses the cognitive model to provide current-task information. Specifically, it selects the dimensions of the learner's current learning state $\mathbf{z}_u^t$ corresponding to the knowledge concepts involved in e_{t+1} and provides them together with the estimated task difficulty and the predicted probability of correct completion. This information is specific to the current task and is provided by default rather than requested through an additional executor action.

The executor therefore begins with the current task, the persistent learner information in the Skill, and the current-task information already provided. It then determines whether this information is sufficient for simulation. If additional cognitive information is useful, the executor may issue a further query to the cognitive model. If finer-grained historical behavioral evidence is needed, it may invoke the retrieval tool to query the interaction database. After every additional resource call, the executor reassesses information sufficiency and determines whether another call is necessary.

The common simulation request is instantiated as

```
# Learner Simulation

Instruction:
Use the installed Simulation Skill, the current-task information
    already
provided, and the execution protocol to simulate the likely response
    behavior
of the target learner.

Learning task:
{{TASK_CONTENT}}

Output:
{{OUTPUT_REQUIREMENT}}
```

All accessible evidence is restricted to information available before the current real interaction. When simulating e_{t+1}, the available target learner history is

$$\mathcal{H}_u^t = \{(e_i, r_i, y_i)\}_{i=1}^t \tag{30}$$

Once the executor determines that sufficient evidence is available, it generates

$$(\hat{r}_{t+1}, \hat{y}_{t+1}) \sim q_M\left(r, y \mid e_{t+1}, \mathcal{S}_u^t\right) \tag{31}$$

Consistent with the problem formulation in Section 3, the implementation organizes the simulated result into the following four fields:

- `attempted`: whether the learner attempts the current task;
- `response_process`: the simulated response process, including partial progress, errors, corrections, verification, and stopping behavior when applicable;
- `final_answer`: the final answer produced by the simulated learner, or `null` if no final answer is reached;
- `predicted_outcome`: the corresponding observable outcome, such as correctness or score.

The first three fields instantiate the simulated response $\hat{r}$, while `predicted_outcome` corresponds to $\hat{y}$.

If the simulated learner does not attempt the task, `attempted` is false, `response_process` is empty, and `final_answer` is `null`. If the learner begins an attempt but stops before reaching a final answer, the generated partial response remains in `response_process`, while `final_answer` remains `null`.

After simulation, the real interaction

$$(e_{t+1}, r_{t+1}, y_{t+1}) \tag{32}$$

is appended to the interaction database only after the real response and outcome have been observed. It is then used by Skill Evolution to update the learning state and response patterns. Simulated responses are never used as learner evidence.

When the downstream executor changes from M to another executor M', the constructed learner Skill and the complete real interaction history in the database are retained. Executor Calibration adapts the shared execution protocol to the new executor, after which M' uses the same learner Skill through the common simulation interface.

C EXPERIMENTS

To systematically evaluate Learner2Skill, we organize the experiments around four research questions. **RQ1:** Can Learner2Skill reproduce learner response behavior more accurately while reducing simulation cost? **RQ2:** Can a Simulation Skill constructed once be reused across different LLM executors? **RQ3:** What is the contribution of each component in Learner2Skill? **RQ4:** Can Learner2Skill support practical educational applications?

C.1 EXPERIMENTAL SETUP

Dataset Many existing open educational datasets provide large-scale records of learner performance or final answers, but datasets containing detailed problem-solving processes at scale remain scarce. To support fine-grained learner simulation, we collaborated with a real online learning platform for high-school students in China and constructed **LearnerTrace-2K**, whose textual content was translated into English for our experiments. The dataset contains temporally ordered practice records from 2,000 high-school students in mathematics and physics, covering both multiple-choice and application questions. In addition to exercises and outcomes, the platform prompts learners to record their main problem-solving ideas and intermediate steps, as illustrated in Figure 4. Each interaction therefore captures whether the learner attempted the problem, how the learner approached it, the final answer, and the corresponding score.

LearnerTrace-2K contains **52,566 learner–exercise interactions, 2,076 exercises, and 709 associated knowledge concepts**. All interactions retain their original practice order. Following this chronology, each learner's trajectory is split into **60%/10%/30%** construction, calibration, and test portions. The first 60% is used for **Skill Construction**, the following 10% provides real interaction feedback for **Executor Calibration**, and the final 30% is used exclusively for final evaluation with **Skill Evolution**. Further details on data collection, statistics, quality control of the recorded problem-solving processes, and representative examples are provided in Appendix H.1. We plan to release LearnerTrace-2K to support future research on learner modeling and fine-grained learner simulation.

Baselines We first compare Learner2Skill with existing **LLM-based learner simulators**, including **EduAgent, Agent4Edu, and CoderAgent**. We additionally construct **PromptLLM** to measure the simulation capability of a general-purpose LLM without a dedicated learner-modeling mechanism. PromptLLM directly conditions on the learner's previously observed responses and generates the learner's behavior on the next exercise.

As complementary comparisons, we further include supervised learner-performance prediction methods, including **KES, DKVMN, EERNN (with Markov), SAKT, and DAISIM**. These methods provide low-cost predictions of future learner performance, but operate primarily at the correctness level and do not generate fine-grained problem-solving behavior. Detailed descriptions and implementation settings of all baselines are provided in Appendix H.2.

Implementation Details Following the chronological split above, Learner2Skill constructs each learner's Simulation Skill from the first 60% of the trajectory. The subsequent 10% is used exclusively for Executor Calibration. Within calibration, learners are randomly divided into **70%/30%**: the former are used to identify recurring Skill-usage errors and propose protocol updates, while the latter are used to validate candidate protocols on learners not involved in protocol editing. After calibration, these observed real interactions are incorporated chronologically to synchronize the corresponding Skills before testing. The final 30% is then processed sequentially for evaluation and test-time Skill Evolution.

The cognitive model used by Learner2Skill is trained and selected entirely within the Skill Construction portion. Specifically, each learner's construction trajectory is further divided chronologically into an internal training subset and an internal held-out subset for checkpoint selection. No interaction from the Executor Calibration or final-test portions is used to optimize or select the cognitive model. After checkpoint selection, the cognitive model is fixed throughout subsequent Skill Construction, Executor Calibration, and final evaluation. Its architecture and training procedure are described in Appendix D.

For supervised learner-performance predictors, models are trained on the first 60% of the trajectory and selected according to performance on the following 10%. Existing LLM-based simulation baselines do not define a separate calibration stage; therefore, both the first 60% and the subsequent 10% are available as pre-test learner history before their evaluation begins. **All methods are ultimately evaluated only on the same final 30% test interactions.**

All LLM-based methods are accessed through APIs and use the same underlying simulation executor and generation settings within each comparison. We use **Claude Sonnet 4.6** and **Qwen3** as simulation executors and employ an independent **GPT-5.5** as the semantic evaluator. Unless otherwise specified, the generation temperature is set to 0.7 and top-p to 0.8. Training is performed on a single NVIDIA A100 80GB GPU. All methods are independently run **three times** under the same experimental setting, and results are aggregated across runs. Complete implementation settings and evaluation details are provided in Appendix H.3 and Appendix H.4.

D Cognitive-Model Instantiation

Section 3.1 uses a cognitive model F_θ to estimate the learner's learning state at time t from the observed real interactions:

$$\mathbf{z}_u^t = F_\theta(\mathcal{H}_u^t) \in [0,1]^K, \tag{33}$$

where K denotes the number of knowledge concepts and the k-th dimension of $\mathbf{z}_u^t$ represents the learner's estimated mastery of concept k at time t. Learner2Skill obtains the concept-level learning state through F_θ. The particular cognitive model can therefore be selected according to the data and application setting, provided that it can estimate the learner's time-indexed learning state from the real interactions observed up to the corresponding time. Our focus is on organizing and using such learning states for reusable learner simulation rather than identifying an optimal cognitive-model architecture. We therefore instantiate F_θ with a representative cognitive model while keeping this component replaceable.

Knowledge-concept mastery is a latent state and cannot be observed directly. What can be observed is the sequence of learning tasks and the corresponding outcomes. A common way to estimate such a latent state is therefore to require it to explain the learner's observed outcomes on different tasks. The cognitive model represents the learner by $\mathbf{z}_u^t$, combines this state with the knowledge concepts and parameters of a task, and predicts the corresponding outcome. The latent state is then estimated indirectly by fitting these predictions to the observed outcomes. This connects the inferred learning state to observable learner behavior rather than treating it as an unconstrained hidden representation.

In this work, we instantiate F_θ with DNeuralCDM

D.1 DNeuralCDM and Task-Representation Modification

DNeuralCDM estimates an unobservable learning state from the learner's observed outcomes over a sequence of tasks. For the t-th real interaction, let $\mathbf{q}_t \in \{0,1\}^K$ denote the knowledge-concept mask of task e_t, where $q_{t,k} = 1$ indicates that the task involves concept k. For the binary outcome setting used by the cognitive model, $y_t \in \{0,1\}$ indicates whether the learner correctly completes task e_t. The interaction is encoded as

$$\mathbf{x}_t = \mathbf{q}_t \otimes [y_t, 1-y_t] \in \mathbb{R}^{2K}, \tag{34}$$

where $\otimes$ separately represents correct and incorrect observations for each knowledge concept. The interaction representation is first embedded as

$$\mathbf{a}_t = W_x \mathbf{x}_t + \mathbf{b}_x, \tag{35}$$

and the historical interactions are then processed chronologically:

$$\mathbf{h}_t = \mathrm{LSTM}\left(\mathbf{a}_t, \mathbf{h}_{t-1}\right). \tag{36}$$

The learner's concept-level learning state is obtained from the recurrent state:

$$\mathbf{z}_u^t = \sigma\left(W_z \mathbf{h}_t + \mathbf{b}_z\right) \in [0,1]^K. \tag{37}$$

The state $\mathbf{z}_u^t$ has no direct supervision because knowledge-concept mastery is latent. DNeuralCDM instead learns this state by requiring it to explain subsequent observed outcomes. For a task e, let $\mathbf{d}_e \in [0,1]^K$ denote its concept-level difficulty, $\alpha_e > 0$ its discrimination, and $\mathbf{q}_e$ its concept mask. Given the current learning state, the model constructs the task-conditioned interaction

$$\mathbf{g}_{u,e}^t = \alpha_e \left(\mathbf{z}_u^t - \mathbf{d}_e\right) \odot \mathbf{q}_e. \tag{38}$$

The resulting representation is passed through the outcome-prediction network:

$$\mathbf{r}_{u,e}^{(1)} = \tanh\left(W_1 \mathbf{g}_{u,e}^t + \mathbf{b}_1\right), \tag{39}$$

$$\mathbf{r}_{u,e}^{(2)} = \tanh\left(W_2 \mathbf{r}_{u,e}^{(1)} + \mathbf{b}_2\right), \tag{40}$$

$$P_\theta\left(Y = 1 \mid e, \mathbf{z}_u^t\right) = \sigma\left(\mathbf{w}_3^\top \mathbf{r}_{u,e}^{(2)} + b_3\right). \tag{41}$$

DNeuralCDM constrains the weights of the outcome-prediction network to be non-negative. We additionally parameterize the discrimination term as positive. Because both $\tanh(\cdot)$ and $\sigma(\cdot)$ are monotonically increasing, for a knowledge concept involved in the current task the prediction satisfies

$$\frac{\partial P_\theta(Y = 1 \mid e, \mathbf{z}_u^t)}{\partial z_{u,k}^t} \geq 0, \qquad \frac{\partial P_\theta(Y = 1 \mid e, \mathbf{z}_u^t)}{\partial d_{e,k}} \leq 0, \qquad q_{e,k} = 1. \tag{42}$$

This monotonicity captures a standard psychometric assumption: holding other factors fixed, stronger mastery of a task-relevant knowledge concept should not decrease the probability of a correct outcome, whereas greater difficulty on that concept should not increase it. Observed outcomes can therefore supervise the latent learning state through a prediction function with an explicit concept-level interpretation.

