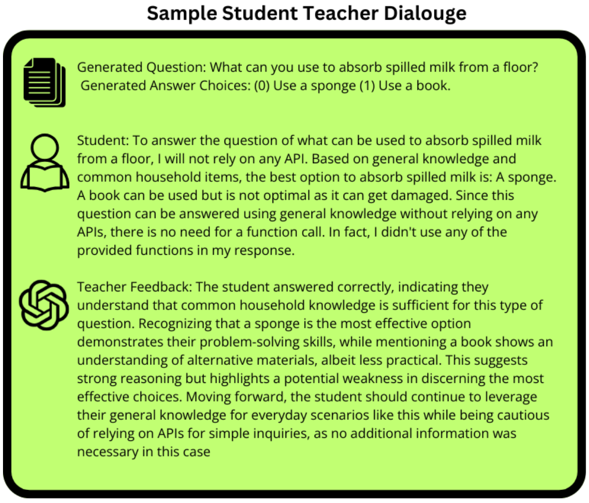
Self Knowledge-Tracing for Tool Use (SKT-Tool): Helping LLM Agents Understand Their Capabilities in Tool Use

Joshua Vigel[*] **Renpei Cai**[*] **Eleanor Chen** **Anish Neema**
Austen Liao[†] **Kevin Zhu**[†] **Sean O'Brien**[†]
Algoverse AI Research
`nasevate@gmail.com, kevin@algoverse.us`

Abstract

Large Language Models (LLMs) enhanced with tool use and APIs improve task performance but often misuse them, leading to inefficiency and unnecessary cost. We propose Self Knowledge-Tracing for Tool Use (SKT-Tool), a method enabling LLMs to assess their capabilities and make informed API usage decisions using knowledge tracing (KT). Our teacher-student framework helps LLMs optimize API calls in real-time without fine-tuning. Experiments across multiple datasets show that SKT-Tool significantly reduces API calls while maintaining accuracy, offering a scalable and cost-effective solution for tool-augmented LLMs. We conclude by analyzing shortcomings in this method and identifying directions for future work.



Sample Student Teacher Dialouge

Generated Question: What can you use to absorb spilled milk from a floor?
Generated Answer Choices: (0) Use a sponge (1) Use a book.

Student: To answer the question of what can be used to absorb spilled milk from a floor, I will not rely on any API. Based on general knowledge and common household items, the best option to absorb spilled milk is: A sponge. A book can be used but is not optimal as it can get damaged. Since this question can be answered using general knowledge without relying on any APIs, there is no need for a function call. In fact, I didn't use any of the provided functions in my response.

Teacher Feedback: The student answered correctly, indicating they understand that common household knowledge is sufficient for this type of question. Recognizing that a sponge is the most effective option demonstrates their problem-solving skills, while mentioning a book shows an understanding of alternative materials, albeit less practical. This suggests strong reasoning but highlights a potential weakness in discerning the most effective choices. Moving forward, the student should continue to leverage their general knowledge for everyday scenarios like this while being cautious of relying on APIs for simple inquiries, as no additional information was necessary in this case

Figure 1: Example dialogue from the synthetic generation process. This is a sample of the interaction string that is passed onto our student model for reference.

1 Introduction

Large Language Models (LLMs) have recently achieved a great number of remarkable abilities when incorporated with external tools like Application Programming Interfaces (APIs) (Parisi et al., 2022; Patil et al., 2023). However, two problems emerge: current LLM agents often struggle choosing which tools to use on different problems (Li et al., 2023), and also that much of current research focuses solely on the API choice (Qin et al., 2023; Li et al., 2023; Chen et al., 2024; Tang et al., 2023), where an LLM indiscriminately calls tools during tests, which may not align with real-world scenarios (Ning et al., 2024; Qiao et al., 2023). This can result in redundant API calls, leading to extra computation and even possible monetary loss in cases where every API call costs money, thus substantially reducing the usefulness of tool-augmented LLM Agents in real-world applications. Furthermore, the introduction of tools tends to decrease the performance of LLMs across general datasets when they have to decide whether or not tool use is appropriate (Ning et al., 2024).

Existing approaches fine-tune LLMs to choose APIs (Ning et al., 2024; Schick et al., 2023; Hao et al., 2023; Yang et al., 2023) but overlook the model's inherent capabilities.

