Teachers as Tutor Builders: Bridging the Expertise Gap in Intelligent Tutor Authoring

Anonymous submission

Abstract

Classroom teachers seeking to use intelligent tutoring systems (ITS) are often limited to rigid, template-based platforms that restrict pedagogical creativity and limit personalized learning. This is because creating a custom tutor requires expertise in programming, interface design, and instructional design, in addition to their own domain knowledge. While authoring tools with more creative control exist, they often fail to adequately scaffold these skills, creating an "expertise gap." We present three studies with 13 teachers that probe this gap by evaluating common tutor authoring system design choices. For this investigation, we developed Apprentice Tutor Builder (ATB), a tutor authoring tool featuring a drag-and-drop interface builder and an AI agent that generates expert models via user demonstration and feedback. Our findings show that while teachers can successfully author tutors with some scaffolding, they struggle in areas such as higher-level interface planning and validating the resulting expert model. A comparative study shows that while large language models can be guided to solve problems, their out-of-the-box performance is unreliable. Based on our findings, we present design recommendations for future teacher-centered tools that bridge the expertise gap by embedding pedagogical and technical scaffolding directly into the authoring environment.

Introduction

Intelligent tutoring systems (ITSs) can improve student learning outcomes by providing guided practice, performance feedback, and personalized problem selection (Koedinger and Anderson 1997; Ma et al. 2014). However, developing these systems is time intensive and can take tens to hundreds of hours to produce a single hour of instruction time (Murray 1999, 2003; Aleven et al. 2009; Weitekamp, Harpstead, and Koedinger 2024), which prevents their widespread adoption (Gupta and Maclellan 2021).

Creating an effective intelligent tutor requires a combination of skills beyond a user's own domain knowledge. These skills include expertise grounded in learning science and learning engineering, which informs the **instructional design** decisions that structure the learning experience. This pedagogical expertise guides authoring choices such as determining the appropriate kinds of tutors to create, the necessary level of scaffolding, and the optimal granularity for practice problems. This is coupled with expertise in **user interface design** to create a usable and engaging interface, and **programming** to implement the tutor's logic, including the expert model that guides and evaluates students. Existing authoring tools often implicitly assume *some* technical or design background (Lane, Core, and Goldberg 2015), creating an "expertise gap" that excludes many classroom teachers who lack formal training in these areas. Consequently, teachers often depend on pre-built or template-based tutors that may not align with their specific instructional goals.

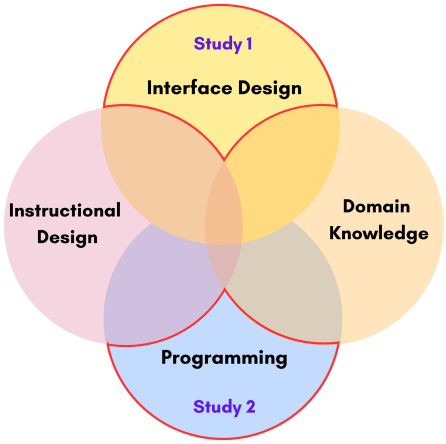


Figure 1: The four overlapping areas of expertise required for authoring intelligent tutors. This paper investigates the design space in the two highlighted areas: UI authoring (Study 1) and expert model authoring (Study 2), while controlling for the other areas of expertise.

We address this gap by investigating how authoring tools can become truly teacher-centric, leveraging their pedagogical expertise rather than requiring a technical mindset. To achieve this, we posit that future authoring tools should strive to meet several key design goals (see Table 1). To test our hypotheses about how to meet these design goals within this space, we developed Apprentice Tutor Builder (ATB), a design probe featuring a drag-and-drop interface builder and an AI agent that learns from user demonstration and feedback to create expert models.

We structure our investigation into three studies. **Study 1** focuses on tutor interface authoring, while **Study 2** investigates expert model creation using an interactive machine

Design Goal	Description
DG1	Feature a tutor UI authoring approach that is usable by end users.
DG2	Enable AI agents to interpret user-created interfaces and learn from them to create expert models.
DG3	Ensure the resulting expert model is explainable and comprehensible to the teacher.
DG4	Require a minimal number of examples for the AI to learn, similar to what a human would need.
DG5	Be usable by teachers with minimal prior AI and coding experience.

Table 1: Desired attributes of interactive machine learning-based tutor authoring systems

learning (IML) approach. We should note that while a range of authoring frameworks exist, we focus our investigation on IML-based systems. IML approaches to expert model creation directly address the programming gap by framing the authoring process with intuitive interactions like demonstration and feedback. These approaches also represent the state-of-the-art in tutor authoring research. Finally, in **Study 3**, we conduct a comparative evaluation of Large Language Models (LLMs) serving as expert models to investigate an alternative state-of-the-art authoring approach. We conclude by presenting design recommendations for future authoring tools that aim to bridge the expertise gap by embedding pedagogical and technical scaffolding directly into the authoring environment.

Related Works

Authoring an intelligent tutor requires a blend of domain, instructional, interface, and programming expertise, creating a significant barrier for educators. Existing authoring tools navigate this challenge by trading expressive power for ease of use. Our work with Apprentice Tutor Builder (ATB) addresses these trade-offs by comparing our approach to dominant frameworks in the field.