To support tasks not observed during cognitive-model training, we replace the task-identity representation used for task-parameter estimation with one derived from task content. For a learning task e, let s_e denote its textual content. A frozen text encoder ϕ extracts a semantic representation

$$\mathbf{v}_e = \phi(s_e) \in \mathbb{R}^d. \tag{43}$$

In our implementation, ϕ is instantiated with BGE-M3. The representation is normalized as

$$\bar{\mathbf{v}}_e = \frac{\mathbf{v}_e}{\|\mathbf{v}_e\|_2}. \tag{44}$$

The task representation contains only task content available before the learner produces the current response. Learner responses and observed outcomes are not included in this representation.

The text encoder only provides the semantic representation of the task. Task difficulty and discrimination are estimated by two learnable neural branches on top of this representation:

$$\mathbf{d}_e = \sigma\left(W_d \bar{\mathbf{v}}_e + \mathbf{b}_d\right), \tag{45}$$

and

$$\alpha_e = \mathrm{softplus}\left(W_\alpha \bar{\mathbf{v}}_e + b_\alpha\right). \tag{46}$$

Here, $\mathbf{d}_e \in [0,1]^K$ represents the difficulty of task e over the K knowledge concepts, and $\alpha_e > 0$ denotes its discrimination. The parameters W_d, $\mathbf{b}_d$, W_α, and b_α are learned jointly with the remaining cognitive-model parameters from observed learner outcomes. Thus, BGE-M3 supplies the frozen semantic representation of a task, while its difficulty and discrimination are learned from learner-response data.

For a task that does not appear during cognitive-model training, its content can be encoded by the same frozen text encoder, and the learned task-parameter branches can directly estimate $\mathbf{d}_e$ and α_e, provided that the corresponding knowledge-concept association is available. These task parameters are then combined with the learner's learning state through the same outcome-prediction mechanism defined above.

Our structural modification is therefore confined to the task representation and task-parameter generation pathway. The historical-interaction encoding, dynamic learning-state update, and outcome-prediction mechanism otherwise retain the DNeuralCDM formulation.

D.2 COGNITIVE-MODEL TRAINING

The cognitive model is trained so that the latent learning state inferred from historical real interactions can explain the learner's subsequent observed outcomes. Following the problem formulation in Section 3, the real interaction history of learner u is

$$\mathcal{H}_u^T = \{(e_t, r_t, y_t)\}_{t=1}^T, \tag{47}$$

where y_t denotes the observable outcome defined in the main text. For the binary cognitive-model instantiation used here, $y_t \in \{0, 1\}$ indicates whether the learner correctly completes task e_t. Cognitive-model training uses task e_t, its concept mask $\mathbf{q}_t$, and the observed outcome y_t. The complete response r_t is used separately for constructing the response patterns described in Section 3.1.

Training follows the chronological order of the real interactions. At time t, the model first forms the learning state from the interactions already observed before the current task:

$$\mathbf{z}_u^{t-1} = F_\theta\left(\mathcal{H}_u^{t-1}\right). \tag{48}$$

The model then combines this state with the parameters and concept association of task e_t to predict the probability of a correct outcome:

$$\hat{y}_t = P_\theta\left(Y_t = 1 \mid e_t, \mathbf{z}_u^{t-1}\right). \tag{49}$$

Let $\Omega_{\mathrm{CD-tr}}$ denote the valid prediction positions used for cognitive-model parameter optimization. The model is optimized using binary cross-entropy:

$$\mathcal{L}_{\mathrm{CD}} = -\frac{1}{|\Omega_{\mathrm{CD-tr}}|} \sum_{(u,t)\in\Omega_{\mathrm{CD-tr}}} \left[y_t \log \hat{y}_t + (1 - y_t)\log(1 - \hat{y}_t)\right]. \tag{50}$$

This objective jointly trains the dynamic learning-state estimator, the task-parameter branches, and the outcome-prediction network. Although $\mathbf{z}_u^t$, $\mathbf{d}_e$, and α_e do not receive direct labels, they are jointly constrained by their ability to explain the observed outcomes.

Importantly, both cognitive-model training and checkpoint selection are restricted to the **Skill Construction portion** of each learner's chronological trajectory. Within the first 60% of interactions assigned to Skill Construction, we further use the first 90% as the internal cognitive-model training subset $\mathcal{D}_{\mathrm{CD-tr}}$ and the remaining 10% as the internal held-out subset $\mathcal{D}_{\mathrm{CD-val}}$. This internal split also preserves chronological order. During validation, the preceding interactions remain available as state context, while validation loss is computed only over the prediction positions belonging to $\mathcal{D}_{\mathrm{CD-val}}$.

The internal cognitive-model validation subset should therefore not be confused with the subsequent 10% portion of the complete learner trajectory used for Executor Calibration. Neither Executor Calibration interactions nor final-test interactions are used to optimize or select F_θ.

The text encoder ϕ remains frozen throughout cognitive-model training. Task representations are precomputed before training with `BAAI/bge-m3`. We use its 1024-dimensional dense representation, encode task texts in FP16 with a batch size of 32 and a maximum input length of 8192 tokens, and apply L_2 normalization before storing the resulting representations.

The interaction representation in Eq. 34 is mapped to the input embedding used by the recurrent learner-state encoder. The LSTM hidden dimension is 256. The outcome-prediction network contains two hidden layers of dimensions 512 and 256. The task-difficulty branch maps the 1024-dimensional task representation to K concept-level difficulty values, while the task-discrimination branch maps the same representation to one discrimination value.

All trainable weight matrices are initialized with Xavier-normal initialization. We optimize the model using Adam with learning rate 10^{-3}. Learners have variable-length interaction sequences; accordingly, each training mini-batch contains one complete learner sequence, while the interactions within the sequence retain their original chronological order. The order of learner sequences is shuffled across training epochs.

Training runs for at most 100 epochs. We apply early stopping when the internal-validation binary cross-entropy, equivalently reported as negative log-likelihood, does not improve for 10 consecutive

Table 4: Implementation settings of the cognitive model.

Setting	Value
Cognitive model	Text-augmented DNeuralCDM
Text encoder	BAAI/bge-m3
Text embedding dimension	1024
Text encoder	Frozen
Embedding representation	Dense
Embedding normalization	L_2 normalization
Embedding precision	FP16
Embedding batch size	32
Maximum text length	8192 tokens
LSTM hidden dimension	256
Prediction hidden dimensions	512, 256
Task-difficulty branch	$1024 \to K$
Task-discrimination branch	$1024 \to 1$
Sequence batch size	1 learner
Optimizer	Adam
Learning rate	10^{-3}
Maximum epochs	100
Early-stopping patience	10 epochs
Training objective	Binary cross-entropy
Checkpoint criterion	Internal-validation BCE/NLL
Weight initialization	Xavier normal
Monotonicity constraint	Non-negative weight clipping
Learning-rate scheduler	None
Device	CUDA if available; otherwise CPU

epochs. After each parameter update, the weights in the outcome-prediction network subject to the monotonicity constraint are clipped to remain non-negative. The checkpoint with the lowest BCE/NLL on $\mathcal{D}_{\mathrm{CD-val}}$ is retained as the final cognitive model. Accuracy and AUC are additionally recorded for monitoring but are not used for checkpoint selection.

Table 4 summarizes the implementation settings.

After checkpoint selection, the trained cognitive model F_θ is fixed. It is subsequently used to construct the learner's initial learning state, provide task-conditioned cognitive information during simulation, and produce the frozen-model base estimate used by test-time Skill Evolution.

D.3 Learning-State Estimation and Outcome Prediction

After training, the primary role of F_θ is to estimate the learner's learning state from the real interactions observed so far. Given

$$\mathcal{H}_u^t = \{(e_i, r_i, y_i)\}_{i=1}^t, \tag{51}$$

the model processes the interactions chronologically. For each interaction i, the concept mask $\mathbf{q}_i$ and observed outcome y_i define $\mathbf{x}_i$ according to Eq. 34; the recurrent state is then updated according to Eq. 36. After all real interactions available at time t have been processed, the resulting

$$\mathbf{z}_u^t = F_\theta(\mathcal{H}_u^t) \tag{52}$$

is the learning state used in Section 3.1. It summarizes the learner's estimated mastery of the K knowledge concepts and is stored as learner-specific information in the Simulation Skill.

For a current task e, the trained cognitive model also provides two task-conditioned quantities. The task-parameter branch first produces the concept-level difficulty $\mathbf{d}_e$. We report a scalar task difficulty by averaging over the concepts involved in the current task:

$$d_e = \frac{\mathbf{q}_e^\top \mathbf{d}_e}{\|\mathbf{q}_e\|_1}. \tag{53}$$

The same cognitive model combines the current learning state with the task parameters to obtain the predicted probability that learner u correctly completes task e:

$$p_{u,e}^{\mathrm{CD}} = P_\theta\left(Y = 1 \mid e, \mathbf{z}_u^t\right). \tag{54}$$

For each simulation, the runtime selects the dimensions of $\mathbf{z}_u^t$ corresponding to the knowledge concepts involved in the current task and provides them to the executor together with d_e and $p_{u,e}^{\mathrm{CD}}$. These quantities constitute the current-task information described in Appendix B.6 and are provided before the executor begins the current simulation.

This current-task information complements, rather than replaces, the persistent learner information maintained by the Skill. The full current learning state $\mathbf{z}_u^t$ remains stored in the learner Skill, while the task-relevant state dimensions, task difficulty, and predicted correctness probability make the information most directly related to the current task immediately available to the executor.

The cognitive model remains associated with the Skill during simulation. After inspecting the learner information and current-task information already provided, the executor can further query the cognitive model when additional cognitive information is needed. This additional use is distinct from the default current-task information above and does not introduce another learner state.

D.4 Use during Test-Time Skill Evolution

During the test-time Skill Evolution described in Section 3.3, the trained F_θ remains frozen. After a new real interaction has been observed and appended to the learner's real interaction history, the cognitive model produces a base learning-state estimate from the expanded history:

$$\tilde{\mathbf{z}}_u^t = F_\theta(\mathcal{H}_u^t). \tag{55}$$

The learner-specific residual defined in Section 3.3 is then applied to obtain the updated learning state:

$$\mathbf{z}_u^t = \Pi_{[0,1]^K}\left(\tilde{\mathbf{z}}_u^t + \boldsymbol{\delta}_u^t\right). \tag{56}$$

Thus, F_θ continues to provide the concept-level base estimate, while the learner-specific correction allows the state stored in the Skill to respond to newly observed real behavior without updating the cognitive-model parameters.