In this paper, we propose Self Knowledge-Tracing for Tool use (SKT-Tool) which offers a solution involving a Student-Teacher framework.

2 Related Work

2.1 Tool Use in Large Language Models

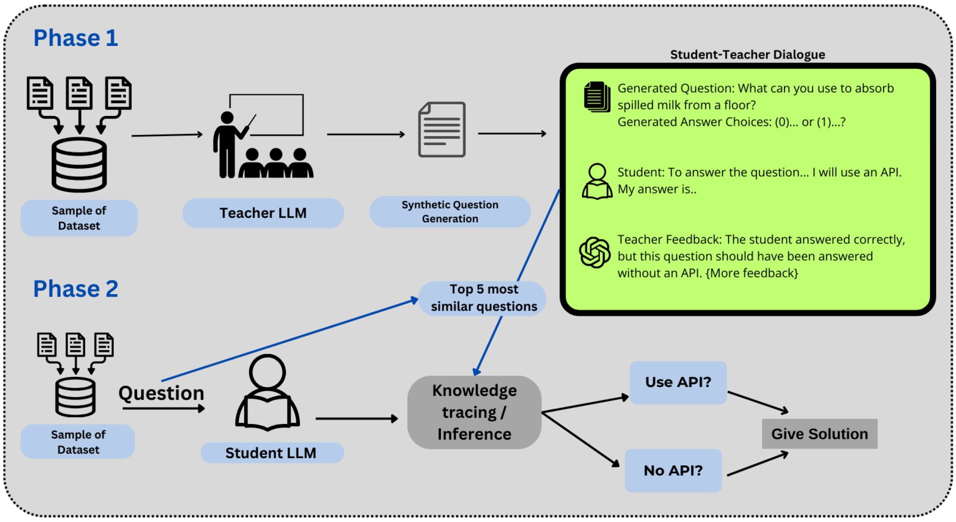
WTU-Eval (Ning et al., 2024) introduces a benchmark designed to evaluate whether LLMs can discern their ability boundaries and decide on tool usage accordingly. The findings of WTU-Eval highlight a critical gap: most current approaches assume that an LLM must invoke an API without first evaluating whether the model is capable of solving the task on its own. The study demonstrated that LLMs frequently misuse tools when unnecessary,

---

[*] Lead Author

[†] Senior Author

The Sixth Workshop on Insights from Negative Results in NLP, pages 150–156
May 4, 2025 ©2025 Association for Computational Linguistics



Generated Question: What can you use to absorb spilled milk from a floor?
Generated Answer Choices: (0)... or (1)...?

Student: To answer the question... I will use an API. My answer is..

Teacher Feedback: The student answered correctly, but this question should have been answered without an API. {More feedback}

Figure 2: Visualization of our system's workflow. In Phase 1, the teacher generates n questions to assess the student's ability. In Phase 2, the student answers actual task/dataset questions. Then, the top-k most similar Phase 1 interactions are retrieved and appended to the question. The student model then performs on its own capabilities before attempting to solve the question.

leading to performance degradation in tasks that do not require external computation.

TRICE (Qiao et al., 2023) introduces a method of teaching LLMs to use tools through continuous feedback. Their findings also introduce the problem of LLMs dropping in question-answering accuracy when introduced to tools due to confusion over how to use them, and they employ an extensive feedback mechanism that relies on instruction-tuning and Reinforcement Learning to align the LLMs behavior towards appropriate tool use.

SKT-Tool differs by employing knowledge tracing, where the Teacher LLM generates targeted questions to assess the Student's capabilities, unlike prior methods relying solely on task feedback. It optimizes through Inference-Time Optimization instead of fine-tuning or reinforcement learning. Unlike TRICE, which focuses only on accuracy, SKT-Tool also reduces API calls while aiming to maintain accuracy.

2.2 Knowledge Tracing for tool use

Our study explores the concept of knowledge tracing (KT), a technique originally used in education to track and predict students' learning progress (Corbett and Anderson, 2005). Given that LLMs often struggle understanding their capabilities in the context of Tool Use, it is intuitive to utilize knowledge tracing in this scenario.

By applying KT principles, an LLM can evaluate its own strengths and weaknesses before deciding to call an external API. For example, if a model consistently struggles with math problems but performs well in general reasoning, it can determine whether using a calculator API is necessary. This self-awareness reduces unnecessary API calls and ensures that external tools are only used when needed.

