# Roadmap for a General-Purpose In-Context Learning Simulator

## Overview and Goals

We propose a roadmap to transform a real-time tutor platform into a **general-purpose In-Context Learning (ICL) simulator**. The goal is to support all domains (math, programming, medicine, policy, etc.) in a *closed-book* fashion – meaning models must rely only on provided context (notes, examples) rather than pretrained recall. Key objectives include anonymizing domain-specific terms to prevent memorized recall, integrating simulated “student” agents (current LLMs or future reasoning agents), supporting diverse task types (beyond multiple-choice to coding, proofs, tables, short answers), and implementing robust evaluation mechanisms (learning curves, effects of explanations, spacing of examples, error-driven learning signals, etc.). The system should allow fine-grained control (via **“dials”**) over memory usage, context placement, reflection frequency, and explanation timing. The roadmap below outlines necessary architectural changes, simulator extensions, abstraction layers for tasks/domains, logging and analysis tools, as well as experimental protocols for benchmarking ICL across domains under anonymized, closed-book conditions.

## Architecture Changes Needed

To evolve the platform, significant architectural modifications are required to handle multi-domain ICL and complex simulation dynamics. The new architecture should be modular and extensible, comprising the following components:

* **Core Simulation Orchestrator:** Coordinates the interaction between a *tutor module* (which provides context/examples, prompts, and possibly feedback) and a *learner module* (the simulated user/agent). This orchestrator manages turn-by-turn dialog or prompt construction in real-time, ensuring the system can operate in interactive mode (for a live tutor feel) or batch mode (for automated evaluations).
* **Pluggable Learner (Student) Module:** A flexible interface to plug in any model or agent as the “student.” Initially this may be current LLMs (e.g. GPT-4, GPT-5-nano, Llama 2/3 variants), but it must accommodate future reasoning agents that use tools or multi-step reasoning. The interface should allow both *stateless* models (pure prompt-in, completion-out) and *stateful agents* (that maintain some working memory or chain-of-thought). This design ensures we can simulate today’s models and easily integrate more advanced agents later. For example, TutorGym provides a standard interface where different AI agents (LLMs, RL agents, etc.) can be evaluated within the same tutoring framework[[1]](https://arxiv.org/html/2505.01563v1#:~:text=,agents%20directly%20learn%20from%20ITS)[[2]](https://arxiv.org/html/2505.01563v1#:~:text=initial%20evaluation%2C%20we%20find%20that,context%20learning). By decoupling the learner API, the simulator could swap a GPT-based student for a bespoke reasoning agent without altering the surrounding system.
* **Domain & Content Module:** A new layer to handle **multi-domain content** and **anonymization**. This module stores domain-specific knowledge representations (e.g. glossaries of terms, formulae, code libraries, policy documents) and methods to *anonymize or alias* them. Domain terms (e.g. medical conditions, proper nouns, math theorems) should be systematically replaced with placeholders or neutral identifiers in both the context provided to the model and in the questions, to force the model to rely only on context. Prior work shows that when problems are altered to replace recognizable constants or names with abstract symbols, LLM performance drops significantly – indicating that models were otherwise relying on memorized solutions[[3]](https://arxiv.org/html/2505.01563v1#:~:text=High%20accuracies%20on%20these%20benchmarks,solving). For instance, the GSM8K-Symbolic benchmark replaced problem-specific names/numbers with tokens and added distractors, which “substantially reduced LLM accuracy, suggesting that LLMs rely upon memorizing solutions instead of general-purpose problem-solving”[[3]](https://arxiv.org/html/2505.01563v1#:~:text=High%20accuracies%20on%20these%20benchmarks,solving). Our simulator will avoid such shortcut recall by dynamic anonymization. The content module would maintain a mapping from original terms to anonymized tokens per session (so that after evaluation, answers can be de-anonymized for human inspection if needed).
* **Task Abstraction Layer:** A unified representation for tasks of various types, decoupling the task logic from the core engine. Each task type (e.g. multiple-choice quiz, code debugging, formal proof, table QA, short-answer question) is defined by:
* A **prompt template** (how the problem and any provided context or examples are formatted),
* An **expected response format** (e.g. selected choice, free-form text, code snippet, step-by-step proof),