The correction is further restricted to knowledge concepts that occur in the recent observation window. Consequently, recent evidence updates only concept dimensions supported by newly observed real interactions, while the remaining dimensions continue to follow the frozen cognitive-model estimate.

After the latest learning state has been stored in the Skill, subsequent simulations use this updated state. For each new current task, the corresponding task-relevant state dimensions, task difficulty, and predicted correctness probability are again provided to the executor before simulation.

E Executor Calibration

Executor Calibration uses a small amount of real interaction feedback observed after the learner-specific Simulation Skills have been constructed and before final testing. At this point, the learner-specific simulation capability has already been organized into the learning state, response patterns, and associated resources of each Skill. The purpose of calibration is therefore to adapt the shared execution protocol π to the current downstream executor M, rather than to reconstruct learner-specific information.

Because the execution protocol is shared across all learner Skills, calibration focuses on Skill-usage errors that recur across different learners. The key signal is thus not a common learner behavior, but a recurring pattern in how the current executor interprets the information already provided, decides whether additional evidence is needed, combines information from different sources, or reflects the available learner evidence in its generated response. This cross-learner perspective allows each learner to contribute only a small amount of real interaction feedback while the shared protocol is refined from evidence observed across multiple learners.

Throughout calibration, the executor parameters remain fixed. The learning state, response patterns, the current-task information provided before each simulation, and the real historical evidence accessible through each Skill also remain unchanged across protocol comparisons. Different protocols may lead the executor to request different additional information, but they begin from the same learner information and the same accessible evidence space. Only the execution protocol in `SKILL.md` is

Algorithm 1 Training and Use of the Cognitive Model

Require: Internal cognitive-model training interactions $\mathcal{D}_{\mathrm{CD-tr}}$, internal cognitive-model validation interactions $\mathcal{D}_{\mathrm{CD-val}}$, task texts and concept masks, frozen text encoder ϕ
Ensure: Trained cognitive model F_θ

Training
Encode each task text s_e using ϕ and normalize $\bar{\mathbf{v}}_e$
Initialize the trainable parameters of F_θ
repeat
  for each learner sequence in $\mathcal{D}_{\mathrm{CD-tr}}$ **do**
    Process the real interactions in chronological order
    Construct the interaction representation using Eq. 34
    Update the learning state using Eqs. 36–37
    Estimate task difficulty and discrimination using Eqs. 45–46
    Compute the predicted correctness probability using Eqs. 38–41
    Update the trainable parameters using Eq. 50
    Enforce non-negative weights in the outcome-prediction network
  end for
  Evaluate BCE/NLL on $\mathcal{D}_{\mathrm{CD-val}}$
  Retain the checkpoint with the lowest internal-validation BCE/NLL
until internal-validation BCE/NLL does not improve for 10 epochs or 100 epochs are reached
Fix the selected cognitive model F_θ

Learning-State Estimation
Input: observed real interaction history $\mathcal{H}_u^t$
Process $\mathcal{H}_u^t$ chronologically using Eqs. 34–37
Obtain and store the learning state $\mathbf{z}_u^t$

Current-Task Information
Input: current learning state $\mathbf{z}_u^t$ and current task e
Select the dimensions of $\mathbf{z}_u^t$ relevant to e
Estimate the current task parameters using Eqs. 45–46
Compute d_e using Eq. 53
Compute $p_{u,e}^{\mathrm{CD}}$ using Eq. 54
Provide the relevant state dimensions, d_e, and $p_{u,e}^{\mathrm{CD}}$ to the executor before simulation

Test-Time Skill Evolution
After a new real interaction is observed, obtain the frozen cognitive-model base state from the updated real history
Optimize only the learner-specific correction according to Section 3.3
Store the resulting updated learning state for subsequent simulations

updated. After the final protocol $\pi_M^\star$ is selected, the real interactions observed during calibration are incorporated chronologically into the corresponding learners' real histories and their Skills are synchronized before final testing.

E.1 Preparing Calibration Feedback

As defined in Section 3.2, the calibration set is

$$\mathcal{D}^{\mathrm{cal}} = \{(u_i, e_i, r_i, y_i)\}_{i=1}^{N_c}, \tag{57}$$

where u_i denotes the learner, e_i the learning task, r_i the real learner response, and y_i the corresponding observable outcome. These interactions are observed after Skill Construction and before final testing.

The participating learners are divided into two disjoint sets:

$$\mathcal{U}_{\mathrm{fit}} \cap \mathcal{U}_{\mathrm{val}} = \varnothing. \tag{58}$$

Learners in $\mathcal{U}_{\text{fit}}$ are used to identify recurring Skill-usage errors and propose protocol edits. Learners in $\mathcal{U}_{\text{val}}$ do not participate in error identification or protocol editing and are used only to determine whether a candidate protocol improves Skill use on learners not involved in its construction.

At calibration round k, the fixed executor M uses the already constructed Skill of each learner in $\mathcal{U}_{\text{fit}}$ together with the current execution protocol $\pi^{(k)}$ to simulate the corresponding calibration tasks through the common simulation interface in Appendix B.6. Before each simulation, the executor receives the learner information stored in the Skill together with the current-task information provided for the current task, including the relevant dimensions of the learner's learning state, the estimated task difficulty, and the predicted probability of correct completion. The real response and outcome are revealed only after simulation and are then used as calibration feedback.

For each calibration case, the analysis therefore has access to:

- the current learning task;
- the learning state and response patterns available before simulation;
- the current-task information already provided before simulation, including the relevant learning-state dimensions, task difficulty, and predicted correctness probability;
- the additional Skill resources available to the executor;
- any additional cognitive-model queries or retrieval calls actually made by the executor, together with the information returned by those calls;
- the simulated response and predicted outcome; and
- the real response and outcome revealed after simulation.

The real calibration response is used only as feedback at this stage. It is not added to the learner's real interaction history during protocol calibration, and it does not update the learning state or response patterns. Consequently, different protocol candidates are compared using the same learner information, the same current-task information, and the same accessible historical evidence.

E.2 Identifying Recurring Skill-Usage Errors

The central step of Executor Calibration is to identify Skill-usage errors that recur across different learners in $\mathcal{U}_{\text{fit}}$.

The analysis is performed jointly over calibration cases from multiple learners rather than independently diagnosing each learner and subsequently aggregating learner-level labels. This distinction is important: the goal is to identify a shared problem in how the executor uses Simulation Skills, not to discover a behavioral pattern shared by the learners themselves.

For each calibration case, we compare the simulated response with the real response revealed afterward and examine how the executor used both the information already provided before simulation and any additional evidence obtained through subsequent resource calls. Following Section 3.2, the analysis considers whether the executor repeatedly exhibits problems in how it:

- **interprets** the learning state, response patterns, current-task information, or information returned by additional resource calls;
- **requests additional evidence** when the information already provided is insufficient, including deciding whether further cognitive information or historical evidence is useful;
- **combines** the persistent learner information, current-task information, and any additionally obtained evidence; or
- **reflects** the available learner evidence in the generated response.

These aspects are used to analyze Skill use and do not define separate failure types. A recurring Skill-usage error may involve more than one of them.

When the calibration cases cannot be analyzed within a single context, they are organized into mixed-learner batches. Each batch contains cases from multiple learners in $\mathcal{U}_{\text{fit}}$, so that every analysis directly compares executor behavior across different learner Skills. The analysis is therefore

performed jointly across learners rather than by first producing independent diagnoses for individual learners and subsequently aggregating them.

The following prompt implements this cross-learner analysis:

```
============================================================
 CROSS-LEARNER SKILL-USAGE ERROR IDENTIFICATION
============================================================

ROLE
Analyze how the current fixed executor uses Simulation Skills
across MULTIPLE different learners.

INPUT
Calibration cases from multiple learners.

For each case, you are given:
- learner identifier;
- current learning task;
- learning state available before simulation;
- response patterns available before simulation;
- current-task information ALREADY provided before simulation,
  including:
  * relevant learning-state dimensions;
  * task difficulty;
  * predicted correctness probability;
- additional Skill resources available to the executor;
- additional cognitive-model queries or retrieval calls actually
  made by the executor;
- information and historical evidence returned by those calls;
- simulated response and predicted outcome;
- real response and observed outcome revealed AFTER simulation.

GOAL
Jointly compare the calibration cases and identify recurring
problems in how the executor uses the Simulation Skill.

The target is a recurring EXECUTOR Skill-usage problem across
different learner Skills, NOT a behavioral pattern shared by learners.

ANALYSIS
Examine whether the executor repeatedly has difficulty in:

1. INTERPRETING
   correctly using the persistent learner information and the
   current-task information already provided before simulation;

2. REQUESTING ADDITIONAL EVIDENCE
   correctly deciding whether further cognitive-model queries or
   historical retrieval are needed beyond the information already
   provided;

3. COMBINING
   coherently integrating the learning state, response patterns,
   current-task information, and any additionally obtained evidence;

4. REFLECTING
   faithfully expressing the available learner evidence in the
   simulated learner response.

RULES
1. Analyze calibration cases JOINTLY across different learners.
```

```
2. Do not independently diagnose each learner and then simply
   summarize the individual diagnoses.

3. Treat the real learner response and outcome only as feedback
   revealed AFTER simulation.

4. The current-task information listed above is already available
   before simulation. Do not treat obtaining this default information
   as an executor decision or an additional resource call.

5. Focus only on problems that can be addressed through the
   shared execution protocol.

6. Do not propose changes to:
   - learning states;
   - response patterns;
   - the default current-task information;
   - cognitive-model parameters;
   - historical records in the interaction database;
   - available tools;
   - executor model parameters.

7. Do not turn a learner-specific behavior, a task-specific answer,
   or an isolated simulation mismatch into a shared protocol rule.

8. Retain a problem only when similar executor-usage behavior is
   supported by calibration cases from different learners.

9. Preserve the learner IDs and calibration interaction IDs that
   support each identified problem.

OUTPUT
For each recurring Skill-usage error, return:
- problem_description
- affected_skill_usage
- supporting_learners
- supporting_interactions
- evidence_summary
```

Different mixed-learner batches may identify semantically equivalent Skill-usage errors. Before protocol editing, these results are consolidated so that the same underlying executor-usage problem is represented once while all supporting learners and calibration interactions are preserved.

The consolidation step uses the following prompt:

```
============================================================
 CROSS-LEARNER SKILL-USAGE ERROR CONSOLIDATION
============================================================

ROLE
Consolidate recurring Skill-usage errors identified from different
mixed-learner calibration batches.

INPUT
{{IDENTIFIED_ERRORS}}

Each identified error contains:
- problem_description
- affected_skill_usage
- supporting_learners
- supporting_interactions
```

```
- evidence_summary

GOAL
Produce the recurring executor Skill-usage problems that are
supported across different learners.

RULES
1. Merge two identified errors only when they describe the same
   underlying problem in how the executor uses the Simulation Skill.

2. Preserve all supporting learner IDs, calibration interaction IDs,
   and evidence.

3. Retain only problems supported by calibration evidence from
   different learners.

4. Do not interpret a shared learner behavior as a shared
   executor-usage problem.

5. Do not promote learner-specific behavior, task-specific answers,
   or isolated simulation mismatches into the shared execution
    protocol.

6. Preserve the distinction between information already provided
   before simulation and additional evidence requested by the executor
    .

7. Do not introduce new evidence that is absent from the supplied
   calibration cases.

OUTPUT
For each consolidated recurring Skill-usage error, return:
- problem_description
- affected_skill_usage
- supporting_learners
- supporting_interactions
- evidence_summary
```

The consolidated cross-learner Skill-usage errors constitute $\mathcal{F}^{(k)}$ in Section 3.2. Before protocol editing, the supporting learner IDs and calibration interaction IDs are checked against the original calibration data. This ensures that each problem retained in $\mathcal{F}^{(k)}$ is grounded in recurring execution behavior observed across different learners rather than in an isolated example.

