Show or Tell? Interactive Task Learning with Large Language Models

Jacob Sansom[1], **Muhammad Khalifa**[1,2], **Honglak Lee**[1,2], **Joyce Chai**[1]
[1]University of Michigan [2]LG AI Research
{jhsansom, khalifam, honglak, chaijy}@umich.edu

Abstract

Large Language Models (LLMs) can perform tasks specified in natural language, making them accessible to users regardless of technical background. However, specifying tasks within a single, static prompt is often both difficult and suboptimal. Interactive Task Learning (ITL) [28, 18]—a goal for autonomous agents—proposes to address this challenge through multi-turn interactions: teachers provide a task description and (optionally) a demonstration, agents attempt the task while asking clarifying questions, and teachers offer feedback. Despite ITL's promise, systematic evaluation of LLMs' interactive learning capabilities remains limited. We introduce the **ListOps Domain**, a novel testbed for evaluating models' ability to learn compositional symbolic tasks through ITL. We evaluate small-to-medium size LLMs (4 to 32 billion parameters) and find that a limited form of teacher feedback—expressing only reminders about broken rules rather than explicitly identifying or correcting errors—enhances generalization. Using this feedback, we compare models' ITL and Few-Shot Learning (FSL) capabilities and find that ITL frequently outperforms FSL, especially within more powerful models. We conclude with a discussion of limitations and recommendations for advancing ITL research.

1 Introduction

The creation of AI agents has been a long sought-after goal hindered by both technical and human challenges. Large Language Models (LLMs) have accelerated the development of such agents [52], however adapting them to perform idiosyncratic tasks in custom environments is still difficult. Typical methods for training and customizing AI agents include finetuning, which is expensive and technically challenging, and prompt engineering, which is brittle and often suboptimal [46, 35, 8]. Laird et al. (2017) [28] propose an alternative approach, arguing that "training" should instead occur over the course of a multi-turn interaction in which a user *teaches* an AI agent in the same manner that they might teach a human learner. Within this multi-turn interaction, which they term Interactive Task Learning (ITL), a user begins with a demonstration of a desired behavior paired with an abstract description thereof. The agent then attempts the workflow, and the user provides feedback on its behavior and performance. Because ITL closely resembles how humans naturally teach one another tasks, it may offer a more intuitive path to customization, particularly for novice users.

LLMs excel at In-Context Learning (ICL) [5, 16, 1], which we define broadly as the ability to perform unseen tasks by conditioning generation on a prompt $\mathcal{S}$ that may contains instructions, demonstrations, or other forms of information. By framing a multi-turn ITL interaction as a prompt $\mathcal{S}_{\texttt{ITL}}$ used to steer generation, LLMs may also excel at ITL. Using this formulation, we explore the following two research questions: **Q1:** What properties must natural language feedback have for LLMs to effectively acquire and generalize task knowledge? **Q2:** How does ITL compare with alternative paradigms for teaching LLMs to perform tasks, such as Few-Shot Learning (FSL) [5]?

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: Multi-Turn Interactions in Large Language Models.

Our contributions are as follows:

1. In Section 4, we introduce the ListOps Domain, a synthetic list transformation environment that enables the systematic exploration of ITL techniques and capabilities.
2. In Section 5.1, we develop a taxonomy of teacher feedback, and we gather empirical data demonstrating that LLMs learn most effectively when broken rules are repeated back to them and when corrective information is withheld.
3. In Section 5.2, we utilize this form of teacher feedback to obtain ITL performance results across a variety of small LLMs. We directly compare ITL performance against that of Few-Shot Learning (FSL) and demonstrate that ITL presents a strong alternative.
4. We conclude by summarizing our limitations and directions for future research in Section 6.

2 Related Work

2.1 Interactive Task Learning

Laird et al. (2017) [28] first articulated ITL as a goal for AI agents and robots. Since then many works have focused on studying and implementing ITL within robots [6, 59, 34, 23, 18] and non-robotic AI agents [31]. Several recent works have incorporated LLMs into ITL, however LLMs have played a subsidiary role, either serving as a knowledge source for a symbolic agent [25, 26, 24], mapping human instructions into a structured language for use by a symbolic agent [29], using feedback to label policy rollouts [33], determining the generality of human feedback [15], or mapping human instructions into code for robotic control [32, 2]. In contrast, our work explores how well LLMs can perform ITL via In-Context Learning (ICL), without the help of a symbolic or otherwise external architecture. Furthermore, our work studies compositional, symbolic reasoning rather than robotics and embodied agents, which are more typical domains of study within the ITL literature.

2.2 In-Context and Few-Shot Learning

Among other reasons, LLMs have gained popularity over the last several years due to their remarkable in-context and few-shot learning abilities. We define In-Context Learning (ICL) broadly as the ability of an LLM to perform unseen tasks by conditioning generation on a prompt $\mathcal{S}$ that may contain instructions, demonstrations, or other forms of information. We define Few-Shot Learning (FSL) as a type of ICL that specifically utilizes examples of task execution within the prompt $\mathcal{S}$. Much work [5, 16, 1] has focused on the ICL and FSL abilities of language models. We build upon this work by studying ITL framed as a form of ICL.

In-Context Reinforcement Learning (ICRL) [37] is particularly relevant to our work. ICRL is a technique whereby an LLM gathers examples of task execution (on-policy), receives feedback in the form of scalar rewards, and combines both to construct a prompt $\mathcal{S}$ that steers itself towards proficiency on unseen tasks. ICRL is similar to ITL, except ITL enables rich natural language feedback, which provides strictly more information than scalar rewards.

2.3 Learning from Natural Language Feedback

Learning from natural language feedback is a vital component of ITL. Natural language enables a teacher to communicate not only *whether* a learner has made a mistake but also *when* the mistake was made as well as *why* the learner's behavior was mistaken. This precision shifts the burden of credit assignment from student to teacher. Natural language is thus theorized to underlie the uniquely human ability to acquire "kind-generalizable knowledge from a single manifestation" [14].

It is therefore unsurprising that learning from natural language feedback has been a popular subject for exploration within AI. Several previous works have studied the use of natural language feedback for **repairing faulty attempts**, wherein language is used to steer a model towards success on a particular task (rather than to generate general knowledge that can be repurposed on novel tasks). Researchers have developed several benchmarks for testing attempt repair using natural language feedback [9, 53, 57, 39] as well as several methods to more effectively enable models to repair their faulty attempts [48, 54, 58, 4, 22, 54].

In contrast, the use of natural language feedback to transmit **generalizable information**, where feedback on one task aids with the completion of future tasks, is less well-explored. [45, 7] introduce Imitation Learning from Feedback (ILF), which utilizes natural language feedback to repair faulty attempts (intra-task learning) and then finetunes over the resulting dataset of trajectories. ICAL [44] does the same, except it also annotates trajectories with abstract rules and knowledge and utilizes retrieval augmentation in addition to finetuning. Critically, both of these methods accumulate knowledge in parallel, treating each learning experience as independent. Doing so avoids several core issues within ITL, such as extracting task knowledge out of a longer stream of partially irrelevant information (e.g., small talk mixed with task feedback) and properly interpreting pedagogical references to prior experiences (e.g., "this task is similar to the one you tried yesterday"). [10] proposes Trace, a method analogous to backpropagation that enables an LLM to update its policy according to natural language feedback (which can include user and environmental feedback). Trace is neurosymbolic, utilizing the LLM to create and update code, which serves as the policy itself. In contrast, we are interested in the native abilities of LLMs themselves.

2.4 List Manipulation as a Domain

Our ListOps Domain, which is introduced in Section 4, draws inspiration from work spanning cognitive psychology and AI. Both fields have utilized list transformation functions of the form $f : \mathcal{A}^* \rightarrow \mathcal{A}^*$, (where $\mathcal{A}^*$ denotes the set of all finite-length sequences over alphabet $\mathcal{A}$) to study inductive reasoning. Within AI, researchers have devoted much effort towards inductive program synthesis (i.e., generating the logic for f based on examples of its execution) [19, 47, 3, 21, 49, 17, 38, 40, 13, 30]. Likewise, list transformation functions have been used to study concept acquisition within humans [42, 43]. To our knowledge, we are the first to use list transformation functions to study ITL. ITL requires both *deductive* and *inductive* reasoning since it proceeds from natural language instructions and feedback in addition to examples. We thus also view our work as an extension of comparisons between these two reasoning modalities [30, 11, 55].