* An **evaluation function** for scoring (discussed more under evaluation). This layer allows adding new task types via configuration or plugins, without rewriting core code. For example, one can define a “code repair” task template that provides a broken code snippet and the instruction to fix it, and tie it to an evaluator that runs unit tests on the model’s code output[[4]](https://www.datacamp.com/tutorial/humaneval-benchmark-for-evaluating-llm-code-generation-capabilities#:~:text=function%20signature%2C%20docstring%2C%20body%2C%20and,7%20tests%20per%20problem)[[5]](https://www.datacamp.com/tutorial/humaneval-benchmark-for-evaluating-llm-code-generation-capabilities#:~:text=samples%20for%20a%20problem%20passes,the%20unit%20tests). Likewise, a “proof step” task can be defined to expect a formal proof line; its evaluator could call a theorem prover (like Lean or Coq) to verify the step’s correctness[[6]](https://arxiv.org/html/2404.12534v1#:~:text=assistants%20easier%20to%20use%20by,improving%20proof%20automation). This abstraction ensures **rich task support** – from MCQs to complex reasoning – by plugging in appropriate context and evaluators.
* **Memory Management Component:** In a real-time tutoring dialogue, the system must manage what context the model “remembers.” Rather than always sending the entire conversation, the simulator will enforce a **memory policy** (adjustable as a dial). This could involve a sliding context window, summary of earlier interactions, or retrieval of relevant notes. Architecturally, this means implementing a *context buffer* that can be truncated or summarized according to policy. For instance, to simulate limited working memory, the system might only include the last N turns of dialogue as context for the model, or explicitly omit older examples unless “review” is triggered. The importance of context positioning is known – LLMs have a **recency and primacy bias**, giving more weight to information at the beginning or end of their input and often forgetting middle content[[7]](https://cs.stanford.edu/~nfliu/papers/lost-in-the-middle.arxiv2023.pdf#:~:text=long%20input%20contexts,models%20use%20their%20input%20context)[[8]](https://cs.stanford.edu/~nfliu/papers/lost-in-the-middle.arxiv2023.pdf#:~:text=performance%20curve%E2%80%94models%20are%20better%20at,2023%3B%20Schick). Our architecture can leverage this: e.g. by placing critical example(s) at the end of the prompt to exploit recency, or by randomizing order to test position effects. We may also integrate an **external memory** (like a vector database for semantic recall of past notes) as an optional module, especially for future agent-based learners that could choose to query their memory. (Notably, giving LLM agents a form of external memory or scratchpad is a known approach to improve their performance on complex tasks[[9]](https://arxiv.org/html/2405.06682v1#:~:text=To%20improve%20their%20performance%2C%20we,42%2C%2010%20%2C%20%2043).)
* **Evaluation & Scoring Module:** A dedicated subsystem to perform **witness-gated scoring** (explained in detail later). It will take the model’s answer and the context (e.g. reference notes, examples given) and output a judgment of correctness that accounts for evidence and consistency. This module may call different evaluators depending on task: for code, run test cases; for math proofs, use a proof checker; for QA, match against ground-truth answers or use semantic similarity with reference solution. Crucially, it will also verify that the answer is *grounded in the provided context*, and optionally perform consistency checks like re-asking the model. The architecture should allow chaining these checks – e.g. first auto-grade the answer, then check coverage of supporting facts from notes, then do a self-consistency validation – to decide whether the answer truly “passes.” By segregating this into a module, we can update or enhance scoring criteria without affecting the rest of the system.

Overall, these changes promote a **modular, extensible architecture**. The simulation orchestrator connects tutors, learners, and tasks in a generic way, while specialized modules handle domain content and anonymization, task definitions, memory context, and evaluation logic. This design will scale to *hundreds of domains and tasks*, similar to how TutorGym scaled to 223 tutor domains by providing a common interface for different curricula[[10]](https://arxiv.org/html/2505.01563v1#:~:text=TutorGym%20provides%20exciting%20new%20opportunities,tutors%20and%20as%20simulated%20learners). It will also enable easy experimentation: e.g. swapping in a new large model, or toggling a memory limitation, can be done by adjusting module configurations rather than rewriting the system.

