AutoTutor meets Large Language Models: A Language Model Tutor with Rich Pedagogy and Guardrails

Sankalan Pal Chowdhury
sankalan.palchowdhury@ethz.ch
ETH Zurich
Switzerland

Vilém Zouhar
vzouhar@ethz.ch
ETH Zurich
Switzerland

Mrinmaya Sachan
msachan@ethz.ch
ETH Zurich
Switzerland

MWPTutor

I see your solution. You seem to have made some errors. How about we start fresh and do it step by step?

Okay, I'm not sure where to start though.

What should be the first step?

I think Cody's starting amount of tickets, which is 49, should be the first step.

Tell me this instead: How many tickets does Cody have left after spending 25 on a beanie from his original 49 tickets?

Oh, he would have 24 tickets left.

24 is correct, well done. What is the next step?

Next, he won 6 more tickets, so I think we add that to the 24 he already has.

Okay, can you do that calculation?

Yes, 24 + 6 is 30, so Cody would have 30 tickets in total.

30 is the correct answer to the problem. Well done!!! Don't hesitate to contack me again if you have any troubles.

GPT4 Tutor

Hi! Could you walk me through your solution?

I'm not sure where to start. I know Cody won 49 tickets and spent 25 on a beanie, but I don't know how to figure out how many he has left.

That's a great starting point. So, if Cody had 49 tickets and he spends some on a beanie, what kind of math operation would you use to find out how many tickets he would have left?

Subtraction, I think.

Exactly, subtraction is the right operation. Can you try subtracting the number of tickets Cody spent on the beanie from the number of tickets he initially won? What do you get?

— 12 more tutor utterances —

28 tickets.

That's correct. Now, if we subtract another ticket from the 28, what number do we get?

27 tickets.

Yes, that's right. And another ticket taken from 27? What does that leave us with?

26 tickets.

Exactly. And if we take one more away from 26, where does that get us?

25 tickets.

— 13 more tutor utterances —

Figure 1: A pair of real tutoring conversations with GPT4 and MWPTutor. GPT4 does not have any higher level control, so it begins to cycle in its tutoring. As a result, while MWPTutor took 6 tutor utterances to guide the student to the correct answer, GPT4 took 29 tutor utterances and the student did not arrive at the correct solution.

ABSTRACT

Large Language Models (LLMs) have found several use cases in education, ranging from automatic question generation to essay evaluation. We explore the potential of using LLMs to author Intelligent Tutoring Systems. A common pitfall of using LLMs as tutors is their straying from desired pedagogical strategies such as leaking the answer to the student and providing no guarantees on the validity or appropriateness of the tutor assistance. While LLMs with certain guardrails can take the place of subject experts, the overall pedagogical design still needs to be handcrafted for the best learning results. Based on this principle, we create a sample end-to-end tutoring system named MWPTutor, which uses LLMs to fill in the state space of a pre-defined finite state transducer. This approach retains the structure and the pedagogy of traditional tutoring systems that has been developed over the years by learning scientists but brings in additional flexibility of LLM-based approaches. Through a human evaluation study on two datasets with math word problems, we show that our hybrid approach achieves a better overall tutoring score than an instructed, but otherwise free-form, GPT-4. MWPTutor is completely modular and opens up the scope for the community to improve its performance by refining its individual modules or using different teaching strategies that it can follow.

CCS CONCEPTS

• **Human-centered computing → Collaborative and social computing systems and tools**; **Empirical studies in collaborative and social computing**; • **Applied computing → Computer-assisted instruction**; **Computer-managed instruction**.

KEYWORDS

Tutoring, math word problems, Large Language Models, AutoTutor

ACM Reference Format:

Sankalan Pal Chowdhury, Vilém Zouhar, and Mrinmaya Sachan. 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 (L@S '24), July 18–20, 2024, Atlanta, GA, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3657604.3662041

L@S '24, July 18–20, 2024, Atlanta, GA, USA

ACM ISBN 979-8-4007-0633-2/24/07
https://doi.org/10.1145/3657604.3662041

1 INTRODUCTION

Computer-based automated tutoring has long been seen as the method to scale up the benefits of expert human tutoring to millions of students, a problem originally posed by Bloom [3]. Of various attempts to do so, the AutoTutor family [20] is perhaps one of the more notable ones. This family of tutors simulates natural language conversations with students based on pre-defined scripts, aiming to emulate one-on-one tutoring. While the AutoTutor family has shown substantial learning gains [20], it is difficult to expand it to different topics. Each expansion requires a lot of expert labor to incorporate the required domain knowledge for every concept in the curriculum and appropriate teaching strategies [15].

Recent developments in Large Language Models (LLMs), such as GPT [22], or FlanT5 [8], have opened up a new venue for computer-based tutoring where the LLMs play the role of the teacher. While previous work has shown the success of LLMs for tutoring [2, 27] the non-modular black-box nature of these models makes it impossible to make incremental improvements or control the pedagogy it pursues in assisting the student. The uninterpretable and uncontrollable nature of LLMs makes it difficult for experts or teachers to make changes in their strategy. It is also difficult to develop trust towards such models, not least because they might be open to jailbreak attacks [17] as well as privacy concerns. Finally, the proprietary nature of highly performant LLMs means that the tutor needs a *persistent* internet connection and an OpenAI account to function, and we are highly dependent on OpenAI policies. This could be bypassed by fine-tuning a local open LLM, but doing so leads to either prohibitively high resource requirements, or significantly worse performance (see FlanT5 vs ChatGPT in [18]).

The design process of AutoTutor and other similar intelligent tutoring systems can be split into two distinct stages, carried out by different, yet potentially overlapping, sets of stakeholders. The first stage is the *general framework level design*, carried out mostly by learning scientists (possibly along with computer scientists). This includes designing the pedagogy, the mechanisms required to interact with the student, and potentially even authoring tools. This is typically done only once for an entire domain. The second stage, performed by either teachers or domain experts, is to *author the scripts*[1], which needs to be done separately for every topic, question, or concept. In the standard framework, both these stages are done by humans. In contrast, in the LLM-only framework discussed in the previous paragraph, both these stages are carried out (implicitly or explicitly) by LLMs.

In this paper, we develop a mixed approach where we only delegate the job of domain experts to an LLM, while the first stage is still left to learning experts. This idea has previously been hinted at by Cai et al. [6], who propose using a question-answering system to automatically list potential misconceptions for a given topic. We extend this idea to have an LLM author the entire script for the tutoring system. This authoring process includes internal checks and guardrails, making the results accurate enough to be deployed without human validation. However, should validation be essential, we also describe a version that can be checked by humans to increase trust.

[0]Code and data publicly available: github.com/eth-lre/MWPTutor

[1]See Appendix A in [12] for an example script.

In our experiments, we find that our system is better at tutoring an AI-simulated student than instructed, but otherwise free-form GPT4. We also find that human evaluators judge that individual utterances are indeed following the specified pedagogy. However, manual analysis of the conversations shows that there is still some scope for improvement. In summary, our contributions in this paper are as follows:

- We adapt the Dialog Based tutoring framework used by ITS's to make it possible for an LLM to author it.
- We design MWPTutor, a tutor for math word problems which can function both with or without live calls to an LLM based on the above principles.
- We compare the performance of both versions of MWPTutor with that of GPT4 being used as a tutor both quantitatively and qualitatively.

2 BACKGROUND

Tutoring is an integral part of teaching where the teacher attempts to identify confusions or misconceptions a student might have, and rectify them. This is done in a way to minimize their future re-occurrence. Bloom et al. [3] showed that tutoring is extremely important for education, as a student receiving one-on-one tutoring can perform much better than a student only getting lectures. As opposed to lecturing, tutoring is more personalized, and therefore only feasible in smaller groups. This lower teacher-to-student ratio needs to be balanced by a higher number of tutors, which is expensive and not always feasible.

2.1 Rule-Based Systems

To combat the above scarcity, Intelligent Tutoring Systems (ITSs) were already being developed since the early 1980s [31]. The idea was to roleplay a human tutor with a computer program that could interact with the student and fulfill the requirement of personalized tutoring. This may not fully replace human tutors but often provides an affordable and scalable addition. An important aspect that differentiates ITSs from simply showing the students the problem solution is their ability to provide step-by-step feedback which promotes long-term learning as opposed to short-term problem-solving success. This was often implemented using rules that check for certain errors using string-matching and act on them with predefined templates. For example, LISP Tutor [1] uses 475 rules to account for common programming errors made by students.