3 Problem Formulation

Algorithm 1 Interactive Task Learning Procedure

$f_{\texttt{general}} = \pi_T(\mathcal{T})$ *// Solicit a general description of the task domain from the teacher*
$\mathcal{H} = [f_{\texttt{general}}]$ *// Initialize a history buffer*
for $i = 0$ to $n - 1$ **do** *// Learn over n unique tasks*
  $t^i \sim \mathcal{T}_{\texttt{train}}$ *// Sample task randomly*
  for $j = 0$ to $m - 1$ **do** *// Attempt this task m times, or until successful*
    $a^i_j \sim \pi_S(\cdot|\mathcal{H}, t^i)$ *// Agent attempts the task*
    $f^i_j \sim \pi_T(t^i, a^i_j)$ *// Get feedback from teacher after attempt*
    $\mathcal{H} \leftarrow \mathcal{H} \cup [t^i, a^i_j, f^i_j]$ *// Add task-attempt-feedback tuple to history buffer*
    if $\texttt{success}(t^i, a^i_j)$ **then break** *// Break if attempt succeeds on task*
  end for
  $A(\pi_S, \mathcal{H}) \leftarrow \frac{1}{|\mathcal{T}_{\texttt{test}}|} \sum_{t \sim \mathcal{T}_{\texttt{test}}} \texttt{success}\Big[t, \pi_S(\cdot|\mathcal{H}, t)\Big]$ *// Accuracy on test set*
end for
return $\pi_S(\cdot|\mathcal{H})$ *// Return final trained policy*

For this paper, we frame ITL as a structured multi-turn interaction between two agents: a student π_S and a teacher π_T. The student's objective is to use In-Context Learning (ICL) to execute tasks effectively. The teacher's objective is to teach the student using natural language in the form of a general description $f_{\texttt{general}} = \pi_T(\mathcal{T})$ of the task domain $\mathcal{T}$ (provided prior to any attempts) and feedback f^i_j (provided after each attempt a^i_j). The student is sequentially given n tasks to attempt and

m opportunities to attempt each task. After each attempt, the teacher provides feedback regarding the student's performance. This multi-turn interaction is stored into a history buffer $\mathcal{H}$ that is used as context to steer the student's generation. Algorithm 1 depicts this multi-turn interaction using pseudocode.

ITL encompasses a broad variety of pedagogical techniques amenable to human teachers. For the sake of simplicity, we defer aspects of ITL such as question-asking and curriculum learning to future work. For a comprehensive discussion of limitations, please refer to Section 6.

4 The ListOps Domain

Many real-world task domains require an agent to learn heterogeneous skills and compose them flexibly at test-time. An example would be a computer-use agent combining skills such as interacting with search bars, clicking on menu items, and handling checkout carts so it can order food. Unfortunately, rigorously evaluating ITL capabilities in real-world domains can be quite difficult. Synthesizing large numbers of user-customized tasks and skills (such as proprietary corporate procedures or data manipulations) is impractical in these domains. Additionally, many LLMs have already encountered generic skills (such as interacting with search bars) during pre-training, making it difficult to isolate ITL capabilities [36].

Thus, to systematically evaluate whether models can acquire heterogeneous skills in an interactive manner, we develop a synthetic environment we call the **List Operations (ListOps) Domain**. Our ListOps Domain tests models on their ability to acquire list-manipulation skills—a popular domain within AI and cognitive psychology (see Section 2.4)—and compose them in novel manners.

The ListOps Domain consists of procedurally-generated **worlds** that serve as testbeds for teaching and learning strategies. Each world $\mathcal{W}$ is designed to emulate a set of user-specific tasks and skills that a user might teach an agent via ITL.

We treat simple list manipulation functions such as `reverse_subsequence` and `swap_indices` as "skills" within the ListOps Domain. We utilize a set $\mathcal{S}$ of 16 list manipulation skills detailed in Appendix B. Each world $\mathcal{W}$ consists of a subset of these skills $\mathcal{S}_{\mathcal{W}} \subseteq \mathcal{S}$. Utilizing these skills, each world can generate infinitely many random tasks of the form $t = (f, x)$, where $f = s_1 \circ s_2 \circ \cdots \circ s_n$ is a transformation function that composes skills $s_1, ..., s_n \in \mathcal{S}_{\mathcal{W}}$, and $x \in \mathcal{A}^*$ is a list of arbitrary length that must be transformed according to the function's definition. The ListOps Domain constructs sequences x from an alphabet $\mathcal{A}$ consisting of all uppercase and lowercase letters. Please refer to Figure 1 for an example task and ground-truth answer.

Function Definition

```
def transform(sequence):
    sequence = blue_falcon(sequence, index=3)
    sequence = yellow_barrel(sequence, index1=0, index2=3)
    sequence = purple_coffee(sequence, index=1, amount=2)
    return sequence
```

Task

```
Please compute transform(["z", "g", "m", "M", "t"]), outputting
the result of each line.
```

Ground-Truth Answer

```
blue_falcon swaps the case of the letter at
`index`. So, line 1 changes "M" to "m":
#L1: ["z", "g", "m", "m", "t"]

yellow_barrel swaps the letters at `index1`
and `index2`. Line 2 thus swaps "z" and "m":
#L2: ["m", "g", "m", "z", "t"]

purple_coffee increments the letter at
`index` by `amount`. "g" → "h" → "i":
#L3: ["m", "i", "m", "z", "t"]
```

Figure 1: An example function, task, and ground-truth reasoning trace from the ListOps Domain. During testing, the LLM is provided the function definition and the task, which includes an input sequence to pass into the function. The LLM is tasked with producing a reasoning trace that transforms the sequence according to the definition of the function. The LLM is required to specify the output of each line using the format `#L2 ['a', 'b', ...]`. The prompt template is in Appendix D.

By generating many worlds at random, testing a particular teaching method within each of these worlds, and then aggregating performance, the ListOps Domain enables us to derive general insights regarding how good the teaching method is. To ensure that skills must be *learned* rather than *inferred* from their names, we obfuscate their names. The obfuscation process renames each skill to a semantically meaningless, world-specific phrase such as `blue_falcon` or `yellow_barrel`.

5 Experiments and Results

5.1 What properties must natural language feedback have for LLMs to effectively acquire and generalize task knowledge?

We taxonomize feedback according to the following hierarchy. Categories 1-3 form a strict hierarchy, with each level providing more information than the previous. Category 3 (Explanation) is then subdivided into three mutually exclusive types that represent different approaches to explaining errors.

1. **Identification:** Simply identifies *whether* or not an error was made (i.e., provides binary feedback). Learning from this sort of signal is roughly equivalent to In-Context Reinforcement Learning (ICRL) [37]. For example: "You made an error."
2. **Localization:** Locates *when* the error was made. Within the ListOps Domain, this corresponds to the line within the transformation function on which the student erred. For example: "You made an error when trying to execute line 2."
3. **Explanation:** Explains *why* the student's actions constitute an error or *what* the student should have done. We subdivide explanatory feedback into three mutually exclusive types:
   (a) **Error Description:** Describes what the student did wrong without explaining why it was wrong or how to fix it. For example: "You removed 'a' from the subsequence when filtering."
   (b) **Rule Explication:** States the general principle or rule that the student violated. For example: "When filtering uppercase letters, leave all lowercase letters in the subsequence."
   (c) **Corrective Information:** Provides the specific correct output for the erroneous step. For example: "The correctly filtered subsequence at line 2 is ['a', 'v', 'i']."

5.1.1 Accuracy of Repair

We test each possible combination c of the above feedback categories across an identical set of 100 ListOps worlds. Within each world, we randomly generate a task t^0 and give the student up to $m = 3$ attempts $[a_0^0, a_1^0, a_2^0]$ to solve it. The student receives feedback according to a particular feedback combination (e.g., Identification + Localization + Error Description) after each attempt. We use the results of this process to compute the **accuracy of repair** A_R^c for a particular combination c, which refers to the proportion of skills s_i that the student executed incorrectly during the first attempt a_0^0 but correctly during a subsequent attempt a_1^0 or a_2^0.

We utilize Gemini 2.5 Flash [12] paired with a parser to generate plaintext feedback within each category. More details regarding the prompts and the parser are available in Appendix D.2. We focus our analysis on relatively small models (4 to 32 billion parameters), which struggle on the ListOps Domain, since we are more interested in the learning dynamics underlying ITL rather than achieving state-of-the-art performance on a real-world task. Small models enable cost-effective experimentation at scale in our controlled, synthetic environment. We thus perform our analysis over four models from three model families: Gemma3-27B [50], Qwen2.5-14B [41], Qwen3-4B, and Qwen3-8B [56]. For consistency, we use simplified naming conventions throughout this paper (e.g., "Gemma3-27B" for the instruction finetuned version of Gemma 3 27B and "Qwen2.5-14B" for Qwen2.5-14B-Instruct; please refer to Table 2 for all of the full model names).

Examining the left side of Figure 2, Localization appears to confer few benefits over Identification. By contrast, adding just one sub-category of Explanation feedback—Rule Explication—confers consistently large benefits. We choose Rule Explication as the representative sub-category of Explanation feedback since it provides the best generalization results (as detailed in Figure 3). The benefits of Rule Explication suggest that **LLMs can take advantage of rich linguistic feedback to more rapidly and consistently address errors.**

Examining the right side of Figure 2 reveals a positive effect of all forms of Explanation feedback on accuracy of repair. In other words, all three categories have a positive impact on the student LLM's ability to repair its faulty generations.