## Simulator Extensions: Dials and Configurable Policies

To study in-context learning under various conditions, the simulator will expose “dials” – configurable settings that modify the learning environment or the agent’s behavior. These extensions allow researchers or instructors to simulate different pedagogical strategies and cognitive conditions. Key dials include:

* **Memory Policy:** Controls how much context the simulated student retains. Settings could range from *fully cumulative* (the model always sees all previous notes and interactions, bounded only by token limit) to *windowed memory* (only the most recent $k$ examples or turns) to *expiring memory* (content older than a certain number of turns is dropped or compressed). This dial lets us model forgetting or limited working memory. For example, one might simulate spaced repetition by **re-introducing a prior example** after some rounds where it was hidden, to see if performance improves – akin to human spaced practice. We expect that if a model is not reminded of a concept for several turns, its performance might drop, mimicking forgetting, and this dial would help quantify that. (Note: LLMs don’t truly update weights with each example, but by manipulating the prompt memory we can approximate short-term learning vs forgetting.)
* **Context Placement:** A setting to vary the ordering and position of demonstration examples or notes in the prompt. This can test primacy vs recency effects. For instance, the simulator could randomly shuffle example order, or specifically place the most relevant note right before the query versus at the beginning. Research has found that LLM performance often follows a U-shaped curve: highest when relevant information is at the start or end of the context, and lower when that info is buried in the middle[[7]](https://cs.stanford.edu/~nfliu/papers/lost-in-the-middle.arxiv2023.pdf#:~:text=long%20input%20contexts,models%20use%20their%20input%20context)[[8]](https://cs.stanford.edu/~nfliu/papers/lost-in-the-middle.arxiv2023.pdf#:~:text=performance%20curve%E2%80%94models%20are%20better%20at,2023%3B%20Schick). By toggling this dial, one can evaluate how robust the learner is to context position – an important aspect for long prompts. The system might also allow *interleaving* unrelated filler content to test if the model can skip irrelevant info or if it gets “lost in the middle.” This dial helps experiment with prompt engineering strategies within the simulator.
* **Reflection/Explanation Mode:** Allows configuring if and when the student model engages in self-reflection or explanation. One setting could force the model to output a short explanation or chain-of-thought before giving the final answer (or the tutor to ask *“Please explain your reasoning”*). Another variant is *post-answer reflection*: if the model’s answer is wrong, the simulator can prompt the model to analyze its error and try again (simulating feedback-driven learning). Recent studies show that LLMs can **significantly improve their problem-solving performance through self-reflection** – i.e., when made to reflect on mistakes and given a chance to correct them, their accuracy increases[[11]](https://arxiv.org/html/2405.06682v1#:~:text=models%20%28LLMs%29%20on%20problem,determine%20their%20individual%20contribution%20to). We can incorporate this by toggling a dial: e.g. “reflection enabled after incorrect answers = on.” We can also control the style: the simulator might give the model a scaffold like *“Let’s think step by step”* (classic chain-of-thought prompting, which is known to improve performance on complex tasks[[12]](https://arxiv.org/html/2405.06682v1#:~:text=Like%20humans%2C%20large%20language%20model,7%20%2C%20%2032%2C%209)) or *“Review what went wrong”* after errors (akin to human self-explanation, which aids learning). By adjusting this, we can measure the **benefit of explanations and reflection** on learning outcomes.
* **Explanation Timing:** In addition to whether explanations occur, we can control **when and how they are provided**. For example, *immediate feedback* mode: after each question, the tutor (or an oracle) reveals the correct answer or explains the solution, which the student model then incorporates into context for future questions. Alternatively, *delayed feedback* mode: let the student attempt several questions, then provide explanations at the end (mimicking an exam review session). This dial helps simulate different instructional strategies. We might find, for instance, that immediate explanations lead to faster error correction but possibly less retention (due to reliance on being told), whereas delayed explanation tests the model’s ability to infer patterns on its own and then adjust. The simulator’s logging (discussed later) would capture how these timing differences affect the learning curve.
* **Tutor Intervention & Hints:** Another possible dial is how proactive the tutor side is. In one extreme, the tutor just gives a batch of examples and then tests the student (passive context). In another, the tutor can interact, e.g., if the student is struggling, the tutor gives a hint or an easier sub-problem. While more complex, this dial moves toward an **interactive ITS (Intelligent Tutoring System)** style, where the sequence of tasks can adapt. For now, this is an extension idea: the platform could include a *policy for hinting*, such as “if the student gets 2 in a row wrong, insert an additional note or simpler example.” Researchers could then see if an adaptive strategy yields better in-context learning than a fixed prompt.