E.3 Editing the Execution Protocol

Given the recurring Skill-usage errors $\mathcal{F}^{(k)}$, the current execution protocol is edited according to Eq. 8:

$$\tilde{\pi}^{(k+1)} = \text{Edit}\left(\pi^{(k)}, \mathcal{F}^{(k)}\right), \qquad |\tilde{\pi}^{(k+1)}|_{\text{tok}} \leq B_{\pi}. \tag{59}$$

Only the execution protocol in `SKILL.md` is modified. The learner's learning state, response patterns, the default current-task information, accessible real historical evidence, associated Skill resources, and executor model parameters remain unchanged. Protocol editing therefore changes how the current executor uses information already available to it and how it decides whether additional evidence is useful, rather than introducing new learner-specific information.

The editing step uses the following prompt:

```
============================================================
 EXECUTION-PROTOCOL EDITING
```

```
============================================================

ROLE
Calibrate the shared execution protocol of Learner2Skill to the
current fixed executor.

CURRENT EXECUTION PROTOCOL
{{CURRENT_PROTOCOL}}

RECURRING SKILL-USAGE ERRORS
{{RECURRING_ERRORS}}

Each error is supported by calibration evidence from different
    learners.

GOAL
Revise the shared execution protocol so that the current executor
can use the existing learner information, the current-task
information already provided, and any additionally requested
evidence more effectively.

EDITING RULES
1. Modify ONLY the execution protocol.

2. Make the smallest generalizable change needed to address the
   recurring Skill-usage errors.

3. The revision may clarify how the executor should:
   - interpret the persistent learner information;
   - interpret the current-task information already provided;
   - decide when additional cognitive information or historical
     evidence is useful;
   - issue further cognitive-model queries or retrieval requests;
   - combine the information already provided with additionally
     obtained evidence;
   - reflect the available learner evidence in the simulated response.

4. Preserve the default provision of current-task information.
   Do NOT turn the relevant learning-state dimensions, task difficulty
    ,
   or predicted correctness probability into optional resource calls.

5. Preserve adaptive ADDITIONAL resource use.
   Do not convert further cognitive-model queries or retrieval into
   a fixed sequence of tool calls.

6. Do not introduce:
   - learner-specific facts;
   - real calibration responses;
   - answers to individual learning tasks;
   - new response patterns;
   - new learning states;
   - new tools;
   - new external information.

7. Do not encode individual calibration examples as protocol rules.

8. Preserve existing protocol instructions that are unrelated to
   the identified recurring errors.

9. Keep the complete revised protocol within the specified
   protocol-length budget.
```

```
OUTPUT
Return only the complete candidate execution protocol.
```

The length constraint B_π prevents repeated calibration rounds from continually expanding the shared protocol with calibration examples or task-specific rules.

Importantly, protocol editing preserves the execution mechanism defined during Skill Construction. For every task, the executor first receives the persistent learner information together with the default current-task information. It then decides whether this information is sufficient and, if necessary, whether further cognitive-model queries or historical retrieval are useful. Calibration changes how these decisions are guided rather than removing the default current-task information or replacing adaptive additional resource use with a fixed tool-use pipeline.

E.4 Validating Protocol Updates

A candidate protocol derived from $\mathcal{U}_{\mathrm{fit}}$ is not accepted using the same learners that contributed to its recurring Skill-usage errors. Instead, it is compared with the current protocol on learners in $\mathcal{U}_{\mathrm{val}}$, which are not involved in error identification or protocol editing.

For every learner in $\mathcal{U}_{\mathrm{val}}$, the current protocol $\pi^{(k)}$ and the candidate protocol $\tilde{\pi}^{(k+1)}$ are evaluated using the same Simulation Skill, the same calibration interactions, the same default current-task information, the same accessible Skill resources and historical evidence, the same executor M, and the same simulation evaluation criterion. The execution protocol is the only component that differs between the paired runs. The two protocols may choose different additional cognitive-model queries or retrieval actions as a consequence of their different execution instructions, but both begin from exactly the same information and resource availability.

The performance difference between the candidate and current protocols is first aggregated within each validation learner, producing one paired improvement for that learner. Validation is then performed over these learner-level paired improvements using a one-sided paired bootstrap. Learners, rather than individual calibration interactions, are therefore the statistical units of the bootstrap.

For each bootstrap resample, validation learners are sampled with replacement, and the mean paired improvement of the resampled learners is computed. The resulting bootstrap distribution is used to obtain the one-sided confidence bound specified in the experimental settings.

A candidate protocol is accepted only when the lower confidence bound of the paired improvement is greater than zero. When this condition is satisfied,

$$\pi^{(k+1)} = \tilde{\pi}^{(k+1)}. \tag{60}$$

Otherwise, the current protocol $\pi^{(k)}$ is retained.

After an accepted update, the updated protocol is used again on $\mathcal{U}_{\mathrm{fit}}$ to identify any remaining recurring Skill-usage errors. Error identification, protocol editing, and independent validation are repeated until no subsequent candidate is accepted or the maximum number of calibration rounds R_{cal} is reached.

The resulting executor-specific shared execution protocol is denoted by

$$\pi^{\star}_{M}, \tag{61}$$

and is subsequently used by all learner Skills under executor M.

E.5 Synchronizing Skills Before Final Testing

During Executor Calibration, the learning state, response patterns, and real historical evidence accessible through each learner Skill remain unchanged. The default current-task information is generated from the same fixed cognitive model and the same learner-information snapshot for each paired protocol comparison. Although the real calibration responses are observed as feedback, they are not immediately incorporated into the learner's real interaction history. This ensures that the current and candidate protocols are compared under the same learner information.

After the final $\pi^\star_M$ is selected, the real interactions observed during calibration are incorporated into the corresponding learners' real interaction histories in their original chronological order. Each learner Skill is then synchronized using the Skill Evolution procedure in Section 3.3: the learning state is updated from the expanded real history, and the response patterns are updated according to the accumulated real behavioral evidence. The calibrated protocol $\pi^\star_M$ remains fixed throughout synchronization.

The synchronized Skill therefore contains the learner information available immediately before final testing, while the interaction database contains the corresponding complete real history. For each subsequent test task, the current-task information is generated from this synchronized learner state using the same cognitive model before the executor begins simulation. Calibration interactions are excluded from the final evaluation, and simulated responses are never written back as real learner evidence.

When the downstream executor changes from M to another executor M', the constructed learner Skills and the real interaction histories in the database are retained. The same low-cost calibration procedure is applied to the new executor to obtain $\pi^\star_{M'}$, allowing the learner-specific simulation capability constructed in Stage 1 to be reused without reconstructing the learner Skill from the complete interaction history.

F Test-Time Skill Evolution

Test-time Skill Evolution operates during online simulation after Executor Calibration. As new real interactions arrive, Learner2Skill updates the learner-specific information maintained in the Simulation Skill while keeping the calibrated execution protocol $\pi^\star_M$, the executor M, and the underlying model parameters fixed.

The update follows the two forms of learner-specific information defined in Section 3.1. The current learning state $\mathbf{z}^t_u$ is adapted to recent real outcomes, while the response patterns $\mathcal{B}^t_u$ are maintained according to accumulated real behavioral evidence. Every newly observed real interaction is examined as potential evidence for the response patterns, although the stored pattern set changes only when the accumulated evidence warrants an update. In all cases, the current real interaction is incorporated only after its prediction has been completed.

F.1 Online Simulation and Skill Update

For the t-th learning task e_t, simulation starts from the Skill available before the current real interaction, $\mathcal{S}^{t-1}_u$. The executor first generates

$$(\hat{r}_t, \hat{y}_t) \sim q_{M,\pi^\star_M}\left(r, y \mid e_t, \mathcal{S}^{t-1}_u\right), \tag{62}$$

as defined in Eq. 9.

Only after this prediction has been completed are the real response r_t and outcome y_t observed. The resulting real interaction (e_t, r_t, y_t) is then appended to the learner's real interaction history, yielding

$$\mathcal{H}^t_u = \mathcal{H}^{t-1}_u \cup \{(e_t, r_t, y_t)\}. \tag{63}$$

The newly observed real evidence is subsequently used to update the two learner-specific components of the Skill. The current learning state is updated as described in Appendix F.2. The same new real interaction is also evaluated against the accumulated evidence underlying the response patterns as described in Appendix F.3. The latest $\mathbf{z}^t_u$ and $\mathcal{B}^t_u$ are then stored in the Simulation Skill $\mathcal{S}^t_u$, while the expanded real interaction history $\mathcal{H}^t_u$ remains in the interaction database and is accessible through retrieval.

The resulting Skill is used for the next interaction. Current-task information for a subsequent task is generated from this latest learning state according to Appendix D.3.

This ordering ensures that the current real response never contributes to its own prediction. Moreover, only real learner interactions are used to update the Skill; simulated responses are never treated as learner evidence.

F.2 Updating the Current Learning State

After the real interaction (e_t, r_t, y_t) has been added to $\mathcal{H}_u^t$, the frozen cognitive model F_θ produces the base estimate $\tilde{\mathbf{z}}_u^t$ from the updated real history. Test-time Skill Evolution does not modify the parameters of F_θ. Instead, it estimates a learner-specific residual from recent real outcomes and applies this residual to the frozen-model estimate.

As defined in the main text, the recent interaction window is

$$\mathcal{W}_t = \{\max(1, t - L + 1), \ldots, t\}, \tag{64}$$

where L determines the number of recent real interactions used for the update. For each $j \in \mathcal{W}_t$, $\tilde{\mathbf{z}}_u^{j-1}$ denotes the base estimate produced by the frozen cognitive model from the real interaction history available before interaction j.

For a candidate correction $\mathbf{d}$, the recent-observation loss is

$$\mathcal{L}_{\text{obs}}^t(\mathbf{d}) = -\sum_{j \in \mathcal{W}_t} w_{t,j} \log P_\theta\left(y_j \mid e_j, \Pi_{[0,1]^K}\left(\tilde{\mathbf{z}}_u^{j-1} + \mathbf{d}\right)\right). \tag{65}$$

We assign greater weight to more recent observations using exponential recency weighting. Let h denote the half-life. We first define

$$\bar{w}_{t,j} = 2^{-\frac{t-j}{h}}, \tag{66}$$

and normalize the weights within the current window:

$$w_{t,j} = \frac{\bar{w}_{t,j}}{\sum_{\ell \in \mathcal{W}_t} \bar{w}_{t,\ell}}. \tag{67}$$

Thus, an observation h interactions older receives half the unnormalized weight of the most recent interaction.