With advancements in AI, ITSs transitioned to dialog as a method of tutoring, moving ITSs from being information-delivery systems to dialog-based agents. One of the key players in this field was the first version of AutoTutor [14], which replaced the rules in traditional ITSs with a full finite state transducer which they called the *Dialog Advancement Network*. This allows the students to interact in natural language and the tutor processes their inputs using methods like latent semantic analysis and attempts to respond using a set of predefined rules. Over the next two decades, various tutors such as the AutoTutor explored different domains, tutoring strategies, affective statements, etc. to become one of the most comprehensive ITSs (see [20] for the summary of developments).

With concepts and tutoring strategies getting more complex, writing the rules governing the finite state transducers quickly becomes time-consuming. Not knowing the student misconceptions in advance, domain experts have to come up with rules for numerous possibilities, introducing a different set of scaling issues. One hour of interaction with a student costs over 100 hours of domain expert time to author the ITS [15]. This is in addition to the time spent by learning scientists designing the pedagogy and dialog strategies. However, this is usually done only once for the topics in a subject area. As a result, this limits the AutoTutor to a small set of subject areas, restricting their impact. Nevertheless, there are many works in the space of rule-based ITS [23], which have been reasonably successful in their domains.

2.2 Large Language Models

Large Language Models (LLMs) are Transformer-based models that serve as general-purpose tools in natural language processing [4]. With recent improvements in LLMs and the availability of LLM-as-a-service models like GPT [22] and BARD [34], it has become easy to use LLMs as tutors by prompting them with the desired qualities of a tutor and then letting them interact with a student. Some works in the programming domain like GPTutor [7] and Phung et al. [25, 26] attempt to use GPT4 similar to information delivery systems. There have also been systems designed to function as Dialog agents, like Ruffle&Riley [30] which includes multiple GPT4 agents functioning with different prompts, and SPOCK [33] which attempts to distill out tutoring knowledge from GPT4 into a smaller LLM. A notable commercial usage of a LLM Tutor is seen in Khanmingo,[2] which uses the GPT4 backend for an interactive tutor.

While LLM tutors solve the scaling issues, it introduces a whole new set of complications, primarily related to its probabilistic output and proprietary models. The behavior of GPT4 can only be controlled by making changes to its initial prompt or by inserting prompts in between, making it hard to introduce the nuanced pedagogical strategies developed by learning scientists over the last two decades. Even for simpler strategies, its probabilistic nature leaves it open to being exploited by users who behave in unwanted ways, often referred to as jailbreak attacks [17]. Even for benign users, GPT4 can often hallucinate incorrect information and output it with confidence [10, 36] which may adversely affect the student. All this, in addition to their hard-to-understand nature, leads to a distrust of GPT4 among educators and policymakers, causing it to be banned in several US public schools.[3]

2.3 Our work

The main benefit of the fully human-authored setting comes from the fact that it is easy to control, easy to interpret, and therefore easy to trust. Most of these benefits come from the pedagogy design stage, and not from the work of the domain experts. The LLM-based approaches, on the other hand, mostly speed up the domain experts' tasks. To bring together the best of both worlds, we propose an expert-designed pedagogy to be assisted by an LLM in the dialogue part. The rest of this paper is structured as follows: Section 3 introduces our general design which applies to several domains, Section 4 describes the specific design of MWPTutor, Section 5 quantitatively evaluates MWPTutor, and Section 6 looks qualitatively at weaknesses of both GPT4 and MWPTutor.

[2] khanacademy.org/khan-labs
[3] bestcolleges.com/news/schools-colleges-banned-chat-gpt-similar-ai-tools

3 DESIGN PRINCIPLES

While this work mainly focuses on Math Word Problems tutoring, our broader goal is to present a paradigm that can be used across different subjects. Before the math-specific aspects, we first describe the model design principles. The output is the implementation of a pedagogical goal as a template state space for a finite state transducer similar to dialog advancement networks, which can then be filled by an LLM. For the pedagogical strategy, a good doctrine to follow can be *Get the student to the correct answer. If possible, do so without telling them the answer outright*

A finite state transducer is characterized by the combination of state space and a transition function. The **state space** can be seen as the memory of the finite state transducer, which stores one or more variables, each of which can take two or more values. In our case, we have two variables, the solution step and the strategy, which factorizes our state space into two subspaces, detailed in Section 3.1 and Section 3.2 respectively. The **transition function** specifies the next stage given the current state and the (student) input and is highly domain-dependent, but we provide examples as they come up alongside the state space descriptions

First, consider a math word problem and a history question to illustrate the possible range of topics which a tutor might apply.

> *[MWP] Mikhael buys two old coats, one at $200 and one at $250. After cleaning and fixing them, he sells them for a profit of 20% each. How much money did Mikhael make from this transaction?*[4]

> *[WW1] List 4 causes of the First World War.*

The optimal tutoring support differs between the two examples.

3.1 Solution Decomposition

To begin creating the state space for a finite state transducer, we first need to list the solutions for the problem, creating the **solution step space**. As a first design principle, we want to have the solution broken down into multiple parts. For the math word problem example, this could look like a tree:

> *Solution Step Space (MWP)*
>
> *Branch A*
> *A1 Calculate the total cost* $200 + 250 = 450$
> *A2 Calculate total profit* $450 \times 20/100 = 90$
>
> *Branch B*
> *B1 Calculate the profit on the first coat* $200 \times 20/100 = 40$
> *B2 Calculate the profit on the second coat* $250 \times 20/100 = 50$
> *B3 Add to get total profit* $40 + 50 = 90$

To complete the question, the student needs to perform A1 followed by A2. Alternatively, they can complete B1 and B2 in any order and then B3. We transition out of a step if the student arrives at the right-hand side of the equation in the (sub)question. These dependency structures could look very different based on the domain, like for the history question, for example:

[4] Mikhael spent $450, so a 20% profit would make him $90 from the transaction.

Solution step space (WW1)

- *C1 Tensions by European colonial exploitation.*
- *C2 Serbia wanted to create a pan-Balkan state causing tensions with Austrohungarians and Ottomans.*
- *C3 Austrohungarian Archduke Franz Ferdinand was assassinated by a Serbian nationalist.*
- *C4 The formation of the Triple Alliance and the responding Central Power Alliance polarized Europe.*
- *C5 German blank check assurance to Austria-Hungary provoked the Triple Alliance.*
- *C6 The German state was militarised.*

In this case, a correct answer requires the student to complete any 4 out of C1-C6 in any particular order to be considered successful. The transition function here is more complicated, but one way would be to check if all underlined key terms have been mentioned.

Splitting a problem into smaller parts is beneficial for both human learning [29, 32] and for LLM solvers [11, 21]. Getting an LLM to split the problem into smaller parts allows us to thereafter focus the LLM on smaller parts of the solution, which can also reduce context lengths. In AutoTutor terminology, these solution steps would correspond to "expectations".

3.2 Pedagogical Strategy

Having broken down a problem into steps, we now need to have a strategy to help the student achieve each step. The general idea is to start vague, hoping for the student to do as much of the work as possible, but then keep getting more specific until the student can complete the step. For the math word problem example, a toy strategy could be:

- *D1 Give out the solution step, ask for the calculation e.g.for A1, we would say "Can you calculate the total cost incurred?"*
- *D2 Read out the calculation to perform e.g. for A1, we would say "You have to add 200 and 250".*
- *D3 Give out the answer, e.g. for A1, we would say "The answer is 450".*

This set of pedagogical strategies forms our **strategy space**. We can transition from D1 to D2 and then to D3 if the conditions for transitioning to the next solution step are not met, at which point we loop in D3. For the history example, we have:

- *P1 Seek the next cause from the student e.g.irrespective of the step, we can say "What other cause can you think of?"*
- *P2 Ask for each missing keyword e.g.for C3 if we already have "assassination" from P1, we can ask "Who was Assasinated and by whom?"*
- *P3 Reveal the answer if the student fails to get it.*

The actual set of moves can be quite complex. For P2, we would need separate utterances for every possible combination of missing keywords. Ideally, these moves should be designed by learning scientists as was the case for AutoTutor. The overall state space is the cross-product of the solution state space and the strategy space. Figure 2 shows what the full state space of our MWP example could look like. For each element of the strategy space, we would require prompt templates that can be filled with the solution step and passed on to LLMs to generate the actual utterance. Running each prompt for every step in the solution space maps every element in the state space to an **utterance**. Not all utterances need to be unique, and some, like P1 above, might not necessitate an LLM to be generated.