Direct cognitive modeling tools, such as the Cognitive Tutor Authoring Tools (CTAT) (Koedinger et al. 2004), offer fine-grained control but require programming or cognitive science expertise to create the underlying expert models. While CTAT's "example tracing" mode lowered this technical barrier, it still requires authors to adopt a formal task-analysis mindset, a key component of learning engineering expertise that many teachers may not formally possess. Our goal is to capture teacher expertise without requiring them to adopt a formal programming or task-analysis framework.

A second framework prioritizes ease of use through templates. Platforms like ASSISTments (Razzaq et al. 2009) and OATutor (Pardos et al. 2023) simplify interface design with predefined templates, lowering the barrier to entry but limiting a teacher's ability to create custom problem structures that align with their pedagogical goals. Similarly, REDEEM (Ainsworth et al. 2003) helps teachers apply pedagogical strategies to existing content but is less suited for authoring complex, procedural tutors from scratch. ATB's goal is to provide a more flexible interface builder and expert model authoring process, striving for a better balance between usability and expressive power.

Within the interactive machine learning (IML) paradigm, "apprentice learning" systems like SimStudent (Matsuda et al. 2008), the Apprentice Learner (AL) (Maclellan et al. 2016), and their successors AL+ (Weitekamp, Harpstead, and Koedinger 2020) and AI2T (Weitekamp, Harpstead, and Koedinger 2024) are promising. Like ATB, these systems use the intuitive metaphor of an author "teaching" an AI agent through demonstration and feedback, which leverages activities natural for educators. However, these systems have primarily been evaluated in controlled lab studies. A central goal of our work is to evaluate the framework with practicing teachers to uncover the specific scaffolding needed to bridge the gap between lab and classroom.

Finally, we should note the recent emergence of large language models (LLMs) in education. While LLMs show promise in tasks such as generating course materials (Pal Chowdhury, Zouhar, and Sachan 2024; Elkins et al. 2024) or assessment grading (Chapagain and Rus 2025; Cohn et al. 2024), using them directly as tutors introduces risks like factual inaccuracies ("hallucinations") and a lack of transparency (Chen et al. 2024). A more viable approach, aligned with our work, is using generative AI as a component within an authoring tool rather than as the tutor itself. Whereas a fully autonomous LLM removes teacher agency, our approach maintains that the teacher is the ultimate authority, using AI as a scaffold to build, validate, and refine tutor components.

ATB System Description

ATB features two primary components designed to test our hypotheses about scaffolding the required expertise. The **Interface Builder** is a drag-and-drop tool that allows us to evaluate the usability of common UI authoring features (DG1, DG5). The tool features a row-and-column layout framework (Zhang, Liu, and Huang 2019) that lets users to nest rows and columns within one another to help position and align interface elements. Additionally, it helps ensure tutors can be displayed properly on a range of devices, such as phones, tablets, and computers. The **Agent Training Module** explores a teachable AI approach (Gupta and Maclellan 2021) for expert model creation (DG2, DG3, DG4) by reframing the abstract task of programming into the more familiar activity of teaching. This module is composed of three core components: a knowledge representation, a performance component, and a learning component.

Knowledge Representation. The apprentice agent's knowledge is composed of two core structures: a Hierarchical Task Network (HTN) for procedural knowledge and a working memory for state information (Erol, Hendler, and Nau 1994). HTNs are a knowledge-based framework that solve complex problems by recursively decomposing high-level tasks into smaller, hierarchically organized subtasks. This decomposition forms an AND-OR tree, where