All these dials would be configurable via the simulator’s interface or experiment configuration. Under the hood, they affect how the prompt for the next question is constructed (which context to include, what additional instructions to prepend or append, etc.). By systematically varying these, we create a **rich simulation environment** to test questions like: Does reflecting on errors improve learning of a concept? How many examples are needed before diminishing returns (the learning curve shape)? What if examples are spaced out versus grouped together? We can answer these by running controlled simulations with different dial settings and comparing results.

Importantly, these extensions make the simulator not just a static few-shot prompt tester, but a **dynamic learning environment**. We can simulate multi-turn learning sessions with conditions akin to educational psychology experiments. For instance, the *spacing effect* (known from human studies to improve long-term retention by distributing learning over time) can be modeled by spacing out related examples across a session[[13]](https://pmc.ncbi.nlm.nih.gov/articles/PMC3399982/#:~:text=,schedules%20than%20massed%20learning)[[14]](https://pmc.ncbi.nlm.nih.gov/articles/PMC1876761/#:~:text=,amount%20of%20information%20massed%20together). Error-driven adjustments can be modeled by enabling reflections. The infrastructure for these controls will be built into the orchestrator and context assembly pipeline.

## Task and Domain Abstraction Layers

Supporting all domains and a variety of tasks requires careful abstraction so that the core simulator logic is domain-agnostic. We implement two key abstraction layers: one for **domains** and one for **task types**.

**Domain Abstraction:** Each domain (e.g. high-school math, Python programming, medical diagnosis, public policy case study) will have a domain profile that includes: - A knowledge base or set of **reference materials** (e.g. a cheat sheet of formulas, an API documentation, a medical guideline) that can be provided as “NOTES” or reference text in the context. - A list of **domain-specific terms** (jargon, proper nouns, symbols) that need anonymization. The system will use this to replace these terms with neutral placeholders. For example, in a medicine domain, “diabetes” might become “Condition\_A,” patient names become “Person\_X.” In coding, specific library names could be obfuscated. This ensures that even if a powerful model *has* seen the term “diabetes” thousands of times in training, it cannot reflexively pull a known fact; it must rely on the provided notes about “Condition\_A.” Recent research supports this approach: models with reduced parameter count or pruned weights lose recall of many facts, yet **preserve the ability to process information given in context**[[15]](https://openreview.net/forum?id=ldJXXxPE0L#:~:text=technique%20for%20reducing%20model%20size,Moderate%20pruning%20impairs%20LLM%E2%80%99s)[[16]](https://openreview.net/forum?id=ldJXXxPE0L#:~:text=abilities%20of%20LLMs%20differently,when%20replacing%20the%20original%20model). In other words, even a downsized model that forgets memorized knowledge can perform well if the needed info is fed in the prompt. By anonymizing terms, we intentionally force the model into that regime – it must act as if it “forgot” and only the prompt matters. This helps evaluate pure in-context learning ability. - **Domain adapter functions:** possibly custom logic needed to generate or evaluate content in that domain. For instance, for math, a function to render equations in text; for policy, maybe a function to inject relevant data or context specifics (like current events can be stripped out or generalized). - **Content difficulty calibration:** a way to select or generate tasks of appropriate difficulty for the model’s level, if needed, or to ensure a mix of easy/hard problems. This can help simulate a curriculum within a domain.

The domain abstraction allows the simulator to load a new domain easily (just provide the content and term list, etc.). It also ensures that any *domain-general behaviors* (like the student’s tendency to forget middle context, or benefit from examples) can be observed independent of specific content. We can plug in domains ranging from elementary arithmetic to legal contract review and apply the same sequence of experiments on each.