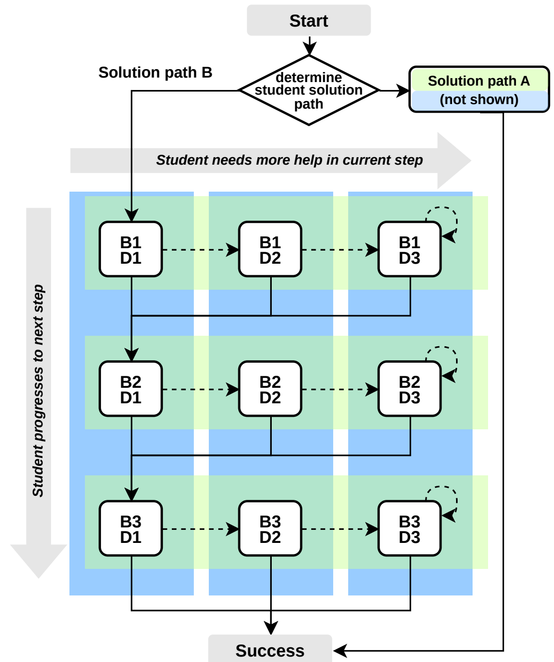

Figure 2: Full state space for our MWP toy example. Vertical blue rectangles show the strategy space and Horizontal green rectangles show the solution step space. Solid arrows indicate success in the current step, and dashed arrows indicate failure. The states for branch A are collapsed for clarity.

3.3 Scaffolding the LLM

While modern LLMs fine-tuned to follow natural language instructions are quite good at doing so, they can at times struggle with more complex tasks. Because of the probabilistic nature of LLMs, reattempting the same problem can lead to different answers. To take the best advantage of these facts, we want to have automated checks for every LLM turn in the strategy space, so calls can be repeated until some checks have been met. For example, D3 could require that the numerical answer (450 for A1) is included in the utterance, while P2 could require the missing keywords to be absent. These checks can take different forms depending on the domain, like limiting the length of an utterance or avoiding the use of more complex language.

3.4 The Trade-off between Reliability and Contextual Awareness

Once we have prompts for each utterance and relevant checks, we can query an LLM to generate the tutor utterances. Here we have a choice on when the generation is done. One option is to generate all possible utterances at design time, and storing them into the state space. This would be similar to what traditional ITSs would do. Doing this not only allows the tutor to run without an active internet connection, but having an exhaustive set of utterances stored allows stakeholders to trust the tutor without necessarily trusting the LLM used to generate the utterances. This is possible because one can reliably say that there would be no "new" errors or hallucinations since the LLM is not used after the design stage.

Despite these advantages, pre-generating means that all possible scenarios that can arise during the conversation need to be anticipated in advance, which is extremely hard to achieve. Therefore, such a model is likely to feel somewhat oblivious of the context of the conversation, which might reduce student engagement, which in turn, may reduce learning gains [5, 19, 24] The alternative to this, is to generate the utterances during the conversation. This allows us to include recent utterances from both the tutor and the student into the prompt, thereby generating a more context-aware response. Due to changing context, it is also possible to allow multiple utterances for a single state, allowing for more fine-grained feedback. Doing this, however, leaves the LLM open to hallucinations and jailbreaks as discussed earlier.

The choice we make depends of factors like how good we believe the backend LLMs are, how much the policymakers trust them, how complex is the problem etc. We do not believe that any of these options is better than the other, and the best option might be a combination of both.

4 MWPTutor

We now detail the design process of our system MWPTutor, an instantiation of the above principles. MWPTutor is adapted to help students solve math word problems. The scenario, originally defined by MathDial [18], involves a student approaching a tutor after unsuccessfully attempting to solve a math word problem. The student's incorrect solution is provided as a part of the input. GSM8k [9], the source for MathDial's problems includes a single step-by-step solution to each problem, which can be seen as a fulfillment of the solution step space. However, to not become constrained to the availability of such solutions, and to allow diverse solution paths, we restrict MWPTutor to only the final numerical answer to the problem, and generate the solutions ourselves. Further, to highlight the tradeoff mentioned in section 3.4, we shall design two slightly different versions MWPTutor^{live}, which is allowed to make live calls to the LLM, and MWPTutor^{cached} which is not. Note that unless otherwise specified, designs mentioned in this section apply to both versions.

4.1 Solution Decomposition

We instantiate the solution steps as short pieces of text describing a step in the math word problem, including a single mathematical calculation. We generate multiple solution paths with our backend LLM, with each step needing to contain a single calculation demarcated in double angular braces (e.g. $\langle\langle 30 + 43 = 73 \rangle\rangle$, see **??**). This requirement standardizes the information density of a step and also keeps things consistent with GSM8K which uses a similar notation. It also allows us to check for calculation errors and discard erroneous paths. Once multiple solution paths have been obtained, we discard the paths not leading to the final correct solution, and group the rest together using exact string match on the calculations into a "tree" of solution steps. Our running example, except for the angular brackets, fits this criterion as a tree with two branches at the root and no branches elsewhere.

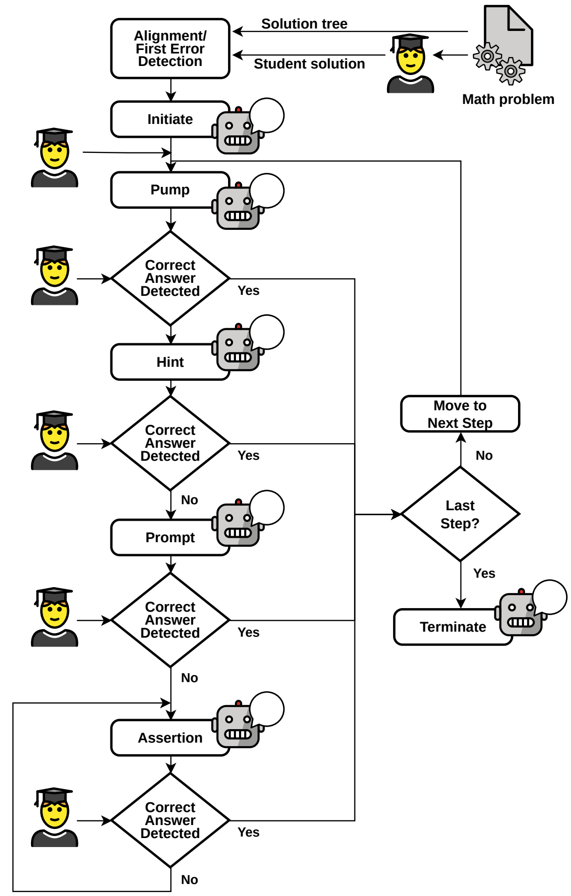

Figure 3: MWPTutor's state space as a flowchart. The symbol indicates student utterance inputs, while the symbol indicates model output. The solution step space is collapsed for clarity

4.2 Solution Alignment

Although not a part of design principles, we wish to not make the students repeat a step they have already performed correctly. To achieve this, we match steps in the original student solution with the closest path in the solution step space. While AutoTutor achieves this using latent semantic analysis and string-matching techniques, we found that modern counterparts, such as BERTScore [37] are surprisingly not an apt replacement, perhaps because GPT4 outputs very long step descriptions, and BERT does not weigh and compare numbers and mathematical operations well.

MWPTutor currently assumes that the student will present their solution with one step in each line. For each student step, we check if the right-hand side of any of the possible follow-up steps is present in it. If more than one follow-up matches, we pick the one with the highest BERTScore. We then proceed down that node of the solution tree and the next student step. If no follow-up matches, the current step is considered to be the first point of error. The pseudo-code for this is shown in Algorithm 1.