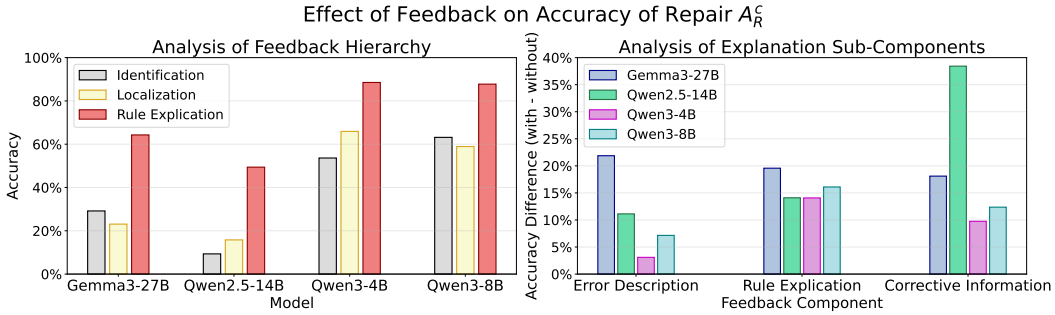

Figure 2: An analysis of the effect of various feedback combinations on accuracy of repair A_R^c. **(Left)** A comparison of feedback categories 1-3, which are strictly hierarchical. **(Right)** A display of the average difference *with* versus *without* each Explanation sub-category, computed across all combinations of Explanation feedback per the method in Appendix C.1. For the raw data, please refer to Table 3.

5.1.2 Accuracy of Generalization

We also test the generalization of each feedback combination c by generating a new task t^1 within the same ListOps world. We ensure that task t^1 includes the skill s_i that the student initially executed incorrectly during task t^0. We then let the student attempt task t^1 once, using the conversational history as context, and determine whether or not it executed skill s_i correctly. We use this procedure to compute the **accuracy of generalization** A_G^c, which refers to the proportion of these skills s_i that were executed correctly during t^1 across all ListOps worlds.

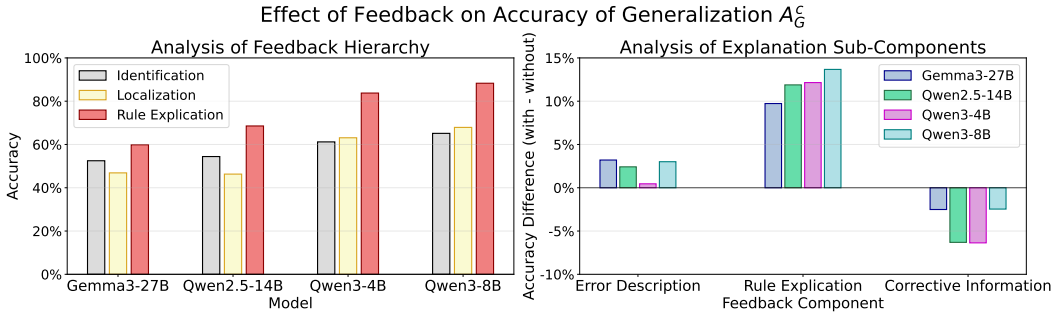

Figure 3: An analysis of the effect of various feedback combinations on accuracy of generalization A_G^c. **(Left)** A comparison of feedback categories 1-3, which are strictly hierarchical. **(Right)** A display of the average difference *with* versus *without* each Explanation sub-category, computed across all combinations of Explanation feedback per the method in Appendix C.1. For the raw data, please refer to Table 4.

Examining the left side of Figure 3 reveals a story similar to that of accuracy of repair: Localization confers few benefits over Identification, however adding the best type of Explanation feedback greatly improves generalization.

Examining the right side of Figure 3, the effect of each Explanation sub-category appears to be mixed. Rule Explication yields a positive effect across all four models. Given that Rule Explication utilizes language to articulate the rule in a general, task-agnostic manner, it makes intuitive sense that an agent of sufficient intelligence would be able to utilize this category of feedback to obtain general understanding; after all, the "transmission of generic knowledge" through language is one method by which humans rapidly come to grasp general concepts [14]. Our experiment evidences the notion that **even small-to-medium size LLMs might possess intelligence sufficient to acquire general knowledge from Rule Explication feedback**.

By contrast, **Corrective Information yields a negative effect on accuracy of generalization across all four models.** Although Corrective Information is effective at enabling the student to repair its faulty attempts, it fails to induce general knowledge. We hypothesize that, by simply providing the student with the answer, Corrective Information may enable the student to succeed at a task without referencing or modifying its (potentially incorrect) representations of the general task structure.

Given the disparate effect of various feedback categories on accuracy of repair versus generalization, we contend that **accuracy of repair is not a good proxy for accuracy of generalization.** Thus, progress made towards repairing faulty attempts using natural language feedback (as surveyed in Section 2.3) may not constitute progress towards pedagogical and learning techniques that can produce general knowledge from natural language feedback.

Despite the positive-on-average performance of Error Descriptions, **the best overall combination of Explanation feedback sub-categories is Rule Explication alone** (paired with error Identification and Localization). Please refer to Table 4 for a comprehensive comparison of all feedback categories. We thus use this feedback combination for the remaining ITL experiments within this paper.

5.2 How does ITL compare with alternative paradigms for teaching LLMs to perform tasks, such as Few-Shot Learning (FSL)?

We compare ITL to two alternative pedagogical methods:

1. **Baseline Prompt:** This is a templated prompt that provides information about each skill's definition as well as an example of its execution. We intend for this to serve as the sort of general description $f_{\texttt{general}}$ that a teacher might give a student at the beginning of an ITL interaction. An example is located in Appendix D.4.
2. **Few-Shot Learning (FSL):** This prompt format provides ground-truth reasoning traces for several randomly-sampled tasks from the relevant ListOps world. An example is located in Appendix D.3.

To ensure a fair comparison between FSL and ITL, we include the Baseline Prompt at the beginning of both, and we utilize an identical set of $n = 5$ tasks per world for each method. Within FSL, we provide the ground-truth reasoning traces for each of these five tasks. We generate these ground-truths by repeatedly prompting Gemini 2.5 Flash [12] until it produces a correct reasoning trace (more details in Appendix C.2). Within ITL, we allow the student to attempt each of these five tasks up to $m = 3$ times—receiving feedback from Gemini 2.5 Flash each time—before forcing the student to learn on a new task. We report ITL results using the best form of feedback identified within Table 4, which includes Identification, Localization, and Rule Explication.

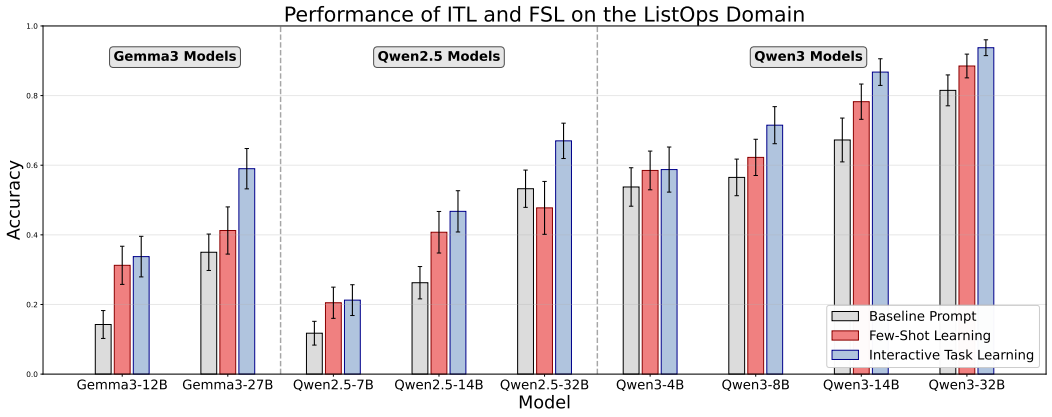

Figure 4: A comparison between a Baseline Prompt, Interactive Task Learning (ITL), and Few-Shot Learning (FSL) on the ListOps domain. We test models from the Gemma3 [50], Qwen2.5 [41], and Qwen3 [56] families. Each bar represents the mean accuracy across 4 tests each for 100 ListOps worlds (seeds), yielding 400 datapoints total. We display 95% confidence intervals around each bar.