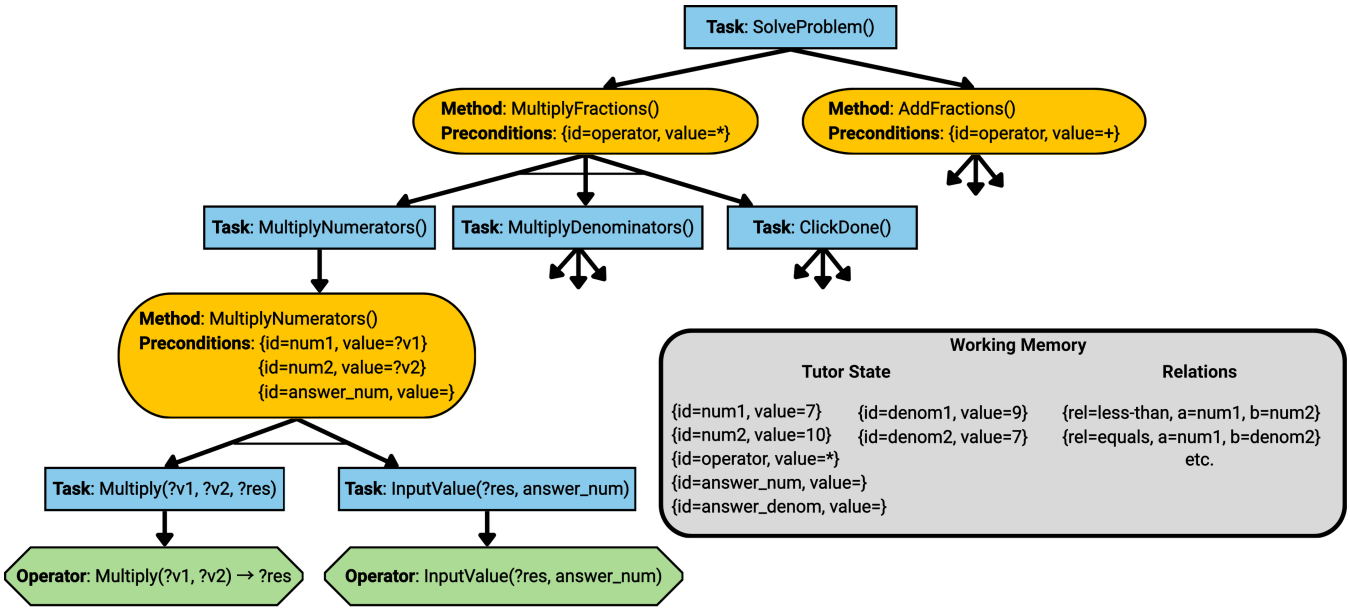


Figure 2: A partial HTN decomposition for multiplying the numerators in a fraction multiplication problem (upper left). The initial working memory consists of information about the tutor's initial state (lower right).

solving a task (OR) requires selecting one valid *method*, and completing a method (AND) requires executing all of its constituent subtasks. These subtasks ultimately resolve into *operators*, which are the primitive actions the agent can perform. The agent's working memory complements the HTN by containing known and inferred information about the current problem. It combines data extracted directly from the tutor interface with the results of relational inference (e.g., equals(a, b)), a process which deduces relationships between values (Maclellan et al. 2016). For this study, our agent initially receives the operators $\{$*add, subtract, multiply, divide, input_value, click_done*$\}$ and the relations $\{$*equals, less_than*$\}$.

Methods are used to represent the various ways a task can be solved (e.g., an AddFractions method vs. a MultiplyFractions method, see Figure 2). The agent selects the appropriate method by matching the current working memory state against each method's *preconditions* - criteria that are automatically acquired during learning. The agent is provided with a set of predefined operators that either update the working memory, analogous to "mental steps" (e.g., multiply), or manipulate the tutor interface (e.g., input_value). We chose an HTN as the core representation due to its strong alignment with established pedagogical principles. Its hierarchical structure enables personalized and adaptive instruction by supporting skill composition, scaffolding at different levels of granularity (Munshi et al. 2023), and the recognition of non-linear solution strategies (Siddiqui et al. 2024). Furthermore, the explicit, symbolic nature of HTNs allows the system to generate human-understandable explanations for its reasoning.

Performance Component. To solve a problem, the agent uses its HTN to generate actions based on the current tutor state. The process begins by converting the state into *facts* in working memory (see Figure 2) and inferring all possible relations between them. The agent then matches the working memory against the preconditions of available methods to find one that can achieve the current task. For example, the *MultiplyFractions* method is selected when a multiplication symbol (*) is present in the working memory. Once a method is found, the agent recursively attempts to solve its subtasks until it reaches operator nodes. These operators are then executed, with their outputs updating the working memory and any interface-related values being returned as problem-solving steps. In our example, the *MultiplyFractions* method decomposes into subtasks like *MultiplyNumerators*, which ultimately resolves into a *Multiply* operator to compute the product and an *InputValue* operator to return the result to the interface. If at any point no matching method is found, the agent cannot produce a solution and signals to the learning component that additional training is required.

Learning Component. The agent acquires new knowledge - specifically, methods and their preconditions - through a combination of demonstration and feedback, see Figure 3. Initially, the agent possesses predefined operators but lacks the methods to solve tasks. When unable to proceed, it requests a demonstration and a task label from the teacher. Upon receiving a demonstration, the agent attempts to *explain* the teacher's action by searching for a sequence of its known operators that can reproduce the demonstrated value given the current tutor state. If successful, it creates a new method whose subtasks correspond to the discovered operator sequence and whose preconditions are a direct copy of the facts in the current working memory. If no explanation can be found, the agent creates a simpler "memorized action" method that directly outputs the demonstrated value

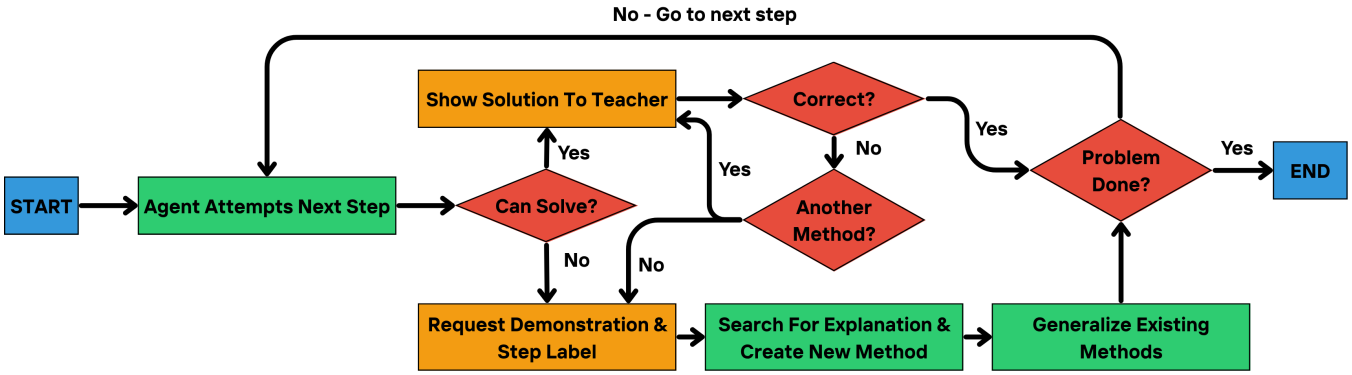


Figure 3: The AI agent learning algorithm to produce expert models. Green boxes are agent operations and orange boxes are places where user input is requested.

in that specific context.