Algorithm 1 Initial solution alignment.
Input: GOLD and STUDENT solutions.
Output: aligned solution step, if exists

GoldSteps ⟵ gold.root
StudStep ⟵ student.head
while GoldSteps is not LeafNode **do**
  ▷ Filter out next solution steps whose RHS is in the current student step
  Matches ⟵ $\{c \mid c \in$ GoldSteps.children $\wedge$ FindNums(StudStep.text) $\in c$.rhs$\}$
  if Matches is empty **then** ▷ Nothing matches, this is the first point of error
    return (GoldSteps, StudStep)
  else
    ▷ Move to next step
    GoldStep ⟵ Max(Matches, key=λx : BertScore(x, StudStep.text))
    StudStep ⟵ StudStep.next
  end if
end while
return Error(*"Student has the right answer."*)

4.3 Pedagogical Strategy

To instantiate our pedagogy, we take inspiration from AutoTutor, where the primary sequence of steps is a pump→hint→prompt→ assertion (Section 2 of [13]). Graesser et al. in their work *Teaching Tactics in Autotutor* [12] define **hints** as "questions that lead the student to the desired information", and **assertions** (which they call elaborations in this work) as "succinctly assert the desired information." For prompts, they use fill-in-the-blanks type utterances leaving out a keyword. In our case, the "keyword" is the equation, so instead we explain the calculation to be performed in a natural language. These are preceded by a "pump", which is the generic utterance "What should be the next step?" The overall strategy then proceeds as follows:

Pump asks the student to propose the next step. If the student proposes the step correctly, we ask them to solve it.

Hint is only triggered if the student fails to propose a valid next step, or fails to execute the step they propose. It essentially gives out the step and asks the student for the calculation. This utterance should take the form of a question, and the only way to correctly answer the question should be to successfully execute the step.

Prompt is triggered if the student fails to execute the step after the hint. This should push the student closer to the answer, without actually revealing the answer.

Assertion is triggered if the student fails to answer even after the prompt. This should simply give out the answer, and have the student acknowledge it.

In our running example, hints, prompts, and assertions correspond to D1, D2, and D3 respectively. The pump is not included.

For scaffolding the LLM, both hints and prompts are required to avoid explicitly outputting the right-hand side of the equation in the step, with the hint additionally having to avoid saying any words that indicate arithmetic operators, currently "add, subtract, sum, multiply, product, divide, quotient." The assertion, on the other hand, must contain the right-hand side. For pumps, there is no use of an LLM, so we do not need a check.

For MWPTutor^{live}, we allow up to 3 prompts before moving to assertion. For both hints and prompts, the LLM is given the steps the student has correctly solved so far and the last 4 utterances in the dialogue (2 each for the student and the tutor) in addition to the solution step and strategy (exact instructions for Hints and Prompts and can be found in our code)

4.4 Answer and Path Detection

To execute the pedagogical strategies described above, we need to detect if the student has given a correct answer. Further, the hint state should only trigger if the student fails to propose a correct follow-up. For this, we need to have a way to match paths. We run into the same matching problems described in Section 4.2, and in case of MWPTutor^{cached}, resort to similar basic techniques. The correct answer detection algorithm is shown in Algorithm 2. For states other than pump, it takes a single reference step and checks if its RHS is present in the student utterance. For the pump state, all possible followups of the last step are passed as reference steps. If it can find the RHS of any of these in the student utterance, it marks that as the next step and proceeds to the next step. If not, it considers a path match to have occurred if every operand on the left-hand side of some step is present. If multiple steps fulfill this, or no step does, we pick the one with the best BertScore alignment to the student utterance.

Since MWPTutor^{live} is allowed to make live calls to the LLM, we make use of it for both correct answer detection, and picking the best path to take when in the "pump" stage. For correct answer detection, we use the RHS match as an additional criterion, while for picking the best path, we rely entirely on the LLM's judgement. The fact that MWPTutor^{cached} currently ignores all non-numeric content in the student response is one of its major limitations, as it can miss a large number of "correct" utterances from the student, but we defer the resolution of this to future work.

Algorithm 2 Correct Answer Detection
Inputs: CurrentSteps from Solution step space, and Student Utterance
Output: Degree of the match, Best matching step

UtteranceNums ⟵ FindNums(Utterance)
  ▷ CurrentSteps is a singleton except for "pump."
for $r \in$ CurrentSteps **do**
  ▷ Go to next solution step
  if r.rhs $\in$ UtteranceNums **then return** (*"Correct"*, r)
end for
PartialMatches ⟵ $\langle r \mid r \in$ CurrentSteps $\wedge$ r.lhs $\subseteq$ UtteranceNums$\rangle$
if PartialMatches is empty **then**
  return (*"No match"*, Max(CurrentSteps, Sort=Bert(x, Utterance)))
else ▷ Only relevant for "pump".
  return (*"Part match"*, Max(PartialMatches, Sort=Bert(x, Utterance)))
end if

4.5 Putting it All Together

Figure 3 summarises the overall state space of MWPTutor. It can be seen as a simplified version of the state space of AutoTutor (see Figure 1 in [6]). The expectation matching is implicitly carried out by the solution aligner and the pump state. To obtain to full finite state transducer structure, one must roll out the states along the solution space. Excluding the unlikely case where the student fails to repeat the answer after the assertion state, the overall conversation should not last more than $2 + 4 \times S$ utterance pairs for MWPTutor^{cached}, and $2 + 6 \times S$ for MWPTutor^{live}, where S is the number of steps.

5 EXPERIMENTS

In this section, we the performance of MWPTutor and GPT4 in terms of how well they can follow our previously mentioned doctrine. We test both models in the interactive setting proposed by Macina et al. [18] which allows us to safely compare imperfect models without having a negative impact on human users.

5.1 Setup

5.1.1 Setting. The interactive tutoring setting was proposed in MathDial as a faster, cheaper, and safer alternative to human experiments, where instead of interacting with human students, a Tutoring System interacts with another LLM role-playing a student. In their data collection experiments, MathDial found that most teachers believed that the student LLM behaved very much like an actual student in most cases. We adopt their setting as is for our experiment section.

Note that for an LLM, the interactive tutoring setting is somewhat related to the MWP solver setting often used as a metric to measure the reasoning abilities of said LLM, but there are significant differences. Unlike the MWP solver setting, the final answer is already provided, so the LLM does not need to come up with it. The difficulty arises from the fact that this LLM needs to get the student LLM to get to this answer, and the student LLM is prompted to not budge from its starting wrong answer without good reason. Despite these differences, some skills like Math Reasoning Abilities transfer between the two settings, so we can expect the final performances in both settings to correlate with each other.

5.1.2 Models. At the time of testing, GPT4[5] was considered to be one of the best-performing LLMs, so we use it both as a backend for the two MWPTutor's and as the baseline single prompt LLM. To best align the single prompt GPT4 with the MWPTutor doctrine, we ask it to roleplay a tutor, setting the primary objective of getting to the correct answer as the goal of the conversation, and the secondary objective of not revealing the answer as a personality trait. Further, to test the potential of an open-source LLM as a backend, we have a second version of MWPTutor with a MetaMath-13B [35]. MetaMath is a version of Meta AI's open source LLM LLaMa[6] fine-tuned to solve math word problems, and is, to the best of our knowledge, the best open LLM for solving math word problems. However, since MWPTutor^{live} would require the Backend to run throughout the conversation, which is unsustainable for a locally-run model, we only use this with MWPTutor^{cached}. We shall refer to the live model as MWPTutor$^{\text{live}}_{\text{GPT4}}$, the cached models as MWPTutor$^{\text{cached}}_{\text{GPT4}}$ and MWPTutor$^{\text{cached}}_{\text{LLaMa}}$ based on the backend, and the single prompt model simply as GPT4 hereafter.

5.1.3 Datasets. To implement MWPTutor, we require a dataset with math word problems and their solutions, as well as incorrect student solutions. We make use of the test sets of two datasets, namely MathDial [18] and MultiArith. MultiArith consists of 180 simple Math Word Problems that are usually solvable in 1 to 3 steps. GPT3 was already able to solve over 90% of these problems with the right prompts [16], and in our tests, we found that GPT4 was able to get a perfect score in the MWP solver setting. With this dataset, we want to test if we can achieve the same performance in the interactive tutoring setting. This dataset does not provide any wrong solutions needed to instantiate the student model, so we make use of errors made by a MetaMath solver as the original student solution where possible, and leave the remaining questions as "No idea how to begin".