The learner-specific residual is obtained by

$$\boldsymbol{\delta}_u^t = \arg\min_{\mathbf{d}} \left[\mathcal{L}_{\text{obs}}^t(\mathbf{d}) + \beta\|\mathbf{d}\|_2^2 + \gamma\|\mathbf{d} - \boldsymbol{\delta}_u^{t-1}\|_2^2\right]. \tag{68}$$

The first regularization term limits the magnitude of the correction, while the second discourages abrupt changes relative to the previous residual.

The resulting current learning state is

$$\mathbf{z}_u^t = \Pi_{[0,1]^K}\left(\tilde{\mathbf{z}}_u^t + \boldsymbol{\delta}_u^t\right), \tag{69}$$

as in Eq. 10. The projection keeps each concept-level state within the range $[0, 1]$ defined in Section 3.1. The updated $\mathbf{z}_u^t$ is then stored in the Simulation Skill and used by subsequent simulations.

In our implementation, we use the most recent

$$L = 8$$

real interactions and set the recency half-life to

$$h = 4.$$

The regularization coefficients are

$$\beta = 0.05, \qquad \gamma = 0.20.$$

For every new real interaction, the learner-specific residual is optimized using Adam with learning rate 0.05 for 20 optimization steps. All parameters of F_θ remain frozen throughout this optimization.

The residual update is further restricted to knowledge concepts appearing in the recent interaction window $\mathcal{W}_t$. Dimensions corresponding to concepts without recent real evidence are not optimized by the current residual update. The resulting correction therefore adapts the learning state only where recent learner behavior provides direct evidence.

F.3 UPDATING RESPONSE PATTERNS

Response patterns $\mathcal{B}_u^t$ are maintained through an evidence-driven adaptive update. Unlike the learning state, the pattern set is not expected to change after every interaction. However, every newly observed real interaction is examined immediately after it is added to the interaction database to determine how it changes the evidence for the current patterns.

For the newly observed interaction (e_t, r_t, y_t), the maintenance procedure determines whether it:

- provides additional support for an existing response pattern;
- conflicts with an existing response pattern under a comparable condition;
- provides evidence for a new candidate response pattern; or
- does not provide evidence requiring a change to the current response patterns.

This distinction separates *evidence assessment*, which occurs for every new real interaction, from *pattern retention*, which depends on the accumulated evidence across multiple interactions. Consequently, a single new interaction may update the backend evidence associated with a pattern without changing the response patterns exposed to the executor.

The semantic evidence assessment uses the following prompt:

```
# Response-Pattern Evidence Update

You are maintaining the response patterns of one learner using a newly
observed REAL interaction.

## Current Response Patterns
{{CURRENT_PATTERNS}}

## New Real Interaction
{{NEW_INTERACTION}}

## Existing Pattern Evidence
{{PATTERN_EVIDENCE}}

## Relevant Real Historical Evidence
{{RELEVANT_HISTORY}}

## Goal
Determine how the new real interaction changes the evidence for the
learner's response patterns.

## Rules
1. Use only REAL learner interactions as evidence.

2. For each existing response pattern that is relevant to the new
   interaction, determine whether the new interaction:
   - SUPPORTS the pattern;
   - CONFLICTS with the pattern under a comparable condition; or
   - is IRRELEVANT to the pattern.

3. If the new interaction suggests response behavior that is not
   represented by an existing pattern, it may be recorded as evidence
   for a NEW CANDIDATE pattern.

4. A behavior observed in only this new interaction must remain a
   candidate. Do NOT promote a single observation directly to a
    retained
   response pattern.
```

```
5. If the new evidence suggests that the condition or response
    behavior
   of an existing pattern should be refined, return the refinement as
    a
   CANDIDATE REVISION rather than directly rewriting the retained
    pattern.
   The revision must be supported by accumulated real evidence before
    it
   can replace the current representation.

6. Preserve the ID of every real interaction used as evidence.
   Do not invent interaction IDs.

7. Do not infer unobserved psychological traits, motivations,
    abilities,
   or other learner characteristics.

8. Do not compute support counts, conflict counts, support ratios, or
   make the final pattern-retention decision. These operations are
   performed programmatically.

## Output

For existing patterns, return:
- pattern_id
- evidence_relation: SUPPORT | CONFLICT | IRRELEVANT
- evidence_interaction_ids

For each new candidate, return:
- condition
- response_behavior
- support_ids

For each candidate revision, return:
- source_pattern_id
- revised_condition
- revised_response_behavior
- support_ids
```

The LLM is used only for the semantic comparison between the new real interaction and the response-pattern descriptions. It does not decide whether a pattern is retained. After the semantic assessment, all returned interaction IDs are checked against the real interaction database before the corresponding backend evidence is updated.

For response pattern j, let $n^+_{u,j}$ and $n^-_{u,j}$ denote the numbers of distinct real interactions that respectively support and conflict with the pattern under comparable conditions. As in Skill Construction, its support ratio is

$$\rho_{u,j} = \frac{n^+_{u,j}}{n^+_{u,j} + n^-_{u,j}}, \tag{70}$$

following Eq. 20.

The same retention criterion used during Skill Construction is applied:

$$n^+_{u,j} \geq m_{\text{sup}}, \qquad \rho_{u,j} \geq \eta_{\text{sup}}. \tag{71}$$

Thus, no additional retention threshold is introduced for Skill Evolution.

For an existing response pattern, the new interaction first updates its supporting or conflicting evidence. The pattern remains in $\mathcal{B}^t_u$ while its accumulated evidence continues to satisfy the retention criterion; otherwise, it is removed from the maintained pattern set.

A newly proposed behavior initially remains a candidate. Candidate patterns obtained at different time steps are consolidated using the same semantic equivalence rule as in Appendix B.2. A candidate

enters $\mathcal{B}_u^t$ only after its accumulated real evidence satisfies the same support and stability requirements above.

A proposed revision of an existing pattern is handled in the same way rather than being applied directly from a single interaction. Its revised condition and response behavior are treated as a candidate representation and must accumulate sufficient real evidence before the stored response pattern is revised. This prevents transient behavior in a newly observed interaction from immediately rewriting a previously supported pattern.

After evidence updating, consolidation, and retention, the maintained response patterns continue to satisfy the pattern budget

$$|\mathcal{B}_u^t| \le J_{\max}, \tag{72}$$

using the same budget defined during Skill Construction.

The supporting and conflicting interaction IDs, candidate patterns, and their evidence statistics are maintained only as backend information for pattern maintenance. The executor receives only the currently retained $(c_{u,j}, \tau_{u,j})$ pairs in $\mathcal{B}_u^t$, consistent with Appendix B.2.

This procedure means that response-pattern maintenance has no fixed update interval. Every newly observed real interaction contributes immediately to the evidence-maintenance process, while the response patterns themselves are retained, revised, added, or removed only when the accumulated real evidence supports the corresponding change.

The two learner-specific components therefore evolve at different granularities. The current learning state $\mathbf{z}_u^t$ is numerically adapted after each newly observed real outcome, whereas $\mathcal{B}_u^t$ changes only when accumulated behavioral evidence warrants a pattern-level update. After each interaction, the latest state and response patterns are stored in the Simulation Skill, while the expanded real history remains in the interaction database. Throughout this process, $\pi_M^\star$, the executor M, and all underlying model parameters remain fixed.

G Theoretical Analysis of Cross-Executor Token Cost

Reusing learner Skills saves construction work, but whether it reduces total token consumption also depends on the costs of calibrating and running each executor. We derive an exact comparison with an existing LLM-based learner-simulation method using the efficiency metrics defined in Appendix H.4.

Cost definition. Let C_{acq} denote Capability Acquisition Cost, the total generative tokens consumed before final testing, and let C_{test} denote Test-Time Cost, the mean generative tokens consumed per test interaction. For $N_{\text{test}} > 0$ evaluated interactions, the total token cost C_{total} is

$$C_{\text{total}} = C_{\text{acq}} + N_{\text{test}} C_{\text{test}}. \tag{73}$$

All costs count prompt and completion tokens. Test-Time Cost already includes method-specific online maintenance, so that cost is not added again. Independent evaluation tokens are excluded for both methods, consistently with the evaluation protocol.

Learner2Skill across executors. Consider $m \ge 1$ executor configurations, indexed by $v \in \{1, \dots, m\}$, that start from the same constructed learner Skills at the same history boundary. Let B be the one-time construction cost of the complete Skill collection required by these workloads, including Skills used for calibration. For executor v, let A_v be its remaining acquisition cost, covering Executor Calibration and any generative pre-test synchronization operations. Shared-protocol calibration calls are counted once per actual execution; learner-specific synchronization calls are also included where they occur.

Let $N_v > 0$ be the number of test interactions for executor v, and let $T_{v,i}^{\text{test}}$ be the tokens consumed while processing its i-th test interaction, including online maintenance. Its mean test cost is $C_{\text{test},v} = N_v^{-1} \sum_{i=1}^{N_v} T_{v,i}^{\text{test}}$. Define O_v as its total post-construction cost:

$$O_v = A_v + \sum_{i=1}^{N_v} T_{v,i}^{\text{test}} = A_v + N_v C_{\text{test},v}. \tag{74}$$

Learner2Skill's total across all m executors, denoted by $C_{\mathrm{L2S}}^{(m)}$, is therefore

$$C_{\mathrm{L2S}}^{(m)} = B + \sum_{v=1}^{m} O_v. \tag{75}$$

Here the subscript L2S denotes Learner2Skill and the superscript (m) indicates a total over m executor configurations, not an exponent. Construction is counted once, while each executor retains its own calibration and subsequent evolution costs. For one executor, this agrees with Eq. 73 with $C_{\mathrm{acq}} = B + A_1$.

Comparison method. Let b identify an existing LLM-based learner-simulation method evaluated on the same learner–task workloads. Within each executor comparison, both methods use the same underlying executor, generation settings, and permitted historical evidence. Let $B_b^{(m)}$ be method b's actual total learner-representation construction or reconstruction cost, and $O_{b,v}$ its remaining acquisition and test cost for executor v. Its total is

$$C_b^{(m)} = B_b^{(m)} + \sum_{v=1}^{m} O_{b,v}. \tag{76}$$

If method b also reuses a profile or memory, its shared construction is counted once. If it has no construction stage, that term is zero and its calls are charged to the stages where they occur. Other shared operations are assigned once across the executor totals. Each included API call must belong to exactly one cost term for either method.

proposition

[. Condition for lower total token cost] Let G_m be Learner2Skill's construction-cost saving relative to method b, and let E_m be its additional post-construction cost:

$$G_m = B_b^{(m)} - B, \qquad E_m = \sum_{v=1}^{m} (O_v - O_{b,v}). \tag{77}$$

Then the total token saving is exactly

$$C_b^{(m)} - C_{\mathrm{L2S}}^{(m)} = G_m - E_m. \tag{78}$$

Consequently, Learner2Skill uses fewer total tokens on these workloads if and only if $G_m > E_m$.