ITL demonstrates performance competitive with FSL, achieving statistically significant improvements over FSL in six out of nine models ($p < 0.05$) while showing higher mean accuracy in all models tested. These results suggest that **ITL provides, at minimum, comparable performance to FSL and may offer advantages for certain model architectures.**

Furthermore, Figure 5 displays the generalization performance of Qwen2.5-14B and Qwen3-14B as a function of the number of tasks seen during learning. Although FSL begins to catch up to ITL as the number of tasks increases, **ITL exhibits an even stronger advantage over FSL when the number of learning tasks is quite low (e.g., one or two).**

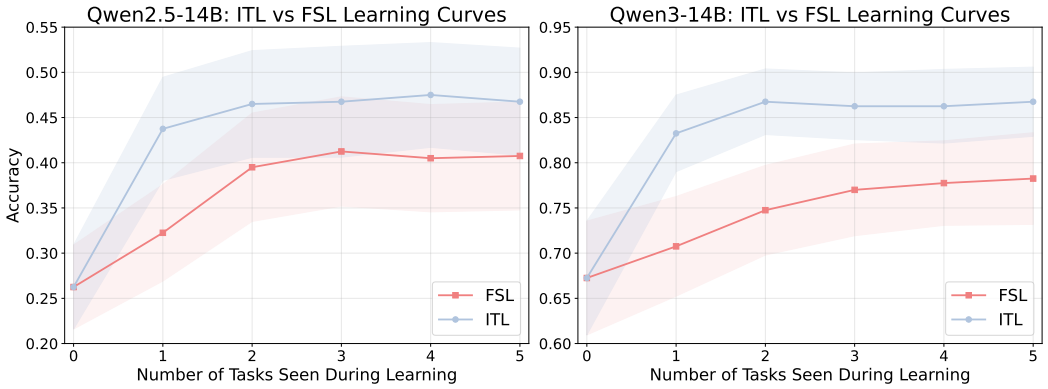

Figure 5: A display of ITL and FSL by Qwen2.5-14B [41] and Qwen3-14B [56]. We plot test accuracy as a function of the number of learning tasks presented during In-Context Learning.

6 Limitations and Future Work

By developing a systematic evaluation of LLMs in an interactive feedback loop, we have made significant progress towards studying the principles most central to Interactive Task Learning. However ITL is a broad concept that encompasses a variety of learning skills beyond what we study in this paper. In this section, we outline several ways in which our ListOps Domain falls short of truly robust ITL, and we propose methods for overcoming these deficits within future work.

6.1 Unstructured Student-Teacher Interactions

The current learning paradigm we study involves a rigid interaction structure between student and teacher. A task is randomly selected by the domain itself and fed to the student, the student is able to attempt this task a fixed number of times, and the teacher is allowed to give feedback only at the conclusion of each attempt. In a real-world setting, such a rigid structure could frustrate a user who is accustomed to more dynamic interactions with other humans. Additionally, our rigid structure does not take full advantage of the pedagogical expertise that a human teacher might possess. For instance, a human teacher might provide helpful advice *prior* to an attempt; our formulation only provides foresight feedback in the form of a templated $f_{\texttt{general}}$ prompt. A human teacher might utilize their theory of mind to identify weaknesses within the student and ask questions or introduce tasks that bring these weaknesses to the surface; our randomized task selection mechanism does not allow for this sort of dynamic interaction. Finally, a robust domain for testing ITL ought to allow the student to ask questions as well. LLMs are known to struggle with meta-cognition [51, 20], an ability that enables humans to identify and resolve uncertainties, so research into question-asking will surely enhance ITL capabilities. We believe the ListOps Domain could be adapted in the future to accommodate many of these features.

6.2 Memory and Adaptibility

Laird et al. (2017) [28] write the following about humans: "After we learn the essence of a task, we can learn extensions and modifications to it, getting better and better with experience." The

ListOps Domain provides an initial testbed for studying this phenomenon. For instance, we introduce skills at random, so not all of them are encountered within the first task instance. Despite this, the ListOps Domain utilizes a static task distribution throughout the lifetime of a ListOps world $\mathcal{W}$. To more effectively study ITL, the ListOps Domain could be extended such that some skills are only introduced after more time (and, as a result, context) has elapsed. Such a change might necessitate more robust memory architectures than in-context learning. Furthermore, additional changes to the internal structure of skills or to the structure of the task itself, which would both cause distribution shift, could be valuable research endeavors.

6.3 Multimodality

By simulating ITL in a plaintext conversation, the ListOps Domain omits many important pedagogical signals that a human user might provide in a real-world scenario. For instance, human teachers oftentimes direct their students' attention towards relevant phenomena by pointing or directing their own gaze [28]. Even for non-embodied AI agents, these gestural and nonverbal forms of communication transmit important information that is lost when represented in text alone. We recommend research within Human-Computer Interaction (HCI) to investigate what sorts of computing form factors and AI models might afford these interactions.

7 Conclusion

In this paper, we have presented an initial analysis of LLMs' ability to perform Interactive Task Learning via In-Context Learning. Overall, LLMs show promise at our formulation of ITL, outperforming FSL in many scenarios, however further research ought to be devoted towards studying ITL abilities in real-world settings. We hope that this work will serve as a foundation for this research.

References

[1] Ekin Akyürek, Dale Schuurmans, Jacob Andreas, Tengyu Ma, and Denny Zhou. What learning algorithm is in-context learning? investigations with linear models. *arXiv preprint arXiv:2211.15661*, 2022.

[2] Leonard Bärmann, Rainer Kartmann, Fabian Peller-Konrad, Jan Niehues, Alex Waibel, and Tamim Asfour. Incremental learning of humanoid robot behavior from natural interaction and large language models. *Frontiers in Robotics and AI*, 11:1455375, 2024.

[3] Alan W Biermann. The inference of regular lisp programs from examples. *IEEE transactions on Systems, Man, and Cybernetics*, 8(8):585–600, 2007.

[4] Xiaohe Bo, Zeyu Zhang, Quanyu Dai, Xueyang Feng, Lei Wang, Rui Li, Xu Chen, and Ji-Rong Wen. Reflective multi-agent collaboration based on large language models. *Advances in Neural Information Processing Systems*, 37:138595–138631, 2024.

[5] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[6] Joyce Chai, Qiaozi Gao, Lanbo She, Shaohua Yang, Sari Saba-Sadiya, and Guangyue Xu. Language to action: Towards interactive task learning with physical agents. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, 2018.

[7] Angelica Chen, Jérémy Scheurer, Jon Ander Campos, Tomasz Korbak, Jun Shern Chan, Samuel R Bowman, Kyunghyun Cho, and Ethan Perez. Learning from natural language feedback. *Transactions on machine learning research*, 2024.

[8] Yanda Chen, Chen Zhao, Zhou Yu, Kathleen McKeown, and He He. On the relation between sensitivity and accuracy in in-context learning. *arXiv preprint arXiv:2209.07661*, 2022.

[9] Ching-An Cheng, Andrey Kolobov, Dipendra Misra, Allen Nie, and Adith Swaminathan. Llf-bench: Benchmark for interactive learning from language feedback. *arXiv preprint arXiv:2312.06853*, 2023.

[10] Ching-An Cheng, Allen Nie, and Adith Swaminathan. Trace is the next autodiff: Generative optimization with rich feedback, execution traces, and llms. *Advances in Neural Information Processing Systems*, 37:71596–71642, 2024.

[11] Kewei Cheng, Jingfeng Yang, Haoming Jiang, Zhengyang Wang, Binxuan Huang, Ruirui Li, Shiyang Li, Zheng Li, Yifan Gao, Xian Li, et al. Inductive or deductive? rethinking the fundamental reasoning abilities of llms. *arXiv preprint arXiv:2408.00114*, 2024.

[12] Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*, 2025.

[13] Andrew Cropper, Rolf Morel, and Stephen Muggleton. Learning higher-order logic programs. *Machine Learning*, 109(7):1289–1322, 2020.

[14] Gergely Csibra and György Gergely. Natural pedagogy. *Trends in cognitive sciences*, 13(4):148–153, 2009.

[15] Yuchen Cui, Siddharth Karamcheti, Raj Palleti, Nidhya Shivakumar, Percy Liang, and Dorsa Sadigh. No, to the right: Online language corrections for robotic manipulation via shared autonomy. In *Proceedings of the 2023 ACM/IEEE International Conference on Human-Robot Interaction*, pages 93–101, 2023.

[16] Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Jingyuan Ma, Rui Li, Heming Xia, Jingjing Xu, Zhiyong Wu, Tianyu Liu, et al. A survey on in-context learning. *arXiv preprint arXiv:2301.00234*, 2022.

[17] John K Feser, Swarat Chaudhuri, and Isil Dillig. Synthesizing data structure transformations from input-output examples. *ACM SIGPLAN Notices*, 50(6):229–239, 2015.

[18] KA Gluck and JE Laird. Interactive task learning: Agents, robots, and humans acquiring new tasks through natural interactions, 2018.

[19] Cordell Green. Application of theorem proving to problem solving. In *Readings in artificial intelligence*, pages 202–222. Elsevier, 1981.

[20] Maxime Griot, Coralie Hemptinne, Jean Vanderdonckt, and Demet Yuksel. Large language models lack essential metacognition for reliable medical reasoning. *Nature communications*, 16(1):642, 2025.

[21] Stanford Artificial Intelligence Laboratory. Automatic-Programming Research Group and C Cordell Green. *Progress report on program-understanding systems*. Stanford Univ., Computer Science Department, School of Humanities, 1974.