MathDial consists of 45 problems from the pre-existing GSM8k [9]. Problems in GSM8k are meant to be solvable by a bright middle school student using only basic arithmetic operators (+, −, ×, ÷), taking up to 8 steps to solve. Of these, MathDial picks problems that have official solutions taking 5 steps or fewer, that GPT3.5 consistently failed to solve. This makes MathDial a significantly harder dataset to teach, and results on this dataset are expected to show the differences between the models more clearly.

5.1.4 Metrics. Keeping in line with our doctrine, our primary metric of evaluation is the **Success Rate (S)**, which is measured as the fraction of conversations where the student says the correct answer eventually. The secondary metric is **Telling Rate (T)**, which is the fraction of conversations where the tutor did not reveal the answer. The tutor is said to have revealed the answer when they say the final answer *before* the student has said it, and the student succeeds in getting to the answer thereafter. Both these metrics were originally defined by Macina et al. [18], but they do not offer a way of combining them into a single score. It is quite easy to increase success at the cost of increased telling. Therefore we also calculate the **Adjusted Success (S-T)** rate as the fraction where the student succeeded without having been told the answer and use the harmonic mean between this number and the success rate as an overall **Tutoring Score ($\frac{2S(S-T)}{2S-T}$)**. We also offer utterance level evaluations in Section 5.3, but since we are unaware of the utterance level strategies from GPT4, we do it only for our model, and the metrics for the same are defined later.

5.2 Conversation Evaluation

We evaluate both success and telling for the MultiArith dataset by directly looking for the answer in student and tutor utterances respectively, and calculate the overall scores based on that. The top half of Table 1 summarises our results. All models are able to transfer their near-perfect solving scores to near-perfect tutoring scores. Both GPT4 and MWPTutor$^{\text{live}}_{\text{GPT4}}$ have no telling, while the telling rate of the MWPTutor^{cached} models is also almost negligble. All 3 versions of MWPTutor have a perfect success rate, with GPT4 not far behind. These results demonstrate that all our models are aligned with our doctrine, and are able to follow it well on a simpler dataset. It must be noted that the metric is noisy since it only requires the number to match, with no restriction on the context, e.g. in the problem "How many groups of 9 can be made out of 5 boys and 40 girls?" if the tutor mentions there are 5 boys, we would count it as telling as the final answer happens to be 5. This means that the real rates could be somewhat lower for all 3 models, and the difference between them is not significant. Nevertheless, such errors are unlikely to be frequent, and all models work reasonably well on these problems.

For the harder MathDial dataset, we wanted to avoid matching errors as described earlier, and so have human annotators mark out utterances where the student or the tutor says the final answer.

[5] gpt-4-1106-preview queried in January 2024
[6] llama.meta.com

		MWPTutor^{cached} LLaMa	MWPTutor^{cached} GPT4	MWPT.^{live} GPT4	GPT4
MultiArith	Success Rate	1.00	1.00	1.00	0.98
	Telling Rate	0.01	0.01	0.00	0.00
	Adjusted Success	0.99	0.99	1.00	0.98
	Tutoring Score	**0.99**	**0.99**	**1.00**	**0.98**
MathDial	Success Rate	0.91	0.96	1.00	0.82
	Telling Rate	0.33	0.11	0.00	0.00
	Adjusted Success	0.58	0.84	1.00	0.82
	Tutoring Score	**0.71**	**0.90**	**1.00**	**0.82**

Table 1: Performance of MWPTutor and GPT4 on MultiArith (180 problems) and MathDial (45 problems).

Each conversation was annotated by 3 annotators, and we took the majority in cases where they disagreed.

The bottom half of Table 1 summarises our result. Both GPT4 and MWPTutor$^{\text{live}}_{\text{GPT4}}$ maintain their 0% telling rate, but MWPTutor$^{\text{live}}_{\text{GPT4}}$ is the only model able to maintain its 100% success rate. This clearly establishes MWPTutor$^{\text{live}}_{\text{GPT4}}$ as the model of choice should live access to GPT4 not be an issue, as discussed in Section 3.4.

Of the cached models, MWPTutor$^{\text{cached}}_{\text{GPT4}}$ gets 6 more problems correct than GPT4, and 1 more when adjusted for telling. This gives MWPTutor$^{\text{cached}}_{\text{GPT}}$ an advantage on GPT4 in terms of the final score. MWPTutor$^{\text{cached}}_{\text{LLaMa}}$ also gets a higher success rate than GPT4, which is somewhat surprising since its backend model is significantly weaker than GPT4 in the MWP Solver setting [35]. This increased success rate however comes at a cost of a very high telling rate, giving it a significantly poorer tutoring score overall. Based on this we do not recommend the use of MetaMath as the backend model unless GPT4-like models need to be avoided for some external reason.

One thing worth mentioning here is that once problematic questions have been identified for either version of MWPTutor^{cached}, one can have human annotators redo only those questions, thereby increasing the success rate, something that is not possible for GPT4. MWPTutor$^{\text{live}}_{\text{GPT4}}$ is somewhere in the middle, as we can only "fix" the solution step space.

5.3 Utterance Evaluation

A major shortcoming of AI-on-AI-tutoring is that the AI student may not be affected by the sometimes subpar tutor responses. To better understand the quality of utterances both in terms of them executing the pedagogical strategy and in term of their context awareness, we hired human evaluators who worked in the education sector and asked them to evaluate utterances made by MWPTutor$^{\text{cached}}_{\text{GPT4}}$ and MWPTutor$^{\text{live}}_{\text{GPT4}}$ for MathDial. Because the pumps are generic and very similar to each other and assertions essentially boil down to reading out the answers, we focus only on the hints and prompts in our evaluation.

Across the 45 conversations, MWPTutor$^{\text{cached}}_{\text{GPT4}}$ and the student model went over 135 solution steps Of these 102 went to the hint state, while 29 reached the prompt state. In the case of MWPTutor$^{\text{live}}_{\text{GPT4}}$, we had 129 steps total, of which 88 went to a prompt hint state, 42 went to the prompt stage. 16 of these would then use a second prompt and 12 would use a third, giving us a total of 70 prompts. We presented each of these 289 utterances to human evaluators, with the past conversation, the original math word problem, and the current solution step as context. We boil down the requirements set on hints and prompts set in Section 4.3 to the following binary criteria which we asked the evaluators to evaluate the utterances on:

- **Grammatical**: Is linguistically and grammatically correct.
- **Not Reveal**: Does not reveal the final answer for the step.
- **Relevant** (hints only): Addresses the step and the only way to correctly answer the question is to complete the current step.
- **Not Redundant** (hints only): Is not redundant to what the student already said in the previous step ie it makes sense to ask this question after the previous response.
- **Specific** (prompts only): Is specific to the step, and not just a vague lead-on statement.
- **Helpful** (prompts only): Can help the student make progress in the current step, given the previous student's utterance.

The annotators were paid the Prolific suggested wage of 10 GBP/hour. Snapshots of our annotation user interface and some example annotations can be found on github. summarise the responses from the evaluators. Almost all utterances are considered Grammatical by the evaluators, which is in line with the known grammatical prowess of GPT4. It is unclear why there is a slightly higher count of non-grammatical utterances in MWPTutor$^{\text{live}}_{\text{GPT4}}$, but it could be potentially due to annotators finding the more eloquent responses of GPT4 harder to judge grammatically. Most of the utterances are also marked to be non-revealing, which is expected given that any proposed utterances that directly revealed the answer were assigned to be regenerated. MWPTutor$^{\text{live}}_{\text{GPT4}}$ performs slightly better here. The evaluators further seem to agree that most hints were relevant to the step and that most prompts are specific, implying that they do what is appropriate given the current solution step. Again, MWPTutor$^{\text{live}}_{\text{GPT4}}$ has an edge in this aspect. Of the two of its prompts found to be non-specific, one is a second prompt and one is a first prompt, so all third prompts are considered specific

The biggest potential weakness of MWPTutor^{cached} comes in the form of its utterances making sense in the context of previous student utterances due to issues discussed in Section 4.4, which is where the final two metrics come in. In this regard, all prompts from MWPTutor$^{\text{cached}}_{\text{GPT4}}$ are considered to be helpful by a majority of evaluators which is a great sign. Surprisingly, MWPTutor$^{\text{live}}_{\text{GPT4}}$ ends up giving 5 prompts that are not considered helpful. One reason why this happens could be the fact that there are more prompts allowed for each step, making individual prompts less helpful. Regardless, it is still a small number.