3 Method

In this section, we detail our technical approach to creating SKT-Tool. Our framework is divided into 2 subsections: (1) the teacher model and synthetic data generation; (2) the student model, Retrieval-Augmented Generation (RAG) and knowledge tracing, Fig. 2 provides a visual overview of the system architecture. As seen in Fig. 2, we set up two language models to converse with each other, where one acts as the **teacher** and the other is the **student**. Our framework aims to improve the student's model ability to decide whether or not it should call an API and answer QA datasets correctly with APIs.

3.1 Teacher Model and Synthetic Data Generation (Phase 1)

We define the Teacher Model to be a more capable LLM with the following responsibilities:

- **Synthetic Data Generation.** The Teacher receives a description of an existing task, along with k samples from the task dataset. The teacher is also provided with the list of tools/APIs that the student will have access to when approaching this task. The teacher is then prompted to generate n synthetic questions that test the student in such a way as to gain insight into the student's capabilities relevant to the task.

- **Probing the Student.** The teacher model asks the generated questions to the student, storing feedback on the correctness of the student's answers. This first part of the conversation chain, where the teacher probes the student's answering capabilities, serves as the basis for our KT mechanism and effectively avoids the cold start problem by knowledge tracing before evaluation. We refer to this synthetic question history as S.

The teacher's synthetic data generation and probing process will be referred to as **Phase 1** of the framework.

3.2 Evaluation of Student Model (Phase 2)

We define the Student Model to be the model that will be evaluated on the specified task, and may be the same model as the teacher model. In phase 2, the Student Model will use the generated conversation history to effectively choose API usage.

- **Retrieval Augmented Generation (RAG)** (Gao et al., 2023) **for knowledge tracing.** As shown in Figure 2, for each question q asked to the student during the evaluation step, we use RAG to identify the five most semantically similar questions to q in S. We append the associated conversations of these questions to the question prompt in few-shot fashion.

4 Experiments

4.1 Settings

To evaluate our method's effectiveness in different scenarios, we use a single dataset as the task per experiment. This *task*, from which a sample is given to the teacher model, is the same as the one to be solved by the student.

4.2 Datasets

To simulate questions asked by real-life users to LLMs, we use two types of datasets in our testing (where dev sets are used whenever possible):

- **General QA Domain.** We select four knowledge-based datasets with diverse question types: MLQA (Lewis et al., 2019), TriviaQA (Joshi et al., 2017), PIQA (Bisk et al., 2019), and HotpotQA (Yang et al., 2018). For HotpotQA, we exclude the context paragraphs, including only the question in the query, as the student model has access to the WikiSearch tool.

- **Math QA Domain.** To account for problem-solving tasks, we focus on math-based datasets as they test quantitative and logical reasoning abilities. Our math dataset pool includes AQUA-RAT (Ling et al., 2017), GSM8K (Cobbe et al., 2021), MathQA (Amini et al., 2019), and SAT-Math from AGIEval (Zhong et al., 2023).

For each task, we sample 250 questions from relevant datasets in both the **General** and **Math** categories.

4.3 Tools

We define our tool pool to include APIs that are most likely to be used based on the datasets above: **Google Translation API**[1], **WolframAlpha API**[2],**Bing Web Search API**[3], and **Wikipedia Search API**[4].

4.4 Models

We compare results from our method using three models with increasing capability as our student: Llama 3.1-8B (Grattafiori et al., 2024), GPT-3.5 (OpenAI, 2023), and GPT-4o (OpenAI, 2024), while our teacher model remains GPT-4o throughout experimentation.

4.5 Metrics

We evaluate our results using 3 different metrics:

- **Accuracy.** Accuracy of the Student on the given task. All datasets consist of 250 questions.

- **AccN.** Accuracy of the Student accounted for unneeded API calls, which we refer to $AccN$. The Student is considered to be incorrect if they use an API on a question that was previously shown to not require an API, regardless if their answer was correct or not.