[22] Xiang Huang, Sitao Cheng, Shanshan Huang, Jiayu Shen, Yong Xu, Chaoyun Zhang, and Yuzhong Qu. Queryagent: A reliable and efficient reasoning framework with environmental feedback-based self-correction. *arXiv preprint arXiv:2403.11886*, 2024.

[23] James R Kirk and John E Laird. Learning hierarchical symbolic representations to support interactive task learning and knowledge transfer. In *IJCAI*, pages 6095–6102, 2019.

[24] James R Kirk, Robert E Wray, and John E Laird. Exploiting language models as a source of knowledge for cognitive agents. In *Proceedings of the AAAI Symposium Series*, volume 2, pages 286–294, 2023.

[25] James R Kirk, Robert E Wray, Peter Lindes, and John E Laird. Integrating diverse knowledge sources for online one-shot learning of novel tasks. *arXiv preprint arXiv:2208.09554*, 2022.

[26] James R Kirk, Robert E Wray, Peter Lindes, and John E Laird. Improving knowledge extraction from llms for task learning through agent analysis. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 18390–18398, 2024.

[27] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

[28] John E Laird, Kevin Gluck, John Anderson, Kenneth D Forbus, Odest Chadwicke Jenkins, Christian Lebiere, Dario Salvucci, Matthias Scheutz, Andrea Thomaz, and Greg Trafton. Interactive task learning. *IEEE Intelligent Systems*, 32(4):6–21, 2017.

[29] Lane Lawley and Christopher Maclellan. Val: Interactive task learning with gpt dialog parsing. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*, pages 1–18, 2024.

[30] Jiachun Li, Pengfei Cao, Zhuoran Jin, Yubo Chen, Kang Liu, and Jun Zhao. Mirage: Evaluating and explaining inductive reasoning process in language models. *arXiv preprint arXiv:2410.09542*, 2024.

[31] Toby Jia-Jun Li, Tom Mitchell, and Brad A Myers. Interactive task learning from gui-grounded natural language instructions and demonstrations. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 215–223, 2020.

[32] Jacky Liang, Fei Xia, Wenhao Yu, Andy Zeng, Montserrat Gonzalez Arenas, Maria Attarian, Maria Bauza, Matthew Bennice, Alex Bewley, Adil Dostmohamed, et al. Learning to learn faster from human feedback with language model predictive control. *arXiv preprint arXiv:2402.11450*, 2024.

[33] Huihan Liu, Alice Chen, Yuke Zhu, Adith Swaminathan, Andrey Kolobov, and Ching-An Cheng. Interactive robot learning from verbal correction. *arXiv preprint arXiv:2310.17555*, 2023.

[34] Huihan Liu, Shivin Dass, Roberto Martín-Martín, and Yuke Zhu. Model-based runtime monitoring with interactive imitation learning. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4154–4161. IEEE, 2024.

[35] Yao Lu, Max Bartolo, Alastair Moore, Sebastian Riedel, and Pontus Stenetorp. Fantastically ordered prompts and where to find them: Overcoming few-shot prompt order sensitivity. *arXiv preprint arXiv:2104.08786*, 2021.

[36] Inbal Magar and Roy Schwartz. Data contamination: From memorization to exploitation. *arXiv preprint arXiv:2203.08242*, 2022.

[37] Giovanni Monea, Antoine Bosselut, Kianté Brantley, and Yoav Artzi. Llms are in-context bandit reinforcement learners. *arXiv preprint arXiv:2410.05362*, 2024.

[38] Peter-Michael Osera and Steve Zdancewic. Type-and-example-directed program synthesis. *ACM SIGPLAN Notices*, 50(6):619–630, 2015.

[39] Jane Pan, Ryan Shar, Jacob Pfau, Ameet Talwalkar, He He, and Valerie Chen. When benchmarks talk: Re-evaluating code llms with interactive feedback. *arXiv preprint arXiv:2502.18413*, 2025.

[40] Nadia Polikarpova, Ivan Kuraj, and Armando Solar-Lezama. Program synthesis from polymorphic refinement types. *ACM SIGPLAN Notices*, 51(6):522–538, 2016.

[41] Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report, 2025.

[42] Joshua Rule, Eric Schulz, Steven T Piantadosi, and Joshua B Tenenbaum. Learning list concepts through program induction. *BioRxiv*, page 321505, 2018.

[43] Joshua Stewart Rule. *The child as hacker: building more human-like models of learning*. PhD thesis, Massachusetts Institute of Technology, 2020.

[44] Gabriel Sarch, Lawrence Jang, Michael Tarr, William W Cohen, Kenneth Marino, and Katerina Fragkiadaki. Vlm agents generate their own memories: Distilling experience into embodied programs of thought. *Advances in Neural Information Processing Systems*, 37:75942–75985, 2024.

[45] Jérémy Scheurer, Jon Ander Campos, Tomasz Korbak, Jun Shern Chan, Angelica Chen, Kyunghyun Cho, and Ethan Perez. Training language models with language feedback at scale. *arXiv preprint arXiv:2303.16755*, 2023.

[46] Melanie Sclar, Yejin Choi, Yulia Tsvetkov, and Alane Suhr. Quantifying language models' sensitivity to spurious features in prompt design or: How i learned to start worrying about prompt formatting. *arXiv preprint arXiv:2310.11324*, 2023.

[47] David E Shaw, William R Swartout, and C Cordell Green. Inferring lisp programs from examples. In *IJCAI*, volume 75, pages 260–267, 1975.

[48] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36:8634–8652, 2023.

[49] Douglas R Smith. The synthesis of lisp programs from examples: A survey. *Automatic program construction techniques*, 307:324, 1984.

[50] Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, et al. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*, 2025.

[51] Guoqing Wang, Wen Wu, Guangze Ye, Zhenxiao Cheng, Xi Chen, and Hong Zheng. Decoupling metacognition from cognition: A framework for quantifying metacognitive ability in llms. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 25353–25361, 2025.

[52] Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):186345, 2024.

[53] Xingyao Wang, Zihan Wang, Jiateng Liu, Yangyi Chen, Lifan Yuan, Hao Peng, and Heng Ji. Mint: Evaluating llms in multi-turn interaction with tools and language feedback. *arXiv preprint arXiv:2309.10691*, 2023.

[54] Jiajun Xi, Yinong He, Jianing Yang, Yinpei Dai, and Joyce Chai. Teaching embodied reinforcement learning agents: Informativeness and diversity of language use. *arXiv preprint arXiv:2410.24218*, 2024.

[55] Fangzhi Xu, Qika Lin, Jiawei Han, Tianzhe Zhao, Jun Liu, and Erik Cambria. Are large language models really good logical reasoners? a comprehensive evaluation and beyond. *IEEE Transactions on Knowledge and Data Engineering*, 2025.

[56] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

[57] Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*, 2024.

[58] Weiran Yao, Shelby Heinecke, Juan Carlos Niebles, Zhiwei Liu, Yihao Feng, Le Xue, Rithesh Murthy, Zeyuan Chen, Jianguo Zhang, Devansh Arpit, et al. Retroformer: Retrospective large language agents with policy gradient optimization. *arXiv preprint arXiv:2308.02151*, 2023.

[59] Lihan Zha, Yuchen Cui, Li-Heng Lin, Minae Kwon, Montserrat Gonzalez Arenas, Andy Zeng, Fei Xia, and Dorsa Sadigh. Distilling and retrieving generalizable knowledge for robot manipulation via language corrections. In *2024 IEEE international conference on robotics and automation (ICRA)*, pages 15172–15179. IEEE, 2024.

A Acknowledgments

This material is based upon work supported by the National Science Foundation Graduate Research Fellowship Program under Grant No. DGE 2241144. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

We thank Jeffrey Brill, Creighton Glasscock, and Anuj Tambwekar for helpful discussions and insights during the initial phases of this work.

B ListOps Domain

As introduced in Section 4, the ListOps Domain consists of compositional transformation functions of the form $f = s_1 \circ s_2 \circ \cdots \circ s_n$. These transformation functions compose skills $s_1, ..., s_n \in \mathcal{S}_\mathcal{W}$, where $\mathcal{S}_\mathcal{W} \subseteq \mathcal{S}$ is a subset of skills drawn from a universal skillset $\mathcal{S}$. Each "skill" is a simple transformation function that intakes a sequence, manipulates it in some way (e.g., swapping the letters at two indices), and then returns the transformed sequence. The entire universal skillset $\mathcal{S}$ is detailed in Table 1.