Hints, however, are found to be redundant in almost a third of cases for MWPTutor$^{\text{cached}}_{\text{GPT4}}$, showing that handling of student's responses to pumps could be better. MWPTutor$^{\text{live}}_{\text{GPT4}}$ brings this number much lower, but still with 12 redundant Hints, it is the second largest red bar in the graph, only behind its cached counterpart. It is clear that MWPTutor$^{\text{cached}}_{\text{GPT4}}$'s heuristic of matching the left-hand side is too demanding, and highly likely to ignore the

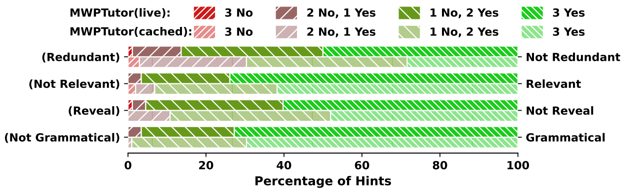

Figure 4: Summary of Human evaluation for Hints from MWPTutor. The total number of Hint samples is 102. Each sample was evaluated by 3 annotators in 4 categories.

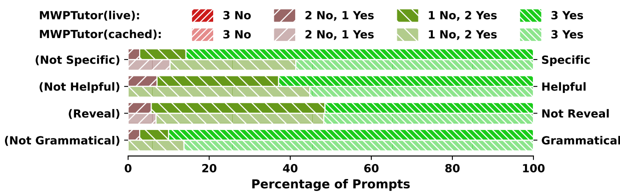

Figure 5: Summary of Human evaluation for Prompts from MWPTutor. The total number of Prompts samples is 29 for MWPTutor$^{\text{cached}}_{\text{GPT4}}$ and 70 for MWPTutor$^{\text{live}}_{\text{GPT4}}$. Each sample was evaluated by 3 annotators in 4 categories

student even if they propose a valid next step to the problem, which can break student immersion.

It should be noted, however that 33 of the 135 steps did not proceed to the hint state for MWPTutor$^{\text{cached}}_{\text{GPT4}}$, implying more than half of the correct proposals by the student are being identified as such. Curiously, the number of non-redundant hints is almost the same for the two models. Further, in a vast majority of cases where a correct proposal is misidentified, there is disagreement between human evaluators indicating there might be some merit in the rejection, though this is also the case for a many of correct rejections.

Overall these evaluations show that utterances generated for MWPTutor$^{\text{cached}}_{\text{GPT4}}$ by the GPT4 backend are consistent with the strategy definitions, and the transition function uses them reasonably well. MWPTutor$^{\text{live}}_{\text{GPT4}}$ further increases the efficacy of the transition function. Both models, however, have scope for improvement.

6 QUALITATIVE ASSESSMENT OF WEAKNESSES

While all our models do quite well on quantitative metrics as seen above, these metrics do not fully grasp what it takes to make a good tutor. To better understand the preparedness of our models, we went over all 45 conversations that happened on the MathDial dataset, for GPT4 and both MWPTutor$_{\text{GPT4}}$ models, and summarize our findings here.

Firstly, **MWPTutor$^{\text{cached}}_{\text{GPT4}}$ tends to sound monotonous and dry.** Part of this is because of the pre-set format for utterances like all hints are hardcoded to begin with "Tell me this instead:," but this can easily be resolved by having it sample from several rephrasings of the same utterance. More difficult to fix are the repetitive strategy and the lack of finer feedback or any affective statements, both of which are there by design. Introducing finer feedback requires a more fine-grained analysis of student utterances, which we found off-the-shelf NLP methods to be incapable of doing while maintaining a high degree of accuracy. Affective statements, like encouragements, challenges, etc. would further require accurate judgment of student confidence and correctness, which we chose to avoid due to it being a possible failure node. **GPT4 and MWPTutor$^{\text{live}}_{\text{GPT4}}$ sound a lot more natural,** including finer feedback and probing, and even affective statements, although they are exclusively positive. The ability of MWPTutor$^{\text{live}}_{\text{GPT4}}$ in achieving appropriate affect statements might indicate that we might have been overzealous in not having affective statements for MWPTutor^{cached}, and a similar effect might be achievable using a smaller language model.

Further, since it matches correct answers with limited attention to context, **MWPTutor$^{\text{cached}}_{\text{GPT4}}$ can sometimes get false positives, and inadvertently reveal the answer**. Consider the following:

> ***Tutor:*** *Tell me this instead: What is the total amount of feed needed for all the chickens for one day if each chicken requires 3 cups of feed?*
> . . .
> ***Student:*** *Sure, 3 cups of feed x 20 chickens = 60 cups of feed per meal. 60 cups x 3 meals = 180 cups of feed for the day. . . .*
> ***Tutor:*** *60 is correct, well done. What is the next step?*

Here, the student gives the wrong answer of 180, but MWPTutor mistakes the intermediate calculation of 60 as the correct answer, and effectively reveals it to the student. **GPT4 is able to understand context usually**, although it is far from perfect. GPT4 fails much worse on this question too. **MWPTutor$^{\text{live}}_{\text{GPT4}}$, on the other hand, tends to have a false negatives**, perhaps because needing approval from both the LLM and the RHS checker is a bit too strict. Further, unlike GPT4, it still cannot account for student solving multiple steps in one utterances, so at times it might look like it is holding the student back. In general, it handles the false negatives well, by either asking the student to repeat or revise steps or asking for clarifications, but ideally, we do not want to prolong conversations anymore than needed.

While it does not have the one-step per utterance restriction, **GPT4 still tends to overcomplicate questions or make redundant calculations**. In one question, where the student failed to convert seconds into minutes, GPT4 gets the student to convert the seconds to hours first, and then back to minutes. This introduced precision errors, which it had to fix by rounding the final answer. In another question GPT4 has a student recalculate a value she had already calculated in a previous step. In yet another question GPT4 makes the student calculate the weekly rate of weight loss, when a simple multiplication of time period by 4 would have yielded the answer. The conversation in Figure 1 is also an example of where this happens in the simpler MultiArith dataset. Due to its more contained and well-structured state space, **MWPTutor$^{\text{cached}}_{\text{GPT4}}$ is less likely to run into this issue**, and handles the first two cases mentioned here pretty well, though it makes the same error in the last case.

The biggest problem with GPT4, continues to be that **GPT4 can lead the student to a wrong answer, sometimes even leading them away from the right answer**. In all 8 cases where GPT4 fails to get the student to the correct answer, it outright accepts a wrong answer as correct. In a further 2 conversation, including the one regarding chicken feed described earlier, GPT4 misdirects the student onto the wrong path but somehow manages to get them to say the correct answer. Finally, in one question, GPT4 rejects a correct equation, and then goes off in a tangential direction, until the maximum limit of 60 utterances is reached (this being the only instance where the utterance limit comes into play). It must be reiterated here that the final correct answer is given to GPT4 as part of its prompt.

All MWPTutor models by design will never okay a wrong answer. Of MWPTutor$^{\text{cached}}_{\text{GPT4}}$'s failures one was a result of inadvertently revealing the final answer, while the other seems to have been marked as incorrect by the annotators because the tutor resorts to revealing rather quickly. Beyond these, in one question it accepts an answer of −40 instead of 40, but this can be chalked down to the fact that the question was asking for the "difference" between two quantities, and did not make it clear whether the sign was required (the first quantity was smaller. Also GPT4 makes the same error. Both are marked as correct by annotators). We would like to clarify that, **all MWPTutor models, especially MWPTutor^{live} can also hallucinate**, even if it did not happen in any of these conversations to the best of the authors' judgement (We attempted to have the evaluators annotate the existence of wrong methods, but found too much disagreement in their responses to consider them viable).

Overall, we conclude that despite good showing on metrics, both methods are still lacking in quality, and decide to not test them on real students until we can significantly improve them.

7 DISCUSSION AND FUTURE WORK

We have stated throughout the paper that although self-sufficient, MWPTutor is still a prototype, and we explained in Section 6 that we still have a long way to go before making MWPTutor viable to be used on real students. In this section, we outline some directions for future work that could bring us closer to these goals.