---

[1] https://cloud.google.com/translate/docs/reference/rest

[2] https://developer.wolframalpha.com/

[3] https://www.microsoft.com/en-us/bing/apis/bing-web-search-api

[4] https://www.mediawiki.org/wiki/API:Search

Task - LLM	**Metric**	Baseline	**k** ($n = 32$)		**n** ($k = 16$)	
			4	**32**	**8**	**64**
PIQA Llama-3.1	Accuracy	0.536	0.516	0.512	0.536	0.496
	API Calls	177	114	104	48	43
	AccN	0.4	0.38	0.412	0.448	0.428
MLQA GPT 4o	Accuracy	0.912	0.832	0.832	0.884	0.868
	API Calls	43	7	4	12	16
	AccN	0.812	0.824	0.82	0.848	0.836
MLQA GPT 3.5	Accuracy	0.628	0.668	0.636	0.66	0.652
	API Calls	137	16	17	9	14
	AccN	0.432	0.652	0.636	0.652	0.64
PIQA GPT 3.5	Accuracy	0.756	0.792	0.772	0.776	0.756
	API Calls	1	0	0	0	0
	AccN	0.752	0.792	0.772	0.776	0.756
AQUA-RAT GPT 3.5	Accuracy	0.576	0.46	0.512	0.472	0.5
	API Calls	67	22	18	31	23
	AccN	0.514	0.448	0.508	0.448	0.488

Table 1: Baseline vs. SKT-Tool varying k **and** n, 250 Questions per task.

- **API Calls.** The total number of questions in which the student used an API throughout an entire task.

4.6 Baseline

To evaluate baseline results, we give the LLM access to tools and prompt it to answer the dataset with no additional instructions. The baseline performance aims to evaluate the ability of LLMs to decide on their own when to use tools.

4.7 Implementation Details

To test the generalizability of our model, we evaluate our metrics across two different variables: k and n, where k is the number of samples from the task given to the teacher before question generation, and n is the number of samples the teacher generates. We test values of $k \in \{4, 32\}$ while keeping $n = 32$, and values of $n \in \{8, 64\}$ while keeping $k = 16$.

5 Results and Analysis

5.1 Across the Datasets

Our method significantly reduces unnecessary API calls across multiple datasets. In MLQA, GPT-3.5 reduced API calls by up to 85%, while GPT-4o achieved a 90% reduction. This suggests that the model learns to answer multilingual questions independently rather than over-relying on translation tools. However, accuracy improvements were inconsistent, particularly for GPT-4o, where MLQA accuracy slightly declined.

For PIQA, the impact was less pronounced. API calls dropped by 73% for Llama3.1, but the relevance of the synthetic conversation history varied, leading to limited accuracy gains. Similarly, in AQUA-RAT, API usage decreased, but accuracy also declined—likely due to the teacher model generating misleading mathematical feedback. Furthermore, we noticed a significant difference in API calls for PIQA during our ablation tests when varying k versus n. This could be due to the combination of RAG, the quality of synthetically generated questions, and the LLM repeatedly calling the API for undesirable results.

Overall, our framework successfully reduces redundant API calls, demonstrating its effectiveness in optimizing tool use. However, we observed no consistent accuracy improvements. The relationships between accuracy, k, and n vary across datasets, indicating the need for further experimentation to refine synthetic question generation and improve overall model performance.

6 Conclusion

We introduce a framework that helps LLMs assess their API usage, reducing calls by up to 50% while maintaining accuracy. After running multiple tests on the interaction history generated by the teacher model, we found that each interaction history appended incurred an average of 300 additional input tokens. The additional input tokens for 5 questions through RAG are not high enough to exceed smaller models' context windows. Though accuracy improvements are limited, our approach enables smaller models to use tools efficiently, lowering costs and expanding practical applications, particularly for weaker, newly-developed LLMs. We also believe that future work in refining the prompting methods for teacher feedback will help improve the student's final accuracy.

7 Limitations and Future Work

The efficacy of SKT-Tool is heavily dependent on the capability of the student model. Our method relies on long question histories being appended to prompts, and smaller models like Llama 3.1 8B struggle due to the large context. This is especially true for Math datasets. Our results also may significantly change when not using a powerful teacher model such as GPT-4o, which we used. Due to time constraints, we were unable to run full experiments on "Multi" setting tests, but we plan to in the future. We also believe the inconsistencies in results indicate a flaw in our method of prompting and implementation, which we aim to fix.

In the future, plan to further polish our framework's prompting methods to ensure optimal synthetically generated questions that target concepts within the task. We also aim to test the effects of our frameworks on the "Multi" setting, and plan to test our framework on more open-sourced models, a wider variety of tools, etc. to have a better understanding of its efficacy.