In addition to learning from demonstrations, the agent also uses feedback to refine *when* to apply methods (the preconditions). When attempting a step, it prompts the teacher for a "Yes/No" confirmation. Negative feedback causes the agent to either try another applicable method or, if none found, request a new demonstration. It is important to note that our learning algorithm only generalizes conditions from positive demonstrations; it does not use negative feedback to perform condition refinement (i.e., specializing the rule that incorrectly applied so it no longer does). While this is a complementary operation to generalization, it was not necessary for the tasks in this study.

To generalize precondition rules, the system uses **anti-unification**, a machine learning technique for generalization (Plotkin 1970; Bulychev, Kostylev, and Zakharov 2010; Jung, Lutz, and Wolter 2020). If any two methods for the same task share an identical subtask sequence, their preconditions are merged using this process. This process pairwise-compares the facts from each method's preconditions, replacing differing values with variables while keeping constant values fixed, therefore finding a least-general-generalization that covers all previously observed demonstrations. This allows the agent to form rules from a minimal number of examples, which is a crucial design goal (DG4).

Evaluation of the ATB Authoring Framework

To investigate the effectiveness of ATB in scaffolding the authoring process for teachers, we designed a mixed-methods study to answer two primary research questions:

RQ1: To what extent can teachers, with varying levels of technical expertise, successfully use ATB to author the UI and expert model components of an intelligent tutor?

RQ2: What patterns and challenges emerge when teachers use a programming-by-demonstration approach to train a learning agent?

The evaluation was divided into two studies, each designed to isolate and test specific aspects of the authoring process. Along with capturing quantitative measures for each study, we conducted a semi-structured interview at the end of the second study to gauge participants' user experience. To analyze this data, we performed a *reflexive thematic analysis* as outlined by Braun and Clarke (Braun and Clarke 2006). We selected this approach to conduct an exploratory, inductive analysis to identify emerging themes from the teachers' experiences. Our process involved multiple researchers collaboratively discussing and refining codes and themes, which ensured a nuanced interpretation of the participants' perspectives.

Participants

We recruited 13 educators (ages 18-54) with professional teaching experience ranging from K-8 to college and private industry. Based on self-reported programming and AI experience, we divided participants into a "Low Expertise" group (n=6, little to no programming experience) and a "High Expertise" group (n=7, moderate to high experience).

Study 1: Tutor Interface Authoring

The goal of this study was to assess the usability of the interface builder (part of RQ1 and DG1) for end users with minimal prior experience (DG5).

Methodology. After providing consent and demographic information, participants viewed a 3-minute tutorial video demonstrating both the interface builder and the agent training process. Participants were tasked with recreating the interfaces for two tutors: "Fraction Arithmetic" and "Square 25" (a procedural task for squaring numbers ending in 5, see Figure 4). By providing target designs, we controlled for instructional and UI design expertise, isolating challenges related to the tool itself. To evaluate performance, we measured: (1) **Completion Time** and (2) **Interaction Data**, specifically the frequency of 'delete' actions. We used the latter as a proxy for planning difficulty, as our row-column layout required deleting and re-adding elements to change their spatial order.

Results. All 13 participants successfully completed the UI authoring tasks. While authoring times for the Fractions tutor were comparable between groups (Low: 6.59 min, High: 5.99 min), the Low expertise group took nearly twice as

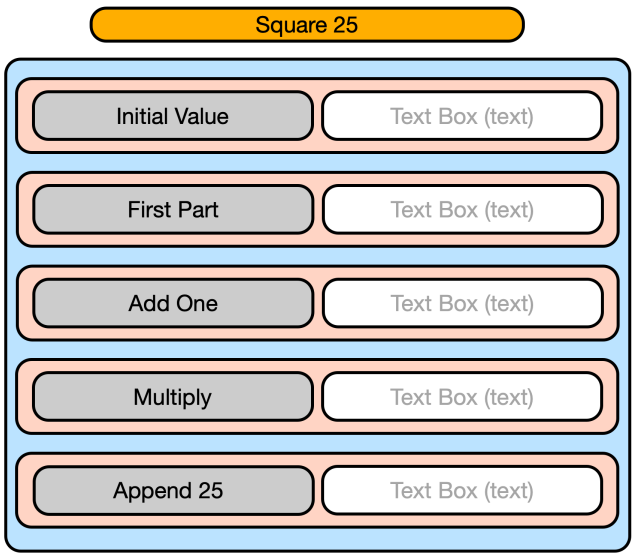


Figure 4: An example of the Square 25 interface created by a participant. The elements are scaffolded by rows (red boxes) and columns (blue box).

long on the simpler Square 25 tutor (6.33 min vs. 3.33 min) and performed significantly more "delete" actions. This suggests that the constraints of the row-column layout created significant planning challenges. Qualitative feedback shows that while participants appreciated the drag-and-drop concept, they found its implementation overly restrictive. As P6 noted, "It would be great if I could space things out how [I want]...floating at the top rather than in line." This indicates that the challenge is not the presence of scaffolding itself, but its specific design. A scaffold that is too rigid can limit user control and creative flexibility, suggesting that effective tools must offer a better balance of the two.