7.1 Improving the Solution Step Space

The current solution step space takes the shape of a tree where solutions can only diverge from each step. But this is non-realistic as we know that all paths must lead to the same final answer and hence need to converge at some point. Further, the order of carrying out certain steps could be a partial order rather than a total order. This would currently be represented by listing out all possible orders as separate paths, but this causes a blowup in the number of steps. Adding a richer structure to the Solution graph could vastly reduce the size of the solution space.

In addition, each step of the solution should be made to follow a more standardized format. Having a more precise definition of the step would allow us to use LSA-like techniques to improve our correct answer detection. Pre-defining possible errors for each step would allow us to prepare to handle said errors. We can also have descriptions of each quantity in the LHS to give more pointed hints.

7.2 Improving the Strategy Space

The current strategy space for the cached models is very linear, and cannot react well to exact student responses. Once we have a better solution step space, we can look for different types of errors and diverge into different Prompts depending on the exact error should the student fail to get the answer correct after the Hint state. There can even be more than one Prompt revealing different amounts of information based on the complexity of the current step. Another thing entirely missing is the "Problematization" strategies described by Reiser [28] and also used by MathDial. While most of these would mainly benefit the cached versions, these can also be made into more specific prompts for the live version. Alongside this, the matching requirements for solution alignment can be made stricter to better know where to start applying our intervention. This could also come with relaxing the format requirement of "one step per line" on the student solution. Further, it would be interesting to add in methods to detect when the student ends up completing multiple steps at once. All these would shorten the average lengths of the conversation to balance more steps in the strategy space.

Alternatively, matching can be done using smaller specialized LLMs trained for this task, and this would also apply to correct answer detection and path matching

8 CONCLUSION

In this paper, we propose using LLMs to take over the authoring of the state space for an AutoTutor-like model to minimize human time requirements and build a sample system to showcase our proposal. We find that our model performs better than a Single Prompt GPT4 model in terms of automatic metrics while maintaining full interpretability and modularity in its state space which GPT4 lacks. Our model also holds up to the desired qualities in its utterances in human evaluation, showing there is high potential in this direction. But at the same time, it still faces issues that become more apparent when going through conversations manually, indicating that there is a lot of work still to be done.

These issues can be fixed by better LLMs and more sophisticated techniques outlined in future work. Several potential steps have already been outlined, and we hope to soon have better versions of MWPTutor and other similar systems that can realize one-on-one tutoring at scale.

ACKNOWLEDGMENTS

This work was partly funded by the ETH-EPFL Joint Doctoral Program in the Learning Sciences, a grant from the Swiss National Science Foundation (Project No. 197155) and a Responsible AI grant by the Haslerstiftung. We extend our gratitude to Jakub Macina from the ETH AI Center, Nico Daheim from UKP Lab, TU Darmstadt, and Grégoire Farquet from Department of Architecture, ETH Zürich for their invaluable feedback.

REFERENCES

[1] John R. Anderson, C. Franklin Boyle, and Brian J. Reiser. 1985. Intelligent Tutoring Systems. *Science* 228, 4698 (1985), 456–462. https://doi.org/10.1126/science.228.4698.456 arXiv:https://www.science.org/doi/pdf/10.1126/science.228.4698.456

[2] David Baidoo-Anu and Leticia Owusu Ansah. 2023. Education in the era of generative artificial intelligence (AI): Understanding the potential benefits of ChatGPT in promoting teaching and learning. *Journal of AI* 7, 1 (2023), 52–62. https://dergipark.org.tr/en/pub/jai/issue/77844/1337500

[3] Benjamin S Bloom. 1984. The 2 sigma problem: The search for methods of group instruction as effective as one-to-one tutoring. *Educational researcher* 13, 6 (1984), 4–16. https://journals.sagepub.com/doi/abs/10.3102/0013189X013006004

[4] Rishi Bommasani, Drew A. Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S. Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, Erik Brynjolfsson, S. Buch, Dallas Card, Rodrigo Castellon,

Niladri S. Chatterji, Annie S. Chen, Kathleen A. Creel, Jared Davis, Dora Demszky, Chris Donahue, Moussa Doumbouya, Esin Durmus, Stefano Ermon, John Etchemendy, Kawin Ethayarajh, Li Fei-Fei, Chelsea Finn, Trevor Gale, Lauren E. Gillespie, Karan Goel, Noah D. Goodman, Shelby Grossman, Neel Guha, Tatsunori Hashimoto, Peter Henderson, John Hewitt, Daniel E. Ho, Jenny Hong, Kyle Hsu, Jing Huang, Thomas F. Icard, Saahil Jain, Dan Jurafsky, Pratyusha Kalluri, Siddharth Karamcheti, Geoff Keeling, Fereshte Khani, O. Khattab, Pang Wei Koh, Mark S. Krass, Ranjay Krishna, Rohith Kuditipudi, Ananya Kumar, Faisal Ladhak, Mina Lee, Tony Lee, Jure Leskovec, Isabelle Levent, Xiang Lisa Li, Xuechen Li, Tengyu Ma, Ali Malik, Christopher D. Manning, Suvir P. Mirchandani, Eric Mitchell, Zanele Munyikwa, Suraj Nair, Avanika Narayan, Deepak Narayanan, Benjamin Newman, Allen Nie, Juan Carlos Niebles, Hamed Nilforoshan, J. F. Nyarko, Giray Ogut, Laurel Orr, Isabel Papadimitriou, Joon Sung Park, Chris Piech, Eva Portelance, Christopher Potts, Aditi Raghunathan, Robert Reich, Hongyu Ren, Frieda Rong, Yusuf H. Roohani, Camilo Ruiz, Jack Ryan, Christopher R'e, Dorsa Sadigh, Shiori Sagawa, Keshav Santhanam, Andy Shih, Krishna Parasuram Srinivasan, Alex Tamkin, Rohan Taori, Armin W. Thomas, Florian Tramèr, Rose E. Wang, William Wang, Bohan Wu, Jiajun Wu, Yuhuai Wu, Sang Michael Xie, Michihiro Yasunaga, Jiaxuan You, Matei A. Zaharia, Michael Zhang, Tianyi Zhang, Xikun Zhang, Yuhui Zhang, Lucia Zheng, Kaitlyn Zhou, and Percy Liang. 2021. On the Opportunities and Risks of Foundation Models. *ArXiv* (2021). https://crfm.stanford.edu/assets/report.pdf

[5] Colin Bryson and Len Hand. 2007. The role of engagement in inspiring teaching and learning. *Innovations in Education and Teaching International* 44, 4 (2007), 349–362. https://doi.org/10.1080/14703290701602748 arXiv:https://doi.org/10.1080/14703290701602748

[6] Zhiqiang Cai, Xiangen Hu, and Arthur C. Graesser. 2019. Authoring Conversational Intelligent Tutoring Systems. In *Adaptive Instructional Systems*, Robert A. Sottilare and Jessica Schwarz (Eds.). Springer International Publishing, Cham, 593–603. https://link.springer.com/chapter/10.1007/978-3-030-22341-0_46

[7] Eason Chen, Ray Huang, Han-Shin Chen, Yuen-Hsien Tseng, and Liang-Yi Li. 2023. GPTutor: A ChatGPT-Powered Programming Tool for Code Explanation. In *Artificial Intelligence in Education. Posters and Late Breaking Results Workshops and Tutorials, Industry and Innovation Tracks, Practitioners, Doctoral Consortium and Blue Sky - 24th International Conference, AIED 2023, Tokyo, Japan, July 3-7, 2023, Proceedings (Communications in Computer and Information Science, Vol. 1831)*, Ning Wang, Genaro Rebolledo-Mendez, Vania Dimitrova, Noboru Matsuda, and Olga C. Santos (Eds.). Springer, 321–327. https://doi.org/10.1007/978-3-031-36336-8_50

[8] Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane Gu, Zhuyun Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Alex Castro-Ros, Marie Pellat, Kevin Robinson, Dasha Valter, Sharan Narang, Gaurav Mishra, Adams Yu, Vincent Zhao, Yanping Huang, Andrew Dai, Hongkun Yu, Slav Petrov, Ed H. Chi, Jeff Dean, Jacob Devlin, Adam Roberts, Denny Zhou, Quoc V. Le, and Jason Wei. 2022. Scaling Instruction-Finetuned Language Models. arXiv:2210.11416 [cs.LG]

[9] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training Verifiers to Solve Math Word Problems. *arXiv preprint arXiv:2110.14168* (2021). https://arxiv.org/abs/2110.14168