**Task Abstraction:** As mentioned earlier, tasks are implemented in a modular way. Concretely, we will define a **Task Definition Interface** with components: - **Problem Generator/Loader:** either generates a new problem instance or fetches one from a dataset. This includes applying anonymization and formatting it into a prompt. For example, a coding task might retrieve a buggy code snippet from a dataset and obfuscate function names, then format as: *“Here is a buggy code. Fix it so that all tests pass:\ncode\n...”*. A table reasoning task might take a table of data (possibly randomized) and a question, then linearize the table into text or provide it as a CSV attachment. - **Solution Verifier:** a function that takes the model’s output and produces a correctness judgment. Different tasks plug in different verifiers: - For multiple-choice or short-answer factual questions, the verifier can compare to a gold answer key. If answers are text, we might use fuzzy matching or even an LLM-based grader to allow variation, especially for short answers (SAQs) where phrasing may differ. - For code tasks, as noted, execute the code against test cases (similar to how the HumanEval benchmark evaluates functional correctness[[4]](https://www.datacamp.com/tutorial/humaneval-benchmark-for-evaluating-llm-code-generation-capabilities#:~:text=function%20signature%2C%20docstring%2C%20body%2C%20and,7%20tests%20per%20problem)[[5]](https://www.datacamp.com/tutorial/humaneval-benchmark-for-evaluating-llm-code-generation-capabilities#:~:text=samples%20for%20a%20problem%20passes,the%20unit%20tests)). The simulator should integrate a sandboxed execution environment for this. - For proof or math derivation tasks, the verifier might interface with a CAS (Computer Algebra System) or formal proof checker. For instance, if the task is to supply the next step in a proof, the verifier can check if that step is logically valid given previous steps or use a tool like Lean to confirm the proof can progress[[6]](https://arxiv.org/html/2404.12534v1#:~:text=assistants%20easier%20to%20use%20by,improving%20proof%20automation). - For table reasoning, the verifier might check the answer against the data in the table – e.g. recompute an aggregation or look up the referenced cell to ensure the answer is supported by the table. In tasks like fact-checking a statement against a table, the verifier would confirm the statement is entailed by table data (similar to TabFact benchmark evaluations). - **Prompt Template:** defines how the context, question, and answer exchange is structured. Having a consistent template per task type ensures the model sees a predictable format, which is important for ICL. For example, a short-answer QA template might be:

CONTEXT: <domain notes or passage>  
QUESTION: <the question?>  
ANSWER:

where the model must fill in the answer. A proof task might have a different structure, possibly multi-turn (each step as a Q&A pair). The template also defines how to insert few-shot examples if we are doing one prompt with multiple Q&A demonstrations. The system could utilize one of several prompting methods (e.g. standard few-shot vs. chain-of-thought demonstrations) depending on configuration.

By standardizing tasks this way, the simulator can load a variety of problems and handle them in a unified loop. The tutor/orchestrator will use the task definitions to present problems to the student model and use the verifier to judge responses. This separation of concerns is akin to how OpenAI’s evaluation frameworks or BIG-bench handle multiple tasks with a common driver.

Additionally, this abstraction layer allows **rich task types beyond MCQ** to be easily integrated: - We can introduce a *code repair task* with its own evaluator, without needing to change how the rest of the system logs data or manages context. - We can have *symbolic math proof tasks* where each step is checked; the task interface might allow multi-turn interaction until the proof is complete or the model gives up. - We can handle *table-based QA* tasks by including table data as part of the context (text or image format) and a specialized evaluation routine[[17]](https://aclanthology.org/2024.findings-acl.23.pdf#:~:text=of%20various%20LLMs%20in%20interpreting,the%20role%20of%20representation%20and)[[18]](https://aclanthology.org/2024.findings-acl.23.pdf#:~:text=1%20Introduction%20Recent%20years%20have,their%20effec%02tiveness%20on%20structured%20data). - We can support *free-form short answers (SAQs)* by using semantic scoring – e.g. requiring certain keywords from the reference text appear in the answer or using an LLM to rate the answer against the reference explanation.