Discussion. Our findings indicate that while teachers can successfully author tutor interfaces, some challenges remain. Despite a row-column scaffolding system to manage component grouping and alignment, some teachers struggled with layout planning and execution. This suggests that simply providing design options is insufficient; tools must also guide teachers in their design choices. Effective support could involve combining direct manipulation with pre-built templates, alignment guides, or generative AI that proposes editable layouts (Calo and Maclellan 2024). This approach guides users toward good design practices without removing teacher agency.

Study 2: Expert Model Authoring

The goal of this study was to investigate the AI-teacher interaction (RQ2) and to test the viability of our teachable AI framework for achieving our design goals related to agent learning (DG2), model explainability (DG3), and learning from minimal examples (DG4).

Methodology. Participants used the interfaces they had built in Study 1 to train ATB's learning agent. They were asked to teach it to solve two problem types for the fractions tutor (multiplication and addition with common denominators) and the complete procedure for the Square 25 tutor. We measured: (1) **Model Accuracy**, evaluated on a held-out set of randomly generated problems (20 for fractions, 10 for Square 25), (2) **Training Time**, and (3) the number of **Validation Problems**. We define this last metric as the number of extra examples a teacher provides *after* the agent has already demonstrated it can correctly solve a problem of that type, which probes the teacher's mental model of the agent's state and their approach to verification.

Results. 12 of 13 participants were able to author expert models that passed the holdout sets. All models authored by participants in the High expertise group achieved 100% accuracy. All but one model from the Low expertise group did so as well. The single failing model was trained exclusively on an edge case for multiplication (using '1' as a numerator), which resulted in the system learning to copy over the other numerator value when one of the numerators is a 1 rather than performing a true multiplication.

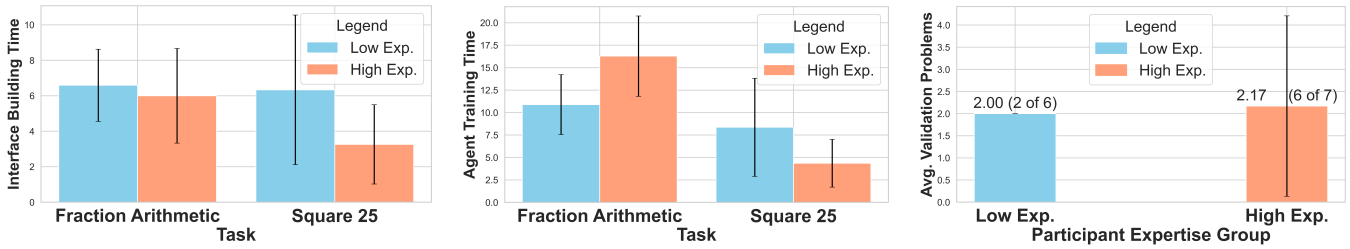
Analysis of training time showed differences based on task complexity. For the more complex fractions tutor, the High expertise group spent more time training the agent (15.14 min vs. 10.89 min). Conversely, for the simpler Square 25 tutor, the Low expertise group spent more time training (4.79 min vs. 8.36 min), see Figure 5.

The majority of the High Expertise group, 86% (6 of 7), provided at least one validation problem after the agent first succeeded on a task. In contrast, only 33% (2 of 6) of the Low Expertise group did so.

Participants generally found the agent training process to be intuitive and "natural," with one commenting, "It's essentially like coaching one student" (P18) and another adding, "I felt like I was talking to a student" (P16). Despite this, participants reported being uncertain about the agent's internal state and when it was truly "taught." For example, P12 expressed they "didn't really have an indication as to how many times I would need to train it." Participants also expressed a desire for more direct control over the agent's learning, with P13 wishing they could just "[tell] the thing... just multiply these two numbers" instead of relying on demonstration.

Discussion. A key challenge was the uncertainty participants felt about the agent's internal state, making it difficult to know when training was complete or how generalized the agent's knowledge had become. This opacity led one teacher to author an incorrect model by training only on an edge case, highlighting that a key task for the teacher is not just demonstrating solutions but also curating a *diverse and representative set of training examples* that allows the agent to produce a generalizable model.

We also observed different validation approaches between expertise groups. The High Expertise group seemed to adopt a "test-and-debug" model and more rigorously tested the agent with more validation problems. In contrast, the Low Expertise group used a "human student" model, assuming agent understanding after a single correct replication of the procedure. This implies that authoring tools must help non-



(a) Time to author tutor interface (in minutes) (b) Time to author expert model (in minutes) (c) Average of validation problems

Figure 5: Interface and expert model authoring times and the average number of validation problems for each expertise group. All error is reported at the 95% confidence interval level.

technical users build a more effective mental model for AI training, perhaps by prompting for validation in familiar pedagogical terms. The goal is to build upon a teacher's natural instinct to check for understanding, scaffolding them toward a more robust validation practice without demanding they think like "software testers".

To achieve this, future tools should serve as active co-authors, providing transparency through real-time feedback and offering guidance, such as requesting more varied examples to ensure robust learning. This aligns with the idea that tools should teach authors how to author effectively (Murray 2003; Lane, Core, and Goldberg 2015) and supports our goals for an explainable and efficient learning system (DG3, DG4).