Proof. Subtracting Eq. 75 from Eq. 76 gives

$$\begin{aligned} C_b^{(m)} - C_{\mathrm{L2S}}^{(m)} &= B_b^{(m)} - B + \sum_{v=1}^{m} (O_{b,v} - O_v) \\ &= G_m - E_m. \end{aligned}$$

Learner2Skill has a smaller total cost precisely when this difference is positive, which is equivalent to $G_m > E_m$. □

Implication for Skill reuse. The condition states that construction savings must exceed any additional calibration, simulation, and maintenance costs. A positive G_m means lower construction cost for Learner2Skill, whereas a positive E_m means higher post-construction cost; either difference can also be zero or negative. If a comparison method rebuilds a representation costing B for every executor, then $G_m = (m-1)B$. If it instead shares one representation with the same construction cost, then $G_m = 0$, and Learner2Skill has a total advantage only when $E_m < 0$. These are conditional cases, not assumptions that the evaluated baselines must rebuild their learner representations.

Calculation and scope. Compute B, A_v, and the corresponding baseline costs from stage-resolved API logs; obtain each test total by summing $T_{v,i}^{\mathrm{test}}$ or multiplying the unrounded mean $C_{\mathrm{test},v}$ by N_v. Acquisition totals alone do not identify the shared construction component. For repeated experiments, calculate paired totals within each repetition before averaging. No likelihood estimates or additional model calls are needed for this calculation.

The result gives a condition for lower total tokens on the specified workloads. Whether that condition holds is an empirical question, and response fidelity is evaluated separately. It does not guarantee lower per-interaction cost or better fidelity, nor does it attribute a cost difference solely to reuse when other construction and execution choices differ. The analysis changes no part of the method and requires no fixed bound on adaptive tool-call counts.

H Experimental Details

H.1 Dataset

H.1.1 Data Description and Statistics

LearnerTrace-2K is collected from authentic mathematics and physics practice on an online learning platform for high-school students in China. It contains records from 2,000 learners and covers both multiple-choice and application questions. For each practice interaction, the platform records the exercise, its associated knowledge concepts, the learner's submitted final answer, and the corresponding score. In addition, a dedicated input area prompts learners to record the main ideas and intermediate steps involved in their problem-solving process. LearnerTrace-2K therefore preserves not only final outcomes but also learner-authored records of how each problem was approached.

During practice, learners are presented with the following instruction:

```
## Problem-Solving Process Recording ##

Please record the main ideas and key steps you used to solve this
    problem.

You may write down the formulas, reasoning, or calculations you used,
    as well as any
revisions made along the way. If you did not finish the problem, you
    may simply record
the parts you had already thought through or completed. Please
    describe your actual
solving process; there is no need to rewrite it as a standard solution
    .
```

Learners are not required to provide a complete or standardized solution. Consequently, the recorded processes naturally contain diverse forms of learner behavior, including complete solutions, partial attempts, intermediate calculations, mistakes, revisions, and stopping points. Figure 4 illustrates the practice interface used to collect these records. Platform-identifying information is hidden for double-blind review.

The original exercises and learner-written records are in Chinese. For the experiments, we translate the textual content into English while preserving the original problem meaning, mathematical expressions, learner responses, and behaviorally relevant information.

LearnerTrace-2K contains **2,000 learners, 52,566 learner–exercise interactions, 2,076 exercises, and 709 knowledge concepts**. All interactions retain their original temporal order.

For each learner, we chronologically divide the interaction trajectory into three consecutive portions: **60% for Skill Construction**, **10% for Executor Calibration**, and **30% for final evaluation with Skill Evolution**. The split is performed independently for each learner while strictly preserving the original interaction order. Accordingly, no interaction from the final 30% test portion is used for Skill Construction, Executor Calibration, cognitive-model training, or model selection.

Training and checkpoint selection of the cognitive model are further restricted to the first 60% Skill Construction portion. Within this portion, we perform an additional chronological train/validation split solely for cognitive-model training and checkpoint selection. This internal split does not alter the overall 60%/10%/30% experimental protocol. Further details of the cognitive-model training procedure are provided in Appendix D.2.

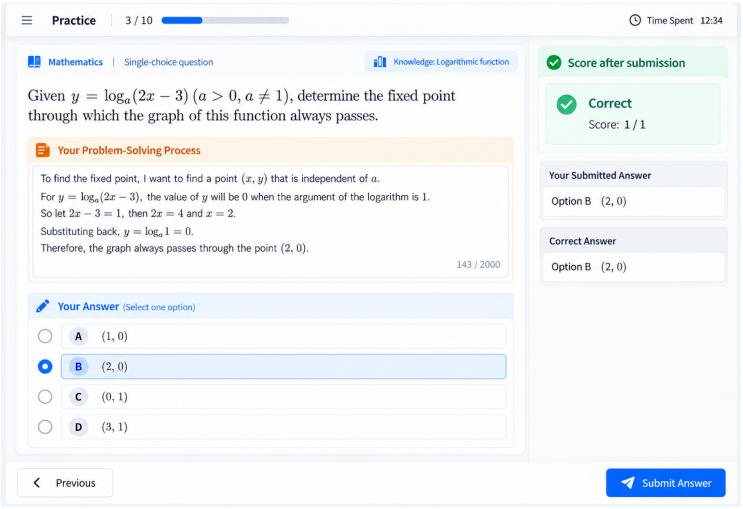

Figure 4: Illustration of the exercise interface used to collect learner-authored problem-solving processes. In addition to submitting a final answer, learners are prompted to record their main reasoning, calculations, intermediate steps, and possible revisions during problem solving. Platform-identifying information is hidden for double-blind review.

H.2 Baseline Implementations

All LLM-based methods are evaluated under the same learner-simulation task. For the main comparison in Table 1, they use **Claude Sonnet 4.6** as the simulation executor and produce the same four response fields: whether the learner attempts the exercise, the simulated problem-solving process, the final answer, and the predicted outcome. At each test interaction, only real learner information observed before the current step is available, and the final 30% of each trajectory is simulated sequentially.

H.2.1 LLM-Based Learner Simulators

LLM-FullHistory. **LLM-FullHistory** is our prompt-based baseline for measuring the learner-simulation capability of the underlying LLM without an additional learner-modeling mechanism. For each test exercise, a unified prompt provides the executor with the current exercise and the complete real learner history available up to that point, and asks it to directly simulate the learner's behavior on the current exercise. The prompt requires four output fields: attempted behavior, problem-solving process, final answer, and predicted outcome. As testing proceeds, newly observed real interactions are appended to the history available for subsequent simulations. LLM-FullHistory therefore performs step-by-step learner simulation directly through the growing historical context, without constructing an additional learner profile, memory, or Simulation Skill.

EduAgent. **EduAgent** incorporates cognitive-science priors to characterize relationships among learner characteristics, learning behaviors, and learning outcomes, and uses this information to construct a student agent for subsequent behavior simulation (Xu et al., 2024). Because LearnerTrace-2K does not directly provide all cognitive information required by the original setting, we automatically construct the required cognitive prior from the real learner history available before testing. Cognitive states are obtained using the same cognitive-modeling procedure used by Learner2Skill, while the remaining available learner information is extracted from historical interactions. The resulting prior is stored before simulation and reused throughout testing. Apart from dataset-specific input construction and conversion to the common output format, we retain EduAgent's learner-modeling and behavior-simulation procedure.

Agent4Edu. **Agent4Edu** constructs personalized learner agents through learner-profile, memory, and action-generation components (Gao et al., 2025). We initialize each learner agent using the real learner interactions available before testing and retain its learner-profile construction and memory-

maintenance mechanisms. During sequential evaluation, after the current interaction has been simulated, the newly observed real learner interaction is incorporated into its memory and becomes available for subsequent simulations. The generated behavior is converted to the common four-field output format.

CoderAgent. **CoderAgent** was originally developed to simulate fine-grained problem-solving processes in programming tasks and organizes process-level simulation using cognitive states and the Programming Tree of Thought (PTOT) (Zhan et al., 2025). Since LearnerTrace-2K contains mathematics and physics exercises, we retain its cognitive-state-guided, step-by-step problem-solving simulation mechanism while replacing programming-specific information with corresponding general problem-solving information. Mathematics and physics exercises serve as the current tasks, learner-recorded problem-solving processes provide historical process evidence, and intermediate reasoning and calculation steps serve as process-level behaviors. CoderAgent can therefore follow its original step-by-step simulation mechanism to generate learner problem-solving behavior for mathematics and physics tasks. Its generated response is then converted to the common output format.

Ability-Aware Simulation (AAS). **Ability-Aware Simulation (AAS)** uses cognitive diagnosis to obtain learner ability representations and leverages ability information to guide subsequent learner simulation (Que et al., 2026). The original method further performs ability-aware matching among candidate LLMs according to their cognitive profiles. Since our main comparison uses the same Claude Sonnet 4.6 executor for all LLM-based methods, we retain AAS's cognitive diagnosis and ability-aware conditioning while using the common executor for response generation. The diagnosed learner ability information is provided to the executor to guide simulation of the corresponding learner's subsequent behavior. The generated response is finally converted to the common four-field format.

H.2.2 Supervised Learner-Performance Predictors

We additionally include four supervised learner-performance prediction methods: **KES** (Liu et al., 2019), **DKVMN** (Zhang et al., 2017), **SAKT** (Pandey & Karypis, 2019), and **DAISim** (Zhao et al., 2023). KES models the evolution of learner knowledge for subsequent performance prediction (Liu et al., 2019); DKVMN uses dynamic key–value memory to represent evolving learner knowledge states (Zhang et al., 2017); SAKT uses self-attention to identify historical interactions relevant to the current exercise (Pandey & Karypis, 2019); and DAISim learns a student simulator from observed student interactions (Zhao et al., 2023).

These methods are trained on the first 60% of each learner trajectory and selected using the following 10%, with final evaluation performed on the same chronological 30% test portion used by the remaining methods. A threshold of 0.5 is used to convert predicted correctness probabilities into binary outcomes.

Because these methods predict learner performance rather than generate complete learner responses, they are evaluated only by **Outcome Prediction Accuracy**. Metrics requiring generated attempt behavior, problem-solving processes, final answers, or prediction–response consistency are not applicable.

H.3 Implementation Details

Stage Settings. Following the chronological split described above, Learner2Skill uses the first 60%, subsequent 10%, and final 30% of each learner's real interactions for **Skill Construction**, **Executor Calibration**, and final evaluation with **Skill Evolution**, respectively.

During **Skill Construction**, we use **Claude Sonnet 4.6** as the offline Skill builder to construct learner-specific Simulation Skills from the first 60% of real interaction histories. To keep each Skill compact, we retain at most **5 response patterns** for each learner. A candidate pattern is retained only when it is repeatedly supported by at least **10 real interactions**. The complete real interaction history is maintained in an external interaction database and accessed through retrieval when needed, rather than being stored directly in the Simulation Skill.

During **Executor Calibration**, the subsequent 10% of real interactions serve as calibration data. Learners are randomly divided into two disjoint sets with a **70%/30%** split for protocol editing and protocol validation, respectively. Calibration runs for at most **3 rounds**, and the shared execution protocol is constrained to at most **1,024 tokens**. Each candidate protocol is compared with the current protocol on the held-out learners using learner-level paired comparisons. We use **1,000 paired bootstrap resamples** to construct a **95% one-sided confidence interval** for the improvement. A candidate protocol is accepted only when the lower confidence bound of the paired improvement is greater than zero; calibration terminates early if no candidate is accepted in the current round. The resulting executor-specific protocol remains fixed during subsequent testing. After calibration, the real interactions observed during this stage are incorporated into the corresponding learner histories in their original chronological order, and the Simulation Skills are synchronized before final testing.