[10] Chenhang Cui, Yiyang Zhou, Xinyu Yang, Shirley Wu, Linjun Zhang, James Zou, and Huaxiu Yao. 2023. Holistic Analysis of Hallucination in GPT-4V(ision): Bias and Interference Challenges. arXiv:2311.03287 [cs.LG]

[11] Shehzaad Dhuliawala, Mojtaba Komeili, Jing Xu, Roberta Raileanu, Xian Li, Asli Celikyilmaz, and Jason Weston. 2023. Chain-of-Verification Reduces Hallucination in Large Language Models. arXiv:2309.11495 [cs.CL]

[12] Arthur Graesser, Natalie Person, and Derek Harter. 2000. Teaching Tactics in AutoTutor. (01 2000). http://users.sussex.ac.uk/~bend/its2000/graesser.pdf

[13] Arthur C. Graesser. 2016. Conversations with AutoTutor Help Students Learn. *International Journal of Artificial Intelligence in Education* 26, 1 (01 Mar 2016), 124–132. https://doi.org/10.1007/s40593-015-0086-4

[14] Arthur C. Graesser, Kurt VanLehn, Carolyn P. Rose, Pamela W. Jordan, and Derek Harter. 2001. Intelligent Tutoring Systems with Conversational Dialogue. *AI Magazine* 22, 4 (Dec. 2001), 39. https://doi.org/10.1609/aimag.v22i4.1591

[15] Kenneth R. Koedinger, Vincent Aleven, Neil Heffernan, Bruce McLaren, and Matthew Hockenberry. 2004. Opening the Door to Non-programmers: Authoring Intelligent Tutor Behavior by Demonstration. In *Intelligent Tutoring Systems*, James C. Lester, Rosa Maria Vicari, and Fábio Paraguaçu (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 162–174. https://link.springer.com/chapter/10.1007/978-3-540-30139-4_16

[16] Takeshi Kojima, Shixiang (Shane) Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large Language Models are Zero-Shot Reasoners. In *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (Eds.), Vol. 35. Curran Associates, Inc., 22199–22213. https://proceedings.neurips.cc/paper_files/paper/2022/file/8bb0d291acd4acf06ef112099c16f326-Paper-Conference.pdf

[17] Yi Liu, Gelei Deng, Zhengzi Xu, Yuekang Li, Yaowen Zheng, Ying Zhang, Lida Zhao, Tianwei Zhang, and Yang Liu. 2023. Jailbreaking ChatGPT via Prompt Engineering: An Empirical Study. arXiv:2305.13860 [cs.SE]

[18] Jakub Macina, Nico Daheim, Sankalan Pal Chowdhury, Tanmay Sinha, Manu Kapur, Iryna Gurevych, and Mrinmaya Sachan. 2023. MathDial: A Dialogue Tutoring Dataset with Rich Pedagogical Properties Grounded in Math Reasoning Problems. In *The 2023 Conference on Empirical Methods in Natural Language Processing*. https://openreview.net/forum?id=fyza2OQ9NI

[19] Florence Martin and Doris Bolliger. 2018. Engagement Matters: Student Perceptions on the Importance of Engagement Strategies in the Online Learning Environment. 22 (03 2018), 205–222. https://doi.org/10.24059/olj.v22i1.1092

[20] Benjamin Nye, Arthur Graesser, and Xiangen Hu. 2014. AutoTutor and Family: A Review of 17 Years of Natural Language Tutoring. *International Journal of Artificial Intelligence in Education* 24 (12 2014). https://doi.org/10.1007/s40593-014-0029-5

[21] Maxwell Nye, Anders Andreassen, Guy Gur-Ari, Henryk Witold Michalewski, Jacob Austin, David Bieber, David Martin Dohan, Aitor Lewkowycz, Maarten Paul Bosma, David Luan, Charles Sutton, and Augustus Odena. 2021. Show Your Work: Scratchpads for Intermediate Computation with Language Models. https://arxiv.org/abs/2112.00114

[22] OpenAI. 2023. GPT-4 Technical Report. arXiv:2303.08774 [cs.CL]

[23] José Paladines and Jaime Ramirez. 2020. A Systematic Literature Review of Intelligent Tutoring Systems With Dialogue in Natural Language. *IEEE Access* 8 (2020), 164246–164267. https://doi.org/10.1109/ACCESS.2020.3021383

[24] Jim Parsons and Leah Taylor. 2011. Improving Student Engagement. *Current Issues in Education* 14, 1 (May 2011). https://cie.asu.edu/ojs/index.php/cieatasu/article/view/745

[25] Tung Phung, JosÃ© Cambronero, Sumit Gulwani, Tobias Kohn, Rupak Majumdar, Adish Singla, and Gustavo Soares. 2023. Generating High-Precision Feedback for Programming Syntax Errors using Large Language Models. In *Proceedings of the 16th International Conference on Educational Data Mining*, Mingyu Feng, Tanja KÃ¤ser, and Partha Talukdar (Eds.). International Educational Data Mining Society, Bengaluru, India, 370–377. https://doi.org/10.5281/zenodo.8115653

[26] Tung Phung, Victor-Alexandru Pădurean, José Cambronero, Sumit Gulwani, Tobias Kohn, Rupak Majumdar, Adish Singla, and Gustavo Soares. 2023. Generative AI for Programming Education: Benchmarking ChatGPT, GPT-4, and Human Tutors. In *Proceedings of the 2023 ACM Conference on International Computing Education Research - Volume 2* (Chicago, IL, USA) *(ICER '23)*. Association for Computing Machinery, New York, NY, USA, 41–42. https://doi.org/10.1145/3568812.3603476

[27] Jaakko Rajala, Jenni Hukkanen, Maria Hartikainen, and Pia Niemelä. 2023. "Call me Kiran" – ChatGPT as a Tutoring Chatbot in a Computer Science Course. In *Proceedings of the 26th International Academic Mindtrek Conference*. 83–94. https://dl.acm.org/doi/pdf/10.1145/3616961.3616974

[28] Brian Reiser. 2004. Scaffolding Complex Learning: The Mechanisms of Structuring and Problematizing Student Work. *Journal Of The Learning Sciences* 13 (01 2004), 273–304. https://doi.org/10.1207/s15327809jls1303_2

[29] Kathryn M. Rich, T. Andrew Binkowski, Carla Strickland, and Diana Franklin. 2018. Decomposition: A K-8 Computational Thinking Learning Trajectory. In *Proceedings of the 2018 ACM Conference on International Computing Education Research* (Espoo, Finland) *(ICER '18)*. Association for Computing Machinery, New York, NY, USA, 124–132. https://doi.org/10.1145/3230977.3230979

[30] Robin Schmucker, Meng Xia, Amos Azaria, and Tom Mitchell. 2023. Ruffle&Riley: Towards the Automated Induction of Conversational Tutoring Systems. arXiv:2310.01420 [cs.CL]

[31] D. Sleeman and J.S. Brown. 1982. *Intelligent Tutoring Systems.* Academic Press. https://books.google.ch/books?id=pjqcAAAAMAAJ

[32] Ting Song and Kurt Becker. 2014. Expert vs. novice: Problem decomposition/recomposition in engineering design. In *2014 International Conference on interactive collaborative learning (ICL)*. IEEE, 181–190. https://ieeexplore.ieee.org/abstract/document/7017768

[33] Shashank Sonkar, Naiming Liu, Debshila Basu Mallick, and Richard G. Baraniuk. 2023. CLASS: A Design Framework for building Intelligent Tutoring Systems based on Learning Science principles. arXiv:2305.13272 [cs.CL]

[34] Ferhat Yarkin and Justin P. Coon. 2022. Simple Gray Coding and LLR Calculation for MDS Modulation Systems. *CoRR* abs/2201.08237 (2022). arXiv:2201.08237 https://arxiv.org/abs/2201.08237

[35] Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T. Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. 2023. MetaMath: Bootstrap Your Own Mathematical Questions for Large Language Models. arXiv:2309.12284 [cs.CL]

[36] Muru Zhang, Ofir Press, William Merrill, Alisa Liu, and Noah A. Smith. 2023. How Language Model Hallucinations Can Snowball. arXiv:2305.13534 [cs.CL]

[37] Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. 2020. BERTScore: Evaluating Text Generation with BERT. arXiv:1904.09675 [cs.CL]