Crucially, the domain and task layers work together with anonymization: **closed-book condition is enforced across all tasks** by ensuring any required knowledge is present in the context (domain notes or few-shot examples) and obfuscating any identifiers that the model might otherwise know. For instance, if a task question originally is *“What is the capital of France?”*, the domain adapter could rename “France” to a fictional country name and provide a note like *“CountryX is a country in Europe. Its capital city is ParisX.”* – so that the model must use the note rather than recall “Paris” from memory. By doing so for all tasks, we can truly measure the model’s *in-context learning and reasoning*, not its trivia recall.

This abstraction strategy also aids **benchmarking**. We can create a diverse suite of tasks and domains, all runnable through the same pipeline, to evaluate generalization of ICL. Each task/domain pair can be like a module that can be distributed or shared with others, promoting standardized evaluation (e.g. a researcher can plug in a new domain module for, say, **astronomy**, and test their model in our simulator, under the same protocols as other domains).

## Logging and Analysis Tools

A critical part of the roadmap is implementing thorough logging and analysis capabilities. Since one goal is to measure learning behaviors (learning curves, error rates, etc.), the simulator must record detailed trace data from each session:

* **Interaction Logs:** Every turn of interaction will be logged, including:
* The context given to the model (e.g. which notes, examples, or hints were in the prompt),
* The model’s response (the answer, and possibly its chain-of-thought if we prompt it to produce one),
* The correctness evaluation from the verifier, including whether it passed the witness-gated checks (context coverage, consistency checks – more on this soon),
* Any feedback provided to the model (e.g. if an explanation or correct answer was revealed). This log can be stored in a structured format (JSON or database) for later analysis.
* **Metadata:** We log metadata like domain, task type, difficulty level of the problem, which model (and version) was used as the student, and what settings (dials) were active (e.g. memory policy = “window 5”, reflection = on, etc.). This will allow comparing performance across conditions in analysis.
* **Outcome Metrics:** For each problem (or each turn), we compute metrics such as binary correctness, partial credit (if applicable), and *time to answer* (in tokens or seconds, if relevant). We may also record the number of attempts the model made if the simulator allows retries (for instance, if reflection is on, the model might answer incorrectly then try again after a hint – we’d log attempt1 wrong, attempt2 correct).
* **Learning Curve Data:** As the model progresses through a sequence of tasks, we track its cumulative performance. For example, we might measure the error rate on first attempts over time. In human learning research, a common metric is the error rate on first try vs practice opportunity number, often producing a downward curve as learning happens. We aim to produce similar plots for the model. In fact, prior work with simulated learners has shown that with just a handful of in-context training examples, LLMs can produce **remarkably human-like learning curves** when error rate is plotted over opportunities[[2]](https://arxiv.org/html/2505.01563v1#:~:text=initial%20evaluation%2C%20we%20find%20that,context%20learning)[[19]](https://arxiv.org/html/2505.01563v1#:~:text=succeeded%20in%20closely%20aligning%20with,19%2C%2020%2C%2049). Our system will explicitly capture data to generate such curves – e.g. if a “skill” can be identified for each question, we could plot error rate vs. problem count for that skill, and compare to human data if available. TutorGym demonstrated this by comparing LLM agent learning curves to real student curves, finding qualitatively similar trends with in-context learning (the LLM’s error rate dropped over 3-10 practice problems in a way that aligned with human error reduction)[[20]](https://arxiv.org/html/2505.01563v1#:~:text=match%20at%20L238%20succeeded%20in,19%2C%2020%2C%2049)[[19]](https://arxiv.org/html/2505.01563v1#:~:text=succeeded%20in%20closely%20aligning%20with,19%2C%2020%2C%2049).
* **Analysis Dashboard:** We will create tools to aggregate and visualize the logged data. This might include:
* Automated generation of learning curve plots (error rate or score vs. problem index). For example, the system can output a graph of how accuracy improves from the first example to the tenth example in context.
* Comparison charts for different conditions – e.g. a side-by-side plot of *with explanations* vs *without explanations*, or *full memory* vs *limited memory*, etc. This will highlight the impact of those dials.
* Histograms or breakdowns of error types: if the verifier provides reasoning (like which test cases failed, or which part of an answer was missing evidence), we can log that and then analyze common failure modes.
* In the code domain, we might log *how many attempts until a solution passed all tests*, giving a distribution of trial counts.
* For free-response answers, if we use an LLM grader with a rubric, we can log the scores or feedback given and analyze that across the dataset.
* **Session Replays:** Because the simulator may be used in research papers or to demonstrate behavior, having an easily readable log or *replay script* is useful. We can output a markdown or HTML log of a session showing each Q, the model’s answer, and whether it was correct (perhaps even highlighting where in the context the evidence for the answer came from). This can qualitatively illustrate how the model is learning (or failing to). Such replays also help debugging the prompt design or identifying if the model is exploiting some loophole (e.g. maybe the model learned it can answer “I don’t know” and the evaluator gives partial credit incorrectly – a log would reveal that).
* **Error Analysis Tools:** We should equip the analysis suite with the ability to filter and examine cases where the model answered incorrectly despite having the info in context, versus cases it answered correctly but without using context, etc. This can be done by cross-referencing the witness signals (below). If we log which context sentence was supposed to contain the answer evidence and whether the model actually mentioned it, we can identify when the model is hallucinating or guessing. We can then quantify *“percentage of correct answers that were fully supported by notes”* or *“instances of hallucination”*.