During **Skill Evolution**, the final 30% of real interactions are processed strictly in their original chronological order. For each test task, Learner2Skill first performs simulation using the Simulation Skill available before the current real interaction. The real response and outcome are observed only after prediction and are then used to update the Skill. The learning-state update estimates the learner-specific residual correction from the most recent **10 real interactions** on top of the latest estimate produced by the frozen cognitive model. Response patterns are updated as new real evidence becomes available, using the same retention criterion as in Skill Construction: a pattern requires repeated support from at least **10 real interactions**, and at most **5 response patterns** are retained. Simulated responses are never used as evidence for Skill updates. Throughout final testing, the simulation executor, calibrated execution protocol, and underlying model parameters remain fixed.

Model and Executor Configuration. For the main comparison in RQ1, Learner2Skill and all LLM-based baselines use **Claude Sonnet 4.6** as the simulation executor, with matched generation settings and the same structured learner-response format.

For the cross-executor experiment in RQ2, we fix the learner-specific Simulation Skills constructed by **Claude Sonnet 4.6** and reuse them with four downstream simulation executors: **Claude Sonnet 4.6**, **Gemini 3.8 Flash**, **Qwen3.8 Flash**, and **DeepSeek V4 Flash**. Changing the executor does not repeat Skill Construction. Instead, Executor Calibration is performed independently for each new executor, after which the previously constructed learner-specific Simulation Skills are directly reused.

Generation and Evaluation Settings. Unless otherwise specified, all generative LLM calls use a temperature of 0.7 and top-p of 0.8. Methods within the same comparison use matched generation settings. All semantic judgments required by the evaluation are performed by an independent **GPT-5.5**. GPT-5.5 serves only as the evaluator and does not participate in Skill Construction, Executor Calibration, learner-response generation, protocol editing, or Skill Evolution. Detailed evaluation criteria and evaluator prompts are provided in Appendix H.4.

Repeated Runs and Computing Environment. All experiments are independently repeated **three times** under the same experimental setting, and results are aggregated across runs. All LLM-based simulation methods are accessed through their corresponding model APIs, while components requiring local training are run on a single **NVIDIA A100 80GB GPU**.

Token Accounting. For every generative API call, prompt and completion token counts are recorded separately and used to measure the generative cost at different stages. Generative tokens consumed before final testing to acquire learner-specific simulation capability are counted as **Capability Acquisition Cost**, whereas tokens consumed by learner simulation and Skill Evolution during final testing are counted as **Test-Time Cost**. Their sum constitutes the **Total Cost** and is further used to compute **Consistency Efficiency (CE)**. Tokens consumed by the independent GPT-5.5 evaluator are excluded from the reported simulation costs.

The complete implementation, experimental configurations, and running scripts are available in our anonymous repository: https://anonymous.4open.science/r/Learner2Skill/.

H.4 Evaluation Protocol

We evaluate learner simulation from two complementary perspectives: **effectiveness**, which measures how faithfully a simulator reproduces the behavior of a real learner, and **efficiency**, which measures the generative cost required to acquire and subsequently use learner-specific simulation capability.

For a real learner interaction, let

$$(a, p, f, y) \tag{79}$$

denote the learner's attempt decision, problem-solving process, final answer, and observed outcome, respectively. The corresponding simulated quantities are

$$(\hat{a}, \hat{p}, \hat{f}, \hat{y}), \tag{80}$$

where $\hat{y}$ denotes the simulator's explicit prediction of the learner's outcome.

We additionally evaluate the outcome actually realized by the generated final answer:

$$\tilde{y} = g_e(\hat{f}), \tag{81}$$

where $g_e(\cdot)$ denotes the answer evaluator associated with exercise e. Thus, $\hat{y}$ represents what the simulator predicts the learner will achieve, whereas $\tilde{y}$ represents the outcome actually produced by its generated answer. Distinguishing these quantities allows us to separately evaluate whether the simulator predicts the learner correctly, reproduces the learner's actual outcome, and generates a response consistent with its own prediction.

H.4.1 Simulation Effectiveness

We evaluate simulation effectiveness at multiple levels of learner behavior, covering whether the learner attempts the task, how the response unfolds, what final answer is produced, and what outcome is ultimately realized.

Attempt Fidelity. Attempt Fidelity measures whether the simulator reproduces the learner's decision to attempt the exercise:

$$Q_{\text{attempt}} = \mathbb{1}[\hat{a} = a], \tag{82}$$

where $\mathbb{1}[\cdot]$ is the indicator function. The metric equals 1 when the simulated and real learners exhibit the same attempt behavior and 0 otherwise.

Process Fidelity. Process Fidelity measures how closely the simulated problem-solving process $\hat{p}$ matches the real learner process p. Because process alignment generally requires semantic comparison, we use the independent GPT-5.5 evaluator.

The evaluator considers the major problem-solving direction, amount of progress, intermediate mistakes or omissions, correction or verification behavior, and stopping point. It assigns

$$s_{\text{process}} \in \{0, 1, 2\}, \tag{83}$$

where 0, 1, and 2 indicate clearly different, partially aligned, and closely aligned response processes, respectively. We normalize this score to $[0, 1]$:

$$Q_{\text{process}} = \frac{s_{\text{process}}}{2}. \tag{84}$$

The evaluator prompt is instantiated as follows:

```
============================================================
 PROCESS FIDELITY EVALUATION
============================================================

ROLE

Evaluate whether a simulated response process matches the real
response process of the same learner on the same exercise.
```

```
INPUT

Learning task:
{{TASK_CONTENT}}

Real learner response process:
{{REAL_RESPONSE_PROCESS}}

Simulated learner response process:
{{SIMULATED_RESPONSE_PROCESS}}

GOAL

Assess how closely the simulated process reproduces the real
learner's problem-solving behavior.

EVALUATION

Consider:
1. the major problem-solving direction;
2. the amount of progress;
3. intermediate mistakes or omissions;
4. correction or verification behavior;
5. the stopping point.

RULES

1. Evaluate the response PROCESS rather than only the final answer.
2. Do not reward a simulated response merely for being mathematically
   better, more complete, or more sophisticated than the real response
    .
3. Judge alignment with the observed learner behavior, not the quality
   of an ideal solution.
4. Do not infer learner characteristics that are not supported by the
   supplied responses.
5. Ignore the identity of the simulation method.

OUTPUT

Return exactly one score:
- 2: closely aligned;
- 1: partially aligned;
- 0: clearly different.
```

Final-Answer Fidelity. Final-Answer Fidelity measures whether the generated final answer $\hat{f}$ reproduces the learner's actual final answer f:

$$Q_{\text{final}} = \mathbb{1}[\hat{f} \equiv f], \tag{85}$$

where $\equiv$ denotes answer equivalence rather than literal string equality.

For answer formats that can be reliably normalized, including multiple-choice options, scalar values, and standardized strings, equivalence is determined programmatically. For other cases, GPT-5.5 judges semantic equivalence without considering whether either answer is objectively correct.

For semantic cases, the evaluator uses:

```
============================================================
 FINAL-ANSWER FIDELITY EVALUATION
============================================================
```

```
ROLE

Determine whether the simulated final answer represents the same
answer as the real learner's final answer.

INPUT

Learning task:
{{TASK_CONTENT}}

Real learner final answer:
{{REAL_FINAL_ANSWER}}

Simulated learner final answer:
{{SIMULATED_FINAL_ANSWER}}

GOAL

Determine whether the two answers are equivalent as learner
responses.

RULES

1. Evaluate answer equivalence only.
2. Do not judge whether either answer is objectively correct.
3. Treat mathematically or semantically equivalent expressions as
   equivalent even when their surface forms differ.
4. Do not use the response process when making the judgment.
5. Ignore the identity of the simulation method.

OUTPUT

Return exactly one label:
- equivalent
- not_equivalent
```

Realized Correctness Fidelity. Final-Answer Fidelity evaluates whether the simulator reproduces the learner's specific answer. We separately evaluate whether the generated answer produces the same success or failure outcome as the real learner:

$$Q_{\text{correctness}} = \mathbb{1}[\tilde{y} = y]. \tag{86}$$

Here, $\tilde{y}$ is obtained by evaluating the generated final answer, whereas y is the real learner's observed outcome. The metric therefore equals 1 when the generated response reproduces the learner's actual success or failure.

Outcome Prediction Accuracy. The simulator explicitly predicts the learner's outcome $\hat{y}$. We evaluate this prediction as

$$Q_{\text{outcome}} = \mathbb{1}[\hat{y} = y]. \tag{87}$$

This metric evaluates conventional learner-performance prediction and is therefore also applicable to supervised learner models that predict outcomes without generating complete learner responses.

Prediction–Response Consistency. We further evaluate whether the simulator's explicit outcome prediction agrees with the outcome actually realized by its generated answer:

$$Q_{\text{consistency}} = \mathbb{1}[\hat{y} = \tilde{y}]. \tag{88}$$

For example, if the simulator predicts that the learner will answer incorrectly but subsequently generates a correct answer, $Q_{\text{consistency}} = 0$. This metric therefore measures the internal consistency between what the simulator predicts and what it actually generates.

Aggregation. All interaction-level effectiveness metrics are averaged over the corresponding test interactions. For example,

$$\bar{Q}_{\text{consistency}} = \frac{1}{N_{\text{test}}} \sum_{i=1}^{N_{\text{test}}} Q^{(i)}_{\text{consistency}}, \tag{89}$$

where N_{test} denotes the number of evaluated test interactions. The remaining interaction-level metrics are aggregated analogously. For readability, effectiveness metrics are reported as percentages in the experimental tables.

H.4.2 Evaluation Procedure

Independent Semantic Evaluation. All semantic judgments are performed by **GPT-5.5**, which is independent of the LLM executors used for learner simulation. GPT-5.5 does not participate in Skill Construction, learner-response generation, Executor Calibration, protocol editing, or Skill Evolution. Method identities are omitted from evaluator inputs, and the same prompts and scoring criteria are applied to all evaluated methods.

Whenever a judgment can be reliably determined programmatically, such as normalized multiple-choice answers or scalar answers, we use deterministic programmatic evaluation rather than the LLM evaluator. GPT-5.5 is used only when semantic comparison is required.

Human Quality Control. We additionally manually inspect a random subset of Process Fidelity judgments with method identities hidden and verify whether the assigned scores follow the predefined $0/1/2$ rubric.

H.4.3 Calibration Criterion

Executor Calibration requires a scalar criterion for comparing the current execution protocol with a candidate protocol. We define

$$Q_{\text{cal}} = \frac{1}{4}\left(Q_{\text{process}} + Q_{\text{final}} + Q_{\text{correctness}} + Q_{\text{outcome}}\right). \tag{90}$$

The four terms measure process alignment, final-answer alignment, realized correctness, and explicit outcome prediction, respectively. Q_{cal} is used only for protocol comparison during Executor Calibration and is not reported as an overall evaluation metric. Attempt Fidelity is excluded because it provides limited discrimination on LearnerTrace-2K, while Prediction–Response Consistency is retained as a separate diagnostic of whether the predicted outcome agrees with the generated response.