Table 1: Skills within the ListOps Domain

Skill Name	**Description**
`swap_indices`	Swaps two letters at specified indices
`increment_letter`	Increments a letter by a specified amount with wraparound
`set_index_to_value`	Sets the letter at a given index to a specified value
`change_case_at_index`	Changes the case of the letter at a specified index
`sort_subsequence`	Sorts a subsequence alphabetically (lowercase before uppercase)
`reverse_subsequence`	Reverses the order of letters in a subsequence
`shift_letter`	Moves a letter to a different position in the sequence
`delete_index`	Removes the letter at a specified index
`delete_letter`	Removes all instances of a specified letter
`insert_letter`	Inserts a letter at a specified index
`set_case`	Sets a letter at a given index to uppercase or lowercase
`remove_duplicates`	Removes duplicate letters from a subsequence
`duplicate_subsequence`	Creates a copy of a subsequence at the same location
`rotate_subsequence`	Performs circular shift of letters within a subsequence
`filter_by_case`	Keeps only uppercase or lowercase letters in a subsequence
`filter_by_alphabet_range`	Keeps only letters within a specified alphabetical range

C Experimental Details and Further Analysis

We deployed all open-weight models (i.e., every model except for Gemini 2.5 Flash) using vLLM [27]. We used greedy decoding (temperature of $t = 0$) across all open-weight models to ensure deterministic, reproducible results and enable fair comparison across different model families. While this may not reflect each model's optimal performance, it provides a consistent evaluation framework for comparing pedagogical approaches.

We use a simplified naming scheme to refer to models throughout this paper. Please refer to Table 2 for all of the "full" model names (i.e., the locations of the models on HuggingFace).

Table 2: Simplified Model Names

Simplified Model Name	HuggingFace Model Name
Gemma3-12B	`google/gemma-3-12b-it`
Gemma3-27B	`google/gemma-3-27b-it`
Qwen2.5-7B	`Qwen/Qwen2.5-7B-Instruct`
Qwen2.5-14B	`Qwen/Qwen2.5-14B-Instruct`
Qwen2.5-32B	`Qwen/Qwen2.5-32B-Instruct`
Qwen3-4B	`Qwen/Qwen3-4B`
Qwen3-8B	`Qwen/Qwen3-8B`
Qwen3-14B	`Qwen/Qwen3-14B`
Qwen3-32B	`Qwen/Qwen3-32B`

For all models, we utilize a decoding limit of 4096 tokens (not including prior context). We forcibly truncate model output after this limit has been reached and return the output.

For all reasoning models—which coincides in this paper with the Qwen3 model family—we modified the chat template so that `<think>` blocks from previous conversational turns were preserved, which is not done by default. Furthermore, for these models, we truncate thinking 256 tokens prior to the 4096 token limit by injecting a `</think>` token, and then we provide the model 256 additional tokens to provide its final answer.

For all experiments, we utilized sequences of length $|x| = 8$, transformation functions composed of 4 skills each, and $|\mathcal{S}_{\mathcal{W}}| = 8$ skills per ITL world.

C.1 More Details on Feedback Comparison Experiments

This section contains more details regarding the experiments performed in Section 5.1. Tables 3 and 4 display the raw accuracy of repair A_c^R and accuracy of generalization A_c^G results for different feedback combinations c.

Table 3: Accuracy of repair A_c^R by feedback combination and model. Feedback categories tested include **(I)**dentification, **(L)**ocalization, **(D)**escription of Error, **(R)**ule Explication, and **(C)**orrective Information, per the hierarchy detailed in Section 5.1.

Feedback Components					Model Performance			
I	**L**	**D**	**R**	**C**	**Gemma3-27B**	**Qwen2.5-14B**	**Qwen3-4B**	**Qwen3-8B**
✓					0.292	0.093	0.536	0.632
✓	✓				0.231	0.158	0.659	0.589
✓	✓	✓			0.654	0.472	0.702	0.750
✓	✓		✓		0.643	0.494	0.885	0.878
✓	✓			✓	0.554	0.798	0.885	0.817
✓	✓	✓	✓		0.785	0.600	**0.946**	0.942
✓	✓	✓		✓	**0.844**	0.783	0.860	0.929
✓	✓		✓	✓	0.810	0.820	0.897	**0.980**
✓	✓	✓	✓	✓	0.829	**0.860**	0.941	0.929

Table 4: Accuracy of generalization A_c^G by feedback combination and model. Feedback categories tested include **(I)**dentification, **(L)**ocalization, **(D)**escription of Error, **(R)**ule Explication, and **(C)**orrective Information, per the hierarchy detailed in Section 5.1.

Feedback Components					**Accuracy of Generalization A_c^G**			
I	**L**	**D**	**R**	**C**	**Gemma3-27B**	**Qwen2.5-14B**	**Qwen3-4B**	**Qwen3-8B**
✓					0.525	0.544	0.612	0.651
✓	✓				0.469	0.463	0.631	0.679
✓	✓	✓			0.544	0.600	0.652	0.761
✓	✓		✓		0.599	**0.686**	**0.838**	**0.883**
✓	✓			✓	0.460	0.443	0.597	0.706
✓	✓	✓	✓		**0.606**	0.627	0.813	0.859
✓	✓	✓		✓	0.500	0.506	0.684	0.713
✓	✓		✓	✓	0.576	0.609	0.732	0.804
✓	✓	✓	✓	✓	0.581	0.565	0.667	0.860

To create the righthand sides of Figures 2 and 3, we average the performance of all combinations *without* that category and subtract it from the average performance of all combinations *with* that same category. For instance, to test the benefits of Error Descriptions (`D`), we perform the following accuracy calculation. Since this is simply a comparison of the Explanation feedback subcategories, we implicitly assume that Identification and Localization are always present, even in the `None` category below.

$$\left[(A_{\texttt{D+R+C}} - A_{\texttt{R+C}}) + (A_{\texttt{D+R}} - A_{\texttt{R}}) + (A_{\texttt{D+C}} - A_{\texttt{C}}) + (A_{\texttt{D}} - A_{\texttt{None}})\right]/4$$

This calculation, in effect, tells us the average positive impact incurred by adding error identification.

C.2 Method for Generating Ground-Truths with Gemini

We generate a dataset of 2000 ground-truth reasoning traces (20 tasks each across 100 ListOps worlds) on the ListOps domain by prompting Gemini 2.5 Flash [12] without any obfuscation (such that the meaning of each subfunction can be inferred from its name). We generate each reasoning trace independently (i.e., utilizing no history) by providing Gemini with the Baseline Prompt detailed in Appendix D. We utilize the default temperature of $t = 1.0$ provided by Google's API. We let Gemini have three opportunities to generate the ground-truth (utilizing randomness forced by the non-zero temperature to encourage Gemini to output novel attempts). If Gemini fails after three consecutive attempts, we provide a templated version of the ground-truth (complete with reasoning explanations) as a single in-context example. We then repetitively query Gemini using the baseline prompt as well as the templated ground-truth until Gemini yields a correct output.

C.3 Detailed Results and Further Analysis for ListOps Domain Experiments

We tested each learning method (e.g., ITL with a particular teacher configuration) across 100 distinct ListOps worlds (i.e., 100 seeds). Within each world, we created a learning set of five tasks and a test set of four tasks, across which we computed an accuracy within that ListOps world. We then report results for the mean accuracy across all worlds as well as 95% confidence intervals. In Table 5, we report mean performance as well as statistical significance. We compute statistical significance using paired t-tests and mark results that are statistically significant improvements with asterisks.

Both FSL and ITL can be viewed as methods for formulating strings $\mathcal{S}_{\texttt{FSL}}$ and $\mathcal{S}_{\texttt{ITL}}$ that steer the LLM's generation. The contents of each string are as follows, where t^i refers to task i, g^i refers to the ground-truth for t^i, and a^i_j and f^i_j refer to the j'th attempt and corresponding feedback for t^i.

$$\mathcal{S}_{\texttt{FSL}} = [f_{\texttt{general}}, t^1, g^1, t^2, g^2, ..., t^n, g^n]$$

$$\mathcal{S}_{\texttt{ITL}} = [f_{\texttt{general}}, t^1, a^1_1, f^1_1, a^1_2, f^1_2, ..., a^n_{\leq m}, f^n_{\leq m}]$$

Table 5: Performance comparison of the Baseline Prompt, Few-Shot Learning (FSL), and Interactive Task Learning (ITL) on the ListOps Domain. **Bold** text indicates the highest-performing learning strategy for each model. Asterisks * indicate when a learning strategy outperforms the strategy directly to its left (i.e., FSL outperforming Baseline Prompt or ITL outperforming FSL) to a statistically significant degree.

Model	Baseline Prompt	Few-Shot Learning	Interactive Task Learning
Gemma3-12B	0.142	0.312*	**0.338**
Gemma3-27B	0.350	0.412	**0.590***
Qwen2.5-7B	0.117	0.205*	**0.212**
Qwen2.5-14B	0.263	0.407*	**0.468***
Qwen2.5-32B	0.532	0.477	**0.670***
Qwen3-4B	0.537	0.585*	**0.588**
Qwen3-8B	0.565	0.623*	**0.715***
Qwen3-14B	0.672	0.782*	**0.868***
Qwen3-32B	0.815	0.885*	**0.938***

It is also possible to view both of these as methods for extracting pedgagogical information from a strong teacher model—Gemini 2.5 Flash in this case. In FSL, the teacher π_T is queried independently using each task to obtain a ground truth: $g^i \sim \pi_T(t^i)$. In ITL, the teacher is queried using the task and the student's attempt to obtain feedback: $f^i_j \sim \pi_T(t^i, a^i_j)$.