Study 3: Comparative Evaluation of LLM-Based Tutors

We conducted a comparative evaluation of several LLMs serving as expert models, an increasingly prominent "LLM-as-tutor" paradigm. We investigated the performance of three leading models, Anthropic's Claude 3.5 Sonnet, Google's Gemini 2.5 Pro, and OpenAI's GPT-5, on the same problem types and holdout sets used in our user study.

Methodology. We evaluated each LLM under two conditions: (1) **zero-shot**, where the model solved each step of the holdout problems without any prior examples, and (2) **in-context learning**, where after the LLM attempted each step of a training problem, an example of the correct (or incorrect) action was appended to the prompt's context. After completing each problem, the LLM's performance was measured against the entire holdout set. This methodology is adapted from the TutorGym framework (Weitekamp, Siddiqui, and MacLellan 2025). Performance was measured by the problem completion rate on the holdout sets. The prompt provided the full tutor state, including each interface element's name, type, and spatial properties (x, y, width, height).

Results. Zero-shot results showed varied performance. Gemini 2.5 Pro and GPT-5 achieved a 30% completion rate on fraction tasks and 90% on the Square 25 task. In contrast, Claude 3.5 Sonnet performed well on fractions (92-100%) but failed to complete any Square 25 problems. In-context learning guided all models toward high accuracy. For fraction arithmetic, GPT-5 reached 100% accuracy after one epoch, while the others required five. However, Claude's performance later dropped due to sign errors (e.g., calculating -57 * 36 as 2,052 instead of -2,052). On the simpler Square 25 task, all models achieved 100% accuracy within two epochs. Although the evaluation was performed on a relatively small training and test sample, the total LLM inference cost of our evaluation was approx. $60.

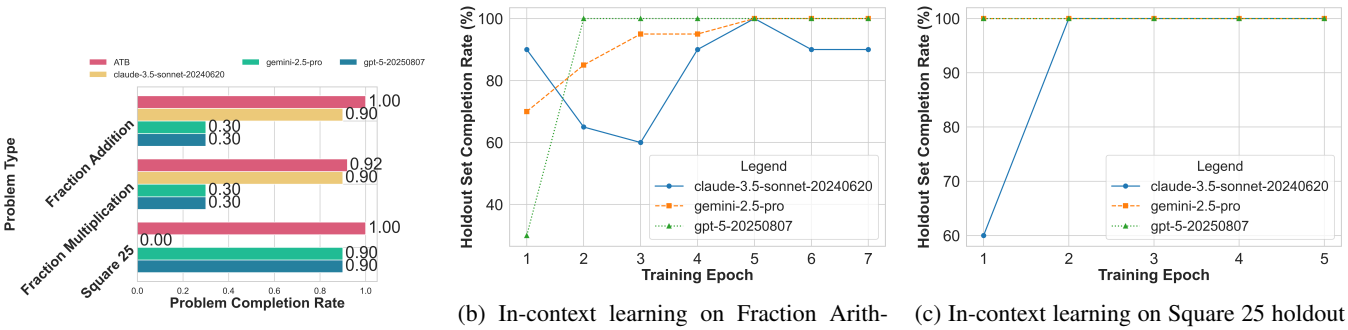
Discussion. Our findings reveal the significant risks of using LLMs as zero-shot expert models in tutoring. Their out-of-the-box performance is unpredictable, with some models failing entirely on simple procedural tasks. The "black box" nature of these models necessitates transparency and robust verification tools to audit for "hallucinations".

Although in-context learning can guide an LLM to solve problems correctly with a few examples (Zong and Krishnamachari 2023), this approach has caveats. The model does not truly "learn" in a way that alters its underlying parameters; rather, context merely guides its inference for a specific task (Weitekamp, Siddiqui, and MacLellan 2025). Practical barriers like non-trivial inference costs and context window limits - which can either cause the model to "forget" previous behaviors or prevent additional training - also remain. Finally, our work shows that if LLMs are, instead, used to generate tutor components, teachers must be able to edit these artifacts to maintain agency and control. Therefore, the future of LLM-based authoring is not replacement but a collaborative model, where the LLM provides an initial scaffold and the teacher, supported by tools like those we propose, refines and validates it.

Design Recommendations for Teacher-Centered Authoring Tools

Based on the findings of our studies, we propose the following design recommendations for future authoring tools that target teachers and other non-technical education stakeholders:

- **Embed Expertise through Hybrid UI Design:** To enable teachers to customize and personalize tutors while reducing technical overhead, interface authoring tools should move beyond a simple palette of components and feature hybrid systems that combine the direct manipulation of drag-and-drop with intelligent and editable au-



(a) Zero-shot performance on holdout sets.

(b) In-context learning on Fraction Arithmetic holdout set.

(c) In-context learning on Square 25 holdout set.

Figure 6: Zero-shot and in-context learning performance of several language models on Fraction Arithmetic and Square 25 tasks.

tomation. This could be a generative AI model that proposes an initial layout or context-aware templates that serve as an editable starting point, embedding design expertise without removing teacher agency.