H.4.4 Simulation Efficiency

We additionally evaluate the generative cost required to acquire learner-specific simulation capability and subsequently use it throughout final testing. For every generative API call made by a simulation method, prompt and completion tokens are recorded and included in the corresponding stage cost.

Capability Acquisition Cost. Capability Acquisition Cost measures the total generative token consumption incurred before final testing. For Learner2Skill,

$$C_{\text{acq}} = C_{\text{construct}} + C_{\text{calibrate}}, \tag{91}$$

where $C_{\text{construct}}$ denotes the generative token consumption of Skill Construction and $C_{\text{calibrate}}$ denotes that of Executor Calibration.

For a comparison method, C_{acq} analogously includes all generative operations performed before final testing to construct, extract, summarize, or otherwise acquire learner-specific simulation information.

Test-Time Cost. Test-Time Cost measures the total generative token consumption during the final test stage:

$$C_{\text{test}} = \sum_{i=1}^{N_{\text{test}}} T_i^{\text{test}}, \tag{92}$$

where T_i^{test} denotes the total number of prompt and completion tokens consumed by the evaluated method for the i-th test interaction.

This cost includes both learner-response generation and method-specific online maintenance performed during testing. For Learner2Skill, LLM calls used for response-pattern maintenance during Skill Evolution are therefore included in C_{test}.

Total Token Cost. We define the total generative cost over the complete evaluation procedure as

$$C_{\text{total}} = C_{\text{acq}} + C_{\text{test}}. \tag{93}$$

Thus, C_{total} captures both the cost of acquiring learner-specific simulation capability before testing and the cost of using and maintaining that capability throughout the final test trajectory.

Consistency Efficiency. Finally, we report **Consistency Efficiency (CE)** to jointly characterize simulation consistency and overall generative cost:

$$Q_{\text{CE}} = \frac{\bar{Q}_{\text{consistency}}}{C_{\text{total}}/10^6}. \tag{94}$$

Here, $\bar{Q}_{\text{consistency}} \in [0, 1]$ is the average Prediction–Response Consistency over all test interactions, and $C_{\text{total}}/10^6$ expresses the total generative cost in millions of tokens. Although effectiveness metrics are displayed as percentages in the experimental tables, CE is computed using the unscaled consistency value in $[0, 1]$. Accordingly, Q_{CE} measures the Prediction–Response Consistency obtained per million generated tokens, with higher values indicating greater consistency efficiency.

CE is reported only for methods that generate both an explicit outcome prediction and a complete learner response and is therefore not applicable to supervised learner-performance predictors.

H.5 Qualitative Analysis of Skill-Guided Learner Simulation

We further examine how learner-specific information encoded in a Simulation Skill manifests in generated responses. We select three held-out interactions that exhibit distinct response behaviors: structured conceptual reasoning, a recurring reasoning error, and early stopping after partial concept recognition. Together, these examples illustrate how persistent learner information guides fine-grained simulation across different stages of problem solving.

For each interaction, the executor receives the current task and the Simulation Skill available at that point in the learner trajectory. The Skill provides the current learning state and persistent response patterns, together with resource interfaces for accessing additional cognitive or historical evidence when needed. The executor produces the same structured response used throughout our experiments: `attempted`, `response_process`, `final_answer`, and `predicted_outcome`.

H.5.1 Structured Conceptual Reasoning

Learning task. Learner 20038 is asked to interpret Newton's second law, $F = ma$, and its algebraic transformations. The task requires reasoning about the relationships among force, mass, and acceleration.

Relevant Skill information. The learner's persistent response patterns characterize a structured strategy for conceptual problems: identifying the governing relation, checking candidate statements against the relevant concept, and forming an explicit conclusion.

```
Relevant Skill Information

Persistent response pattern:
On conceptual problems, the learner typically identifies the governing
relation, checks the relevant statements systematically, and then
    provides
an explicit final judgment.
```

Simulated learner response.

```
Simulation Output

{
  "attempted": true,
  "response_process": "The learner first applies Newton's second law
    and
  checks the relation between acceleration, net force, and mass.
    Although
  m = F/a follows algebraically from F = ma, mass is an intrinsic
    property
  of the object and should not be interpreted as being determined by
    force
  or acceleration.",
  "final_answer": "Acceleration is proportional to the net force and
  inversely proportional to mass.",
  "predicted_outcome": 1
}
```

Observed learner behavior. The real learner similarly reasoned from Newton's second law, checked the candidate statements, and produced the correct answer, with an observed outcome of 1. The simulated response captures the structured conceptual process represented in the learner's Skill together with the resulting answer.

H.5.2 A Persistent Omitted-Condition Error

Learning task. Learner 20007 encounters a kinematics problem with $v > 0$, $a < 0$, and an increasing magnitude of acceleration. Solving the task requires jointly considering multiple conditions to determine how the velocity changes.

Relevant Skill information. The learner's persistent response patterns contain a recurring behavior on multi-condition problems: reasoning from the most salient conditions while leaving some restrictions unchecked, followed by an explicit acknowledgement of the omitted conditions.

```
Relevant Skill Information

Persistent response pattern:
For problems requiring multiple conditions to be checked jointly, the
    learner
may reason from salient conditions without examining all restrictions.
The resulting response may explicitly acknowledge the omitted
    conditions.
```

Simulated learner response.

```
Simulation Output

{
  "attempted": true,
  "response_process": "The learner uses the signs of velocity and
  acceleration to determine that the velocity is decreasing and
    considers
  the change in acceleration, but does not fully integrate all of the
  given conditions.",
  "final_answer": "Other result (not all conditions were examined).",
  "predicted_outcome": 0
}
```

Observed learner behavior. The real learner likewise used a subset of the given conditions, explicitly noted that some restrictions had not been checked, and produced the same type of final response. The observed outcome was 0.

This interaction illustrates how a persistent response pattern specifies the form of a learner's error. The Skill captures both the conditions associated with the behavior and its characteristic manifestation, which are reflected in the simulated reasoning process and final response.

H.5.3 Stopping after Partial Concept Recognition

Learning task. Learner 20443 is asked to identify the fundamental mechanical quantities in the International System of Units. The relevant concepts include length, mass, and time.

Relevant Skill information. The learner's persistent response patterns record a stopping tendency on some tasks that require further reasoning after initial concept recognition. The learner may identify the relevant knowledge and begin processing the problem while leaving the response unfinished.

```
Relevant Skill Information

Persistent response pattern:
The learner may recognize the relevant concept and begin processing
    the
problem, but stop before reaching a completed final answer.
```

Simulated learner response.

```
Simulation Output

{
  "attempted": true,
  "response_process": "The learner recognizes that the problem
    concerns
  fundamental quantities in the SI system and identifies length, mass,
  and time as relevant, but stops before completing the final judgment
    .",
  "final_answer": null,
  "predicted_outcome": 0
}
```

Observed learner behavior. The real learner similarly recognized the relevant concept and began the problem but stopped before completing the response. The observed outcome was 0.

Here, the Skill captures a response tendency that occurs between concept recognition and final-answer production. The generated process preserves this stopping point: the partial attempt is represented in `response_process`, while the unfinished response is represented by `final_answer: null`.

H.5.4 Analysis across Cases

The three interactions reveal how persistent response patterns shape different parts of learner behavior. In the first case, a structured strategy organizes the learner's reasoning toward a successful response. In the second, a recurring condition-checking pattern determines how an error emerges and is expressed. In the third, a stopping tendency determines whether recognized knowledge is carried through to a completed answer. These patterns operate at different stages of response generation—reasoning strategy, error formation, and response termination—while being represented within the same Simulation Skill.

The structured simulation interface makes these behavioral dimensions directly observable. The `attempted` field represents engagement with the task; `response_process` records the learner-

specific trajectory of the attempt; `final_answer` captures the response eventually reached, if any; and `predicted_outcome` represents the expected observable result. The examples therefore connect the learner information maintained in the Skill with the fine-grained behavioral dimensions evaluated in Section 4.1.

H.5.5 Skill Use and Evolution across Interactions

A Simulation Skill persists across the learner trajectory and evolves as new real interactions become available. For task e_t, the executor starts from the Skill $\mathcal{S}_u^{t-1}$ accumulated from previously observed interactions. Its current learning state and persistent response patterns provide learner-specific information for simulation, while the associated resource interfaces provide additional cognitive or historical evidence when required by the current task.

The executor produces the structured simulation through the same interface used in the examples above:

```
Simulation Interface

{
  "attempted": ...,
  "response_process": "...",
  "final_answer": "...",
  "predicted_outcome": ...
}
```

This output corresponds to $(\hat{r}_t, \hat{y}_t)$ in our formulation. After the real interaction (r_t, y_t) becomes available, it is incorporated into the learner's evidence history. The learning state is updated with the newly observed interaction, and persistent response patterns are revised according to the accumulated real evidence, yielding the Skill $\mathcal{S}_u^t$ for subsequent tasks.

The resulting sequential process is $\mathcal{S}_u^{t-1} \longrightarrow (\hat{r}_t, \hat{y}_t) \longrightarrow (r_t, y_t) \longrightarrow \mathcal{S}_u^t$. Across successive interactions, the Simulation Skill carries forward the learner-specific information acquired from previous observations and incorporates newly observed behavior. The executor can therefore reuse the same persistent Skill across tasks while the Skill evolves with the learner trajectory.

Ethics Statement

This work uses authentic learning interaction data collected from an online learning platform for high-school students, including exercise responses, scores, and learner-recorded problem-solving processes. Anonymized learner identifiers are used throughout the experiments, and all statistics and representative cases presented in the paper are reported in anonymized form. The data are used for research on learner modeling, learner simulation, and the evaluation of educational algorithms and instructional designs.

Learner2Skill provides a controllable and repeatable environment for simulating learner behavior, supporting the development of educational algorithms, the analysis of instructional strategies, and the evaluation of assessment designs. In this work, simulated responses are treated as approximations of real learner behavior and are evaluated against held-out authentic learner interactions. For practical educational applications involving real learners, simulated evidence can be combined with authentic learning evidence and human judgment.

Reproducibility Statement

To support reproducibility, we provide the complete formulation of Learner2Skill, experimental protocols, and implementation details in the main paper and appendix. The appendix further specifies the organization of the Simulation Skill, prompts used for Skill Construction and Executor Calibration, the cognitive-model implementation, the Skill Evolution procedure, baseline implementations, evaluation protocols, hyperparameters, API settings, and computing environment.

All learner interactions retain their original temporal order and are chronologically split into 60% training, 10% validation, and 30% test portions. All methods are ultimately evaluated on the same held-out test interactions. For LLM-based experiments, we report the models and their respective roles, generation settings, and evaluation setup. Each experiment is independently repeated three times under the same setting, and token consumption is recorded to support reproducibility of both effectiveness and efficiency analyses.

We additionally provide the experimental code, configuration files, and implementation resources required for reproduction through an anonymous code repository. We also plan to release LearnerTrace-2K to support future research on fine-grained learner simulation.