Furthermore, we believe that, in future research, more standardized benchmarks can be set for LLMs' decisions on whether or not tool usage is necessary. We also believe that future research should study how knowledge tracing can be a powerful tool when applied to LLMs as scaffolding rather than merely to humans.

References

Aida Amini, Saadia Gabriel, Shanchuan Lin, Rik Koncel-Kedziorski, Yejin Choi, and Hannaneh Hajishirzi. 2019. Mathqa: Towards interpretable math word problem solving with operation-based formalisms. In *North American Chapter of the Association for Computational Linguistics*.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. 2019. Piqa: Reasoning about physical commonsense in natural language. In *AAAI Conference on Artificial Intelligence*.

Yanfei Chen, Jinsung Yoon, Devendra Singh Sachan, Qingze Wang, Vincent Cohen-Addad, Mohammad-Hossein Bateni, Chen-Yu Lee, and Tomas Pfister. 2024. Re-invoke: Tool invocation rewriting for zero-shot tool retrieval. In *Conference on Empirical Methods in Natural Language Processing*.

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*, abs/2110.14168.

Albert T. Corbett and John R. Anderson. 2005. Knowledge tracing: Modeling the acquisition of procedural knowledge. *User Modeling and User-Adapted Interaction*, 4:253–278.

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Qianyu Guo, Meng Wang, and Haofen Wang. 2023. Retrieval-augmented generation for large language models: A survey. *ArXiv*, abs/2312.10997.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*. Accessed: 2025-03-15.

Shibo Hao, Tianyang Liu, Zhen Wang, and Zhiting Hu. 2023. Toolkengpt: Augmenting frozen language models with massive tools via tool embeddings. *ArXiv*, abs/2305.11554.

Mandar Joshi, Eunsol Choi, Daniel S. Weld, and Luke Zettlemoyer. 2017. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. *ArXiv*, abs/1705.03551.

Patrick Lewis, Barlas Oğuz, Ruty Rinott, Sebastian Riedel, and Holger Schwenk. 2019. Mlqa: Evaluating cross-lingual extractive question answering. *ArXiv*, abs/1910.07475.

Minghao Li, Feifan Song, Yu Bowen, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. 2023. Api-bank: A comprehensive benchmark for tool-augmented llms. In *Conference on Empirical Methods in Natural Language Processing*.

Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. 2017. Program induction by rationale generation: Learning to solve and explain algebraic word problems. In *Annual Meeting of the Association for Computational Linguistics*.

Kangyun Ning, Yisong Su, Xueqiang Lv, Yuanzhe Zhang, Jian Liu, Kang Liu, and Jinan Xu. 2024. Wtu-eval: A whether-or-not tool usage evaluation benchmark for large language models. *ArXiv*, abs/2407.12823.

OpenAI. 2023. Gpt-3.5: Large language model. [Online; accessed March 2025].

OpenAI. 2024. Gpt-4o: Large language model. [Online; accessed March 2025].

Aaron Parisi, Yao Zhao, and Noah Fiedel. 2022. Talm: Tool augmented language models. *ArXiv*, abs/2205.12255.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. 2023. Gorilla: Large language model connected with massive apis. *ArXiv*, abs/2305.15334.

Shuofei Qiao, Honghao Gui, Huajun Chen, and Ningyu Zhang. 2023. Making language models better tool learners with execution feedback. *ArXiv*, abs/2305.13068.

Yujia Qin, Shi Liang, Yining Ye, Kunlun Zhu, Lan Yan, Ya-Ting Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Runchu Tian, Ruobing Xie, Jie Zhou, Marc H. Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2023. Toolllm: Facilitating large language models to master 16000+ real-world apis. *ArXiv*, abs/2307.16789.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *ArXiv*, abs/2302.04761.

Qiaoyu Tang, Ziliang Deng, Hongyu Lin, Xianpei Han, Qiao Liang, Boxi Cao, and Le Sun. 2023. Toolalpaca: Generalized tool learning for language models with 3000 simulated cases. *ArXiv*, abs/2306.05301.

Rui Yang, Lin Song, Yanwei Li, Sijie Zhao, Yixiao Ge, Xiu Li, and Ying Shan. 2023. Gpt4tools: Teaching large language model to use tools via self-instruction. *ArXiv*, abs/2305.18752.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Conference on Empirical Methods in Natural Language Processing*.