- **Integrate Real-Time, Verifiable Feedback:** To align with a teacher's need to assess understanding and build trust in the tutor, systems should provide transparent, real-time mechanisms for validating expert models created with machine learning agents such as those presented in (MacLellan and Koedinger 2022). This could be a dashboard that shows the model's accuracy on a set of system-generated problems or a feature that lets the teacher quickly input test cases and see if the agent can solve them. This makes model accuracy verifiable and enables teachers to more confidently determine when their tutor is ready for deployment.
- **Structure Authoring as a Guided Dialogue:** Frame the authoring process as a collaboration where the system serves as a co-author. By using prompts to elicit and formalize a teacher's pedagogical knowledge (e.g., *"That's a new strategy. What would you call this method?"*), the system can help teachers build more robust and *complete* expert models that define a broader space of solution strategies (Weitekamp, Harpstead, and Koedinger 2024). Additionally, to help teachers create generalizable models, the system should guide teachers in providing a varied set of training examples to minimize the required number of examples. This process is akin to scaffolding AI/machine learning expertise.

Limitations

We acknowledge several limitations of this work. Our study involved a relatively small sample of 13 teachers. While this was sufficient for a qualitative evaluation of our design probe, a larger-scale study may be needed to generalize our findings. Second, the authoring tasks were relatively constrained. We used simple procedural domains (fraction arithmetic and a squaring algorithm) to control for domain expertise, and participants were asked to recreate existing interfaces rather than design novel ones, which may not fully reflect the challenges of designing a tutor from scratch. Future work should explore how these findings apply to more complex domains and more open-ended design tasks. Finally, the design space of tutor authoring systems is vast. This work serves as a probe into a single point within that design space, with a focus on state-of-the-art systems that give the most user control. Future work might investigate new or broader reaching areas in tutor authoring.

Conclusions and Future Work

This work demonstrates that while teachers are capable of authoring intelligent tutors, their efforts are hindered when tools assume expertise in interface design, instructional design, and programming. Our studies yield two key design insights for the next generation of teacher-centered authoring tools: (1) for UI authoring, systems should embed design expertise through editable automation and scaffolds that balance guidance with creative control, and (2) for IML-based expert model authoring, teachable systems should provide appropriate training guidance and be paired with transparent, verifiable feedback to build teacher confidence and ensure model accuracy.

Building on these findings, our future work will focus on creating systems that act as co-authors. We will explore hybrid interface builders that combine direct manipulation with generative AI suggestions and enhance agent transparency through integrated testing visualizations. While our studies focused on scaffolding interface design and programming expertise, future tools must also address the instructional design gap (see Figure 1) by acting as pedagogical partners that help teachers structure content and apply learning science principles. Finally, by integrating capabilities for learning from demonstration, feedback, and natural language instruction, such as in (Lawley and Maclellan 2024), we aim to create a unified expert model authoring experience that truly enables teachers to create effective, personalized learning opportunities for their students.

References

Ainsworth, S.; Major, N.; Grimshaw, S.; Hayes, M.; Underwood, J.; Williams, B.; and Wood, D. 2003. REDEEM: Simple intelligent tutoring systems from usable tools. In *Authoring Tools for Advanced Technology Learning Environments:*

Toward Cost-Effective Adaptive, Interactive and Intelligent Educational Software, 205–232. Springer.

Aleven, V.; Mclaren, B.; Sewall, J.; and Koedinger, K. 2009. A New Paradigm for Intelligent Tutoring Systems: Example-Tracing Tutors. *I. J. Artificial Intelligence in Education*, 19: 105–154.

Braun, V.; and Clarke, V. 2006. Using thematic analysis in psychology. *Qualitative Research in Psychology*, 3(2): 77–101.

Bulychev, P. E.; Kostylev, E. V.; and Zakharov, V. A. 2010. Anti-unification Algorithms and Their Applications in Program Analysis. In Pnueli, A.; Virbitskaite, I.; and Voronkov, A., eds., *Perspectives of Systems Informatics*, 413–423. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN 978-3-642-11486-1.

Calo, T.; and Maclellan, C. 2024. Towards Educator-Driven Tutor Authoring: Generative AI Approaches for Creating Intelligent Tutor Interfaces. In *Proceedings of the Eleventh ACM Conference on Learning@ Scale*, 305–309.

Chapagain, J.; and Rus, V. 2025. Automated Assessment of Student Self-Explanation in Code Comprehension Using Pre-Trained Language Models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, 28996–29003.

Chen, Y.; Ding, N.; Zheng, H.-T.; Liu, Z.; Sun, M.; and Zhou, B. 2024. Empowering private tutoring by chaining large language models. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*, 354–364.

Cohn, C.; Hutchins, N.; Le, T.; and Biswas, G. 2024. A chain-of-thought prompting approach with llms for evaluating students' formative assessment responses in science. In *Proceedings of the AAAI conference on artificial intelligence*, volume 38, 23182–23190.

Elkins, S.; Kochmar, E.; Cheung, J. C.; and Serban, I. 2024. How teachers can use large language models and bloom's taxonomy to create educational quizzes. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, 23084–23091.

Erol, K.; Hendler, J. A.; and Nau, D. S. 1994. UMCP: A Sound and Complete Procedure for Hierarchical Task-network Planning. In *Aips*, volume 94, 249–254.

Gupta, A.; and Maclellan, C. J. 2021. Designing Teachable Systems for Intelligent Tutor Authoring. *AAAI2021 Spring Symposium on Artificial Intelligence for K-12 Education*.