C.4 Analysis of Attempts During Interactive Task Learning

Figure 6 displays the number of incorrect and correct attempts that each model made when approaching the five learning tasks. The number of failed attempts per learning trajectory was quite varied. Generally, the number of failed attempts has an inverse relationship with model performance at ITL, while the number of successful attempts has a direct relationship. It is unclear based on these results alone, however, whether there is a causal link between these phenomena (or in which direction causation flows).

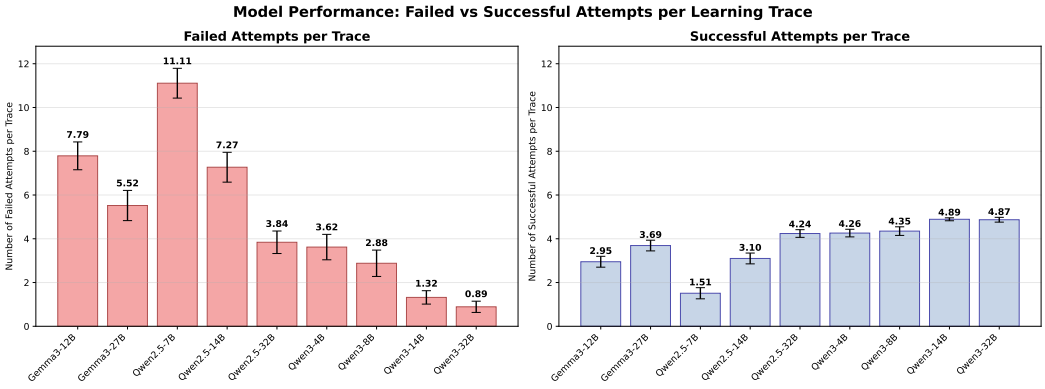

Figure 6: A display of the number of failed (left) and successful (right) attempts made by models during Interactive Task Learning. Learning occurred over $n = 5$ tasks, with a maximum of $m = 3$ attempts per task.

C.5 Token Usage Comparison between ITL and FSL

Displayed in Table 6 is a breakdown of the average number of tokens required for different portions of the learning trace $\mathcal{S}_{\texttt{FSL}}$ or $\mathcal{S}_{\texttt{ITL}}$. We multiply the average length (in tokens) of each category by the number of times it occurs within a trace (displayed in Figure 6) to obtain estimates of the mean number of tokens within each trace type. Note that this estimation process omits tokens utilized for

chat formatting, the task definitions themselves, and the $f_{\texttt{general}}$ prompt. The latter two are present in both FSL and ITL, and the former utilizes a relatively small number of tokens, so we omit it.

Table 6: A Breakdown of the number of tokens for different pedagogical paradigms. Each number displayed is the mean number of tokens rounded to the nearest integer. We report FSL token counts using the Qwen3 [56] tokenizer.

Model	Successful Attempt	Failed Attempt	Feedback	Full Trace
FSL	544	N/A	N/A	2719
ITL with Gemma3-12B	594	734	39	7891
ITL with Gemma3-27B	579	626	35	5912
ITL with Qwen2.5-7B	686	724	43	9619
ITL with Qwen2.5-14B	651	728	38	7699
ITL with Qwen2.5-32B	534	584	29	4741
ITL with Qwen3-4B	1502	1867	30	13397
ITL with Qwen3-8B	1404	1905	26	11777
ITL with Qwen3-14B	1390	2114	19	9707
ITL with Qwen3-32B	1563	2835	17	10229

We can draw a few conclusions from Table 6. Namely, due to the inclusion of failed attempts, ITL traces tend to be much longer than FSL traces. We anticipate this would enable FSL to outperform ITL in scenarios where the context length rather than the number of tasks is held constant. Furthermore, failed attempts tend to be longer than successful attempts across all models tested. This effect is especially pronounced within the Qwen3 family—all of which are reasoning models—which we observed have a tendency to meet the 4096 token limit when the reasoning process goes awry. Additionally, even the correct reasoning traces within ITL tend to be longer than the correct reasoning traces produced by Gemini 2.5 Flash for use in FSL, although this effect is only pronounced among the Qwen3 family. Finally, we note that feedback requires few tokens on average, making it an insignificant contributor to ITL's long context lengths.

D Prompts

D.1 Student Prompt

The following is an example of the prompt given to the student LLM. The student has access to the following core components:

1. The $f_{\texttt{general}}$ prompt, which provide details about the skills within this particular ITL world. See Appendix D.4 for a complete example.
2. `function_str`, which provides the Python code for the function that the student was tasked with executing.
3. The `input` that the student was tasked with transforming.

Student Prompt

```
{f_general}

{function_str}

Please predict transform({input})
Please think step by step and then conclude by responding with your answer.

After executing each line, you MUST output the line's result using this
syntax:
#L1 ['x', 'x', 'x', 'x', 'x', 'x', 'x', 'x']
```

```
Two things to note about this format:
- Replace the number 1 with the line number within the function. For
instance, if you are outputting the result from line 2, use #L2
- Replace the letters 'x' with the correct sequence.
```

D.2 Teacher Prompt

The following is an example of the prompt given to the teacher LLM. The teacher has access to the following core components:

1. The $f_{\texttt{general}}$ prompt, which provide details about the skills within this particular ITL world. See Appendix D.4 for a complete example.
2. `function_str`, which provides the Python code for the function that the student was tasked with executing.
3. The `input` that the student was tasked with transforming.
4. The student's response.
5. A ground truth response.
6. A ground truth judgment of the student response's `correctness` (either "Correct" or "Incorrect"), which is determined using a parser written in Python.

After the student completes an attempt a_j^i during learning, we parse the result to identify whether it made any errors. If our parser finds no errors, we simply respond with the string "Good job! You executed this task without any errors!" and move on to a new task. If our parser detects one or more errors—which happens frequently since an incorrect output on line i becomes an incorrect input to line $i + 1$—we provide feedback for the first error only.

We utilize Gemini 2.5 Flash [12] to provide natural language feedback regarding Localization as well as all three Explanation categories (using a json format). Our parser searches for strings with the format "`#L1 ['a', 'b', ...]`", which we instruct the student to output at the conclusion of each line within the transformation function (see Figure 1 for an example of a reasoning trace using this format). We do not explicitly generate separate Identification feedback since Localization feedback always specifies whether or not an error was made. We query Gemini using the prompt detailed in Appendix D.2, which gives it access to the student's response, a synthetically-generated ground-truth reasoning trace for the task, the parser's analysis (which identifies the line in which the student failed), and the templated general information string $f_{\texttt{general}}$ provided to the student.

To optimize the performance of Gemini, we do not obfuscate any of the skill names; all student generations are de-obfuscated before being shown to the teacher, and all teacher generations are re-obfuscated before being shown to the student.

Teacher Prompt

```
{f_general}

You are responsible for providing feedback to a student tasked with
predicting the output of the following function for the input {input}:
{function_str}

The student's response was as follows:
{student_response}

The ground truth reasoning trace that would have successfully solved this
problem is as follows:
{ground_truth}
```

```
The student may have made multiple errors, but your job is to provide
feedback specifically for the student's first error, which occurred at line
{step}.
The student responded with {faulty_list} when it should have responded with
{gt_list}.

Please provide feedback for the student. ONLY discuss this particular error,
not other errors that the student may have made.

Please address your feedback to the student, not to me:
- BAD: "The student's response is incorrect due to mistake x"
- GOOD: "Your response is incorrect due to mistake x"

You must analyze the student's error and provide feedback in exactly four
categories.
Each category serves a distinct purpose and should NOT overlap with others.

[Temporal Location]
This type of feedback ONLY identifies for the student *when/where* its error
occurred, without describing what went wrong.
- Must specify the line number in which the error occurred
- Must NOT describe the error itself or how to fix it
- Keep extremely brief (1 sentence maximum)
Example: "You made an error when trying to execute line 3."

[Error Identification]
Describes WHAT the student did wrong, without explaining why it was wrong or
how to fix it.
- Must be purely descriptive of the student's actual behavior
- Must NOT include normative details regarding what should have been done
instead
- Must NOT reference general rules or principles
Examples:
- "You treated 'E' as a lowercase letter when sorting."
- "You removed 'a' from the subsequence when filtering."
- "You left a copy of 'G' in its original location."

[Rule Identification]
States the general principle or rule that the student violated.
- Must be a generalizable rule that applies beyond this specific case
- Should explain the correct principle without referencing this particular
instance
- Focus on the "why" behind correct behavior
Examples:
- "When sorting, uppercase letters must be ordered after their lowercase
counterparts."
- "When filtering uppercase letters, leave all lowercase letters in the
subsequence."
- "When shifting a letter, you must always move rather than copy the letter."

[Corrective Information]
Provides the specific correct output for the erroneous step ONLY.
- Must show the correct intermediate result at the point of error
- Should NOT include full explanation or reasoning
- Keep focused on the immediate correction needed
Examples:
- "The correct sorted subsequence at line 3 is ['a', 'x', 'E']."
- "The correctly filtered subsequence at line 2 is ['a', 'v', 'i']."
- "The correct result for line 1 is ['c', 'w', 'b', 'G', 'z']."

Please output your response in a json format as follows:
{{
    "temporal_location" : "You made an error when trying to execute line 3.",
```

```
    "error_identification" : "You treated 'E' as a lowercase letter when
sorting.",
    "rule_identification" : "When sorting, uppercase letters must be ordered
after their lowercase counterparts.",
    "corrective_information" : "The correct sorted subsequence at line 3 is
['a', 'x', 'E']."
}}
Do not output ANYTHING except this json file. If you output additional text,
you will BREAK the code.
```