Wanjun Zhong, Ruixiang Cui, Yiduo Guo, Yaobo Liang, Shuai Lu, Yanlin Wang, Amin Saied Sanosi Saied, Weizhu Chen, and Nan Duan. 2023. Agieval: A human-centric benchmark for evaluating foundation models. *ArXiv*, abs/2304.06364.

A Appendix

A.1 A. Datasets and Task Details

This section details the datasets used in our experiments, including question distribution and preprocessing.

A.1.1 General QA Datasets

Dataset	**Domain**	**Sample Size**
MLQA	Multilingual QA	250
TriviaQA	Knowledge-based QA	250
PIQA	Physical Commonsense QA	250
HotpotQA	Multi-hop Reasoning	250
General-Multi	Mixed (Shuffled)	200 (50 each)

Table 2: General QA datasets used in experiments. Each single dataset uses 200 samples, while the multi-task setting takes 50 from each and shuffles them.

Dataset	**Domain**	**Sample Size**
AQUA-RAT	Algebra, Arithmetic QA	250
GSM8K	Grade-School Math QA	250
MathQA	General Math QA	250
SAT-Math (AGIEval)	Standardized Test Math	250
Math-Multi	Mixed (Shuffled)	200 (50 each)

Table 3: Math QA datasets used in experiments. Each single dataset uses 200 samples, while the multi-task setting takes 50 from each and shuffles them.

A.1.2 Math QA Datasets

A.2 Prompt Templates

We used structured prompts for both the teacher and student LLMs. Below, we provide an overview of these templates.

A.2.1 Teacher Model Prompt For Generating Synthetic Questions

Here's the prompt we used to ask the teacher model to generate test questions and their corresponding answers:

```
You are a teacher with one student.
Your student is going to take a test.
Some of the questions in
the test require APIs, and
some are answerable by the student
itself.

Because we want to minimize API calls,
you will need to learn what questions
the student needs an API to answer,
and which it can answer itself.
We also want the student to answer
questions correctly.
To do this, you must learn what
questions can be solved by each API.

To do this, you will need to
learn the capabilities of the student.
For example, you will need to
learn the student's proficiency in
math to be able to determine in
which cases the student needs a
calculator API.

To learn the student's capabilities,
you will have the opportunity to
give your student a 'pre-test'.
You must generate question-answer
pairs for this pre-test such
that the student's answers teach you
about the student's capabilities
```

```
and when it needs an API or not,
and also how specific APIs can
help solve types of questions.

To make sure that the pre-test
questions give relevant information,
you will be given a small sample
of the questions from the test.
You are not to copy the questions:
merely generate similar ones that
can be compared with.

The process for generating the
pre-test will consist of these steps:
1: You are given a sample of the
dataset, and the number of
questions to generate.
2: You will generate the required
number of questions to learn
the student's capabilities. In each
question, you should tell the
student which API to use, or
not to use one at all.

{Few-shot Examples}

Here is a description of the task:
{DESCRIPTION}

Here is a sample of the dataset:
{K SAMPLE QUESTIONS}

Here is the list of APIs that
the student has access to and
their descriptions:
{API List}

Generate {N} questions and do
not generate anymore. You can
write your thoughts but only include
them at the beginning and nowhere
else. When you write your questions,
start each question with a
singular newline so that the user
can use .split to get the text
for each question specifically.
Do not include any other newlines
except when before a question.
```

A.2.2 Student Model Prompt (Baseline)

Here's our basic prompt for the student when establishing our baseline:

```
You are an AI-Student tasked with
answering questions to the best
of your ability.
You have access to tools (APIs)
but should only use them if
absolutely necessary.
Your goal is to minimize API
usage by answering questions
independently whenever possible.

### Rules for Deciding API Usage:
1. Only use an API if you cannot
answer the question without it.
2. Ensure the tool available is
relevant to the question.
3. Use only one API per question,
choosing the most suitable one.

Let's begin!
```

A.2.3 Student Model Prompt (SKT-Tool)

In addition to the baseline prompt, we include the interaction history during **Phase 1** (3.1):

```
Here is a list of previous questions
you have answered, and some feedback
from an external teacher.
Use this information to determine
your own capabilities, and whether
to use an API on your next question.
Please only use it if you feel
that it's relevant. You may also use
it as an example for how to answer
questions. If you determine API
use is necessary, use one.

Here's the list of past interactions:
{Top 5 most similar synthetic questions}
```