Jung, J. C.; Lutz, C.; and Wolter, F. 2020. Least General Generalizations in Description Logic: Verification and Existence. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(03): 2854–2861.

Koedinger, K.; Aleven, V.; Heffernan, N.; McLaren, B.; and Hockenberry, M. 2004. Opening the Door to Non-programmers: Authoring Intelligent Tutor Behavior by Demonstration. *International conference on intelligent tutoring systems*.

Koedinger, K. R.; and Anderson, J. R. 1997. Intelligent Tutoring Goes To School in the Big City. *International Journal of Artificial Intelligence in Education*, 8: 30–43.

Lane, H. C.; Core, M. G.; and Goldberg, B. S. 2015. Lowering the Technical Skill Requirements for Building Intelligent Tutors: A Review of Authoring Tools. *Design Recommendations for Intelligent Tutoring Systems*, 303.

Lawley, L.; and Maclellan, C. 2024. VAL: Interactive Task Learning with GPT Dialog Parsing. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, CHI '24. New York, NY, USA: Association for Computing Machinery. ISBN 9798400703300.

Ma, W.; Adesope, O. O.; Nesbit, J. C.; and Liu, Q. 2014. Intelligent tutoring systems and learning outcomes: A meta-analysis. *Journal of educational psychology*, 106(4): 901.

Maclellan, C. J.; Harpstead, E.; Patel, R.; and Koedinger, K. R. 2016. The Apprentice Learner Architecture: Closing the Loop between Learning Theory and Educational Data. *International Educational Data Mining Society*.

MacLellan, C. J.; and Koedinger, K. R. 2022. Domain-general tutor authoring with apprentice learner models. *International Journal of Artificial Intelligence in Education*, 1–42.

Matsuda, N.; Cohen, W. W.; Sewall, J.; Lacerda, G.; and Koedinger, K. 2008. SimStudent : Building an Intelligent Tutoring System by Tutoring a Synthetic Student. In *NA*.

Munshi, A.; Biswas, G.; Baker, R.; Ocumpaugh, J.; Hutt, S.; and Paquette, L. 2023. Analysing adaptive scaffolds that help students develop self-regulated learning behaviours. *Journal of Computer Assisted Learning*, 39(2): 351–368.

Murray, T. 1999. Authoring Intelligent Tutoring Systems: An analysis of the state of the art. *International Journal of Artificial Intelligence in Education (IJAIED)*, 10: 98–129.

Murray, T. 2003. An Overview of Intelligent Tutoring System Authoring Tools: Updated analysis of the state of the art. *Authoring Tools for Advanced Technology Learning Environments: Toward Cost-Effective Adaptive, Interactive and Intelligent Educational Software*, 491–544.

Pal Chowdhury, S.; Zouhar, V.; and Sachan, M. 2024. AutoTutor meets Large Language Models: A Language Model Tutor with Rich Pedagogy and Guardrails. In *Proceedings of the Eleventh ACM Conference on Learning@ Scale*, 5–15.

Pardos, Z. A.; Tang, M.; Anastasopoulos, I.; Sheel, S. K.; and Zhang, E. 2023. Oatutor: An open-source adaptive tutoring system and curated content library for learning sciences research. In *Proceedings of the 2023 chi conference on human factors in computing systems*, 1–17.

Plotkin, G. D. 1970. A note on inductive generalization. *Machine intelligence*, 5(1): 153–163.

Razzaq, L.; Patvarczki, J.; Almeida, S. F.; Vartak, M.; Feng, M.; Heffernan, N. T.; and Koedinger, K. R. 2009. The Assistment Builder: Supporting the life cycle of tutoring system content creation. *IEEE Transactions on Learning Technologies*, 2(2): 157–166.

Siddiqui, M. N.; Gupta, A.; Reddig, J. M.; and MacLellan, C. J. 2024. HTN-Based Tutors: A New Intelligent Tutoring Framework Based on Hierarchical Task Networks. In *Proceedings of the Eleventh ACM Conference on Learning@ Scale*, 491–495.

Weitekamp, D.; Harpstead, E.; and Koedinger, K. 2024. AI2T: Building Trustable AI Tutors by Interactively Teaching a Self-Aware Learning Agent. *arXiv preprint arXiv:2411.17924*.

Weitekamp, D.; Harpstead, E.; and Koedinger, K. R. 2020. An Interaction Design for Machine Teaching to Develop AI Tutors. *Conference on Human Factors in Computing Systems - Proceedings*, July(July): 1–11.

Weitekamp, D.; Siddiqui, M. N.; and MacLellan, C. J. 2025. TutorGym: A Testbed for Evaluating AI Agents as Tutors and Students. arXiv:2505.01563.

Zhang, J.; Liu, C.; and Huang, A. 2019. Implementation of Responsive Web Page Layout Based on Media Query and Flexible Box Model. In Zhao, P.; Ouyang, Y.; Xu, M.; Yang, L.; and Ren, Y., eds., *Advances in Graphic Communication, Printing and Packaging*, volume 543 of *Lecture Notes in Electrical Engineering*. Singapore: Springer.

Zong, M.; and Krishnamachari, B. 2023. Solving math word problems concerning systems of equations with gpt-3. In *Proceedings of the AAAI conference on artificial intelligence*, volume 37, 15972–15979.