D.3 Few-Shot Learning Prompt

Few-Shot Learning Prompt

```
{f_general}

EXAMPLE 1:
You are tasked with predicting the output of a function named ‘transform‘
for a particular input.
Here is the function's definition:

{function_str_1}

Please predict transform({input_1})
Please think step by step and then conclude by responding with your answer.

After executing each line, you MUST output the line's result using this
syntax:
#L1 ['x', 'x', 'x', 'x', 'x', 'x', 'x', 'x']
Two things to note about this format:
- Replace the number 1 with the line number within the function. For
instance, if you are outputting the result from line 2, use #L2
- Replace the letters 'x' with the correct sequence.

{ground_truth_1}

EXAMPLE 2:
You are tasked with predicting the output of a function named ‘transform‘
for a particular input.
Here is the function's definition:

{function_str_2}

Please predict transform({input_2})
Please think step by step and then conclude by responding with your answer.

After executing each line, you MUST output the line's result using this
syntax:
#L1 ['x', 'x', 'x', 'x', 'x', 'x', 'x', 'x']
Two things to note about this format:
- Replace the number 1 with the line number within the function. For
instance, if you are outputting the result from line 2, use #L2
- Replace the letters 'x' with the correct sequence.

{ground_truth_2}

EXAMPLE 3:
...
```

```
Please now try executing a new function transform for a new input.
Here is the new function's definition:

{function_str}

Please predict transform({input})
Please think step by step and then conclude by responding with your answer.

After executing each line, you MUST output the line's result using this
syntax:
#L1 ['x', 'x', 'x', 'x', 'x', 'x', 'x', 'x']
Two things to note about this format:
- Replace the number 1 with the line number within the function. For
instance, if you are outputting the result from line 2, use #L2
- Replace the letters 'x' with the correct sequence.
```

D.4 General Task Description

The following is a templated prompt that provides a detailed description of every skill within a particular ListOps world $\mathcal{W}$. The skill descriptions themselves are static, however the list of skills, as well as their obfuscated names, changes depending on the ListOps world. Please note that, in the prompt fed to the student LLM, the names of the skills would be obfuscated to match the names used within the corresponding ListOps world.

This prompt is used in various experiments:

- It is used as a general description of the task domain $f_{\texttt{general}} = \pi_T(\mathcal{T})$ at the beginning of an ITL session.
- It is used as the Baseline Prompt and as a component of the Few-Shot Learning prompt for the experiments displayed in Figure 4.
- It is given to Gemini 2.5 Flash [12] to improve its feedback for the student.

Please note that the following prompt is an example that details several skills used within a particular ListOps World. Thus, not every skill in the universal skillset $\mathcal{S}$ is represented below.

General Task Description

```
Your task will involve knowledge about a custom API that utilizes the
following "atomic" functions:

`set_case(sequence, index=int, case=str)`
This function sets the letter at index `index` to either uppercase or
lowercase.
The variable `case` can be either 'upper' or 'lower'.
Here is an example of this function being executed:
```
set_case(['e', 'x', 'A', 'E', 'p', 'l', 'e'], index=3, case='upper')
This function requires us to set the letter at index 3 to uppercase.
Currently, index 3 is occupied by 'E'.
This letter is already uppercase, therefore we should not change anything.
We can simply return the input sequence:
['e', 'x', 'A', 'E', 'p', 'l', 'e']
```

`delete_index(sequence, index=int)`
This function identifies the letter corresponding to `index` and removes it
from the list.
Here is an example of this function being executed:
```

```
‘‘‘
delete_index([’e’, ’x’, ’A’, ’M’, ’p’, ’l’, ’e’], index=3)
This function requires us to delete the letter at index 3
At index 3 is the letter M
Removing this letter, we can return the following sequence:
[’e’, ’x’, ’A’, ’p’, ’l’, ’e’]
‘‘‘

‘change_case_at_index(sequence, index=int)‘
This function identifies the letter at ‘index‘ (zero-indexed).
It then changes the case of this letter (making it uppercase if it is
currently lowercase and vice versa).
It then returns the resulting sequence.
Here is an example of this function being executed:
‘‘‘
change_case_at_index([’e’, ’x’, ’A’, ’M’, ’p’, ’l’, ’e’, ’Y’], index=0)
The letter e is at index 0
This letter is currently lowercase, so we should convert it to uppercase
The uppercase version of e is E
Thus, the function would return [’E’, ’x’, ’A’, ’M’, ’p’, ’l’, ’e’, ’Y’]
‘‘‘

‘shift_letter(sequence, index=int, amount=int)‘
This function identifies the letter corresponding to ‘index‘ and shifts it
to the right by ‘amount‘.
A negative value for ‘amount‘ corresponds to a leftward shift.
Here is an example of this function being executed:
‘‘‘
shift_letter([’e’, ’x’, ’A’, ’M’, ’p’, ’l’, ’e’], index=4, amount=-2)
This function requires us to shift the letter at index 4 left by 2
At index 4 is the letter p
Shifting this letter to the left 2 would place it before ’A’
Thus, it would return the following sequence:
[’e’, ’x’, ’p’, ’A’, ’M’, ’l’, ’e’]
‘‘‘

‘duplicate_subsequence(sequence, index1=int, index2=int)‘
This function identifies the subsequence starting at index1 and ending at
index2 (inclusive, zero-indexed).
It then duplicates this subsequence and returns the result.
Here is an example of this function being executed:
‘‘‘
duplicate_subsequence([’e’, ’x’, ’A’, ’e’, ’p’, ’l’, ’e’], index1=1,
index2=3)
This subfunction requires us to duplicate the subsequence from index1=1 to
index2=3.
At index 1 is the letter x.
At index 3 is the letter e.
Thus, the subsequence is [’x’, ’A’, ’e’].
Duplicating this subsequence results in the subsequence [’x’, ’A’, ’e’, ’x’,
’A’, ’e’].
Re-inserting this into its original location within the list, this function
returns:
[’e’, ’x’, ’A’, ’e’, ’x’, ’A’, ’e’, ’p’, ’l’, ’e’]
‘‘‘

‘insert_letter(sequence, index=int, letter=character)‘
This function inserts ‘letter‘ at index ‘index‘.
Here is an example of this function being executed:
‘‘‘
insert_letter([’e’, ’x’, ’A’, ’E’, ’p’, ’l’, ’e’], index=4, letter=Y)
This function requires us to insert the letter Y at index 4.
Currently, index 4 is occupied by ’p’.
```

````
I will therefore insert 'Y' between 'p' and its predecessor 'E'.
Doing so will yield the following sequence:
['e', 'x', 'A', 'E', 'Y', 'p', 'l', 'e']
```

`set_index_to_value(sequence, index=int, letter=character)`
This function identifies the letter at `index` (zero-indexed).
It then sets this letter to `letter` and returns the resulting sequence.
Here is an example of this function being executed:
```
set_index_to_value(['e', 'x', 'A', 'M', 'p', 'l', 'e', 'Y'], index=7,
letter=e)
Currently, the letter Y is at index 7
This function replaces this letter with e
Thus, the function would return ['e', 'x', 'A', 'M', 'p', 'l', 'e', 'e']
```

`rotate_subsequence(sequence, amount=int, index1=int, index2=int)`
This function identifies the subsequence starting at index1 and ending at
index2 (inclusive, zero-indexed).
It then performs a circular shift of the subsequence.
A positive value for `amount` indicates that the letters within the
subsequence ought to be shifted to the right by that amount.
A negative value for `amount` indicates that the letters ought to be shifted
to the left.
Here is an example of this function being executed:
```
rotate_subsequence(['e', 'x', 'A', 'e', 'p', 'l', 'e'], amount=-1, index1=1,
index2=3)
This subfunction requires us to rotate the subsequence from index1=1 to
index2=3.
At index 1 is the letter x.
At index 3 is the letter e.
Thus, the subsequence is ['x', 'A', 'e'].
Rotating this subsequence by -1 indicates a leftward shift of 1.
This results in the subsequence ['A', 'e', 'x'].
Re-inserting this into its original location within the list, this function
returns:
['e', 'A', 'e', 'x', 'p', 'l', 'e']
```
````
````