From an engineering perspective, we might implement this logging via a combination of a database (for structured analysis) and on-the-fly logging to files for immediate feedback. The analysis tools could be Python scripts or a Jupyter notebook that queries the database to produce the charts. Since the user will likely run many experiments (various domains and settings), we should emphasize reproducibility: each run should be timestamped and have a unique ID so results can be compared, and possibly a mechanism to aggregate multiple runs for statistical significance (e.g. run 5 simulations with different random seeds and then average).

Finally, the logged data will support the **benchmarking protocols** described later. For example, if we define a benchmark as the average score after 5 examples on a set of tasks, the logger will have captured those scores. Or if the protocol is to measure *area under the learning curve* as a single number, we can compute that from the logs.

## Evaluation and Witness-Gated Scoring

One of the novel aspects of this ICL simulator is the **witness-gated scoring** system. The idea is that an answer shouldn’t be counted as fully “correct” by the simulator unless it meets several criteria (witness conditions) beyond just matching the expected answer. This ensures the model isn’t succeeding by luck or by unwittingly using external knowledge, and it rewards answers that demonstrate learning from context. The **witness-gating** will typically involve three components:

1. **Correctness of the Answer (Basic Accuracy):** The answer must be factually or functionally correct with respect to the problem. This is the first gate: if the answer is outright wrong (fails the tests, or doesn’t match the key), it’s incorrect. We implement this via the task-specific evaluators (answer key match, test case results, proof checker, etc.). Only if this is passed do we proceed to the next checks.
2. **Evidence from Context (Context Coverage):** The answer must be *well-supported by the provided context (notes or examples)*. In other words, the model should be using the information given, not pulling in outside facts or hallucinations. To enforce this, the evaluation module will check that for each key piece of the answer, there is a “witness” in the context. For example, if the question was “What is the capital of CountryX?” and the notes said “CountryX’s capital is ParisX,” then the model answer “ParisX” is only considered valid if the notes indeed had that fact. If the model answered correctly *but* the notes did not actually cover that (meaning the model might have recalled the pattern “X’s capital is probably Y” from pretraining or just guessed), we would mark it as failing the witness check. Essentially, the simulator acts like an open-book exam proctor: the student must show their work or sources.

How to implement this? A simple method is to require the model to output not just an answer but also a reference to the context or an explanation using the context. For instance, we could prompt the model: *“Answer the question and cite which note you used.”* If the citation or content in the answer corresponds to actual provided text, then we know it used the context. Another approach is automated: after the model answers, the system itself can search the provided context for sentences or keywords related to the answer. If the answer contains unique tokens or names that never appeared in the context, that’s a red flag (it might have hallucinated something not given). We can use string overlap or embedding-based similarity to find if the answer’s content is derivable from the notes. In a more advanced approach, we could employ an LLM verifier that reads the context and the answer and outputs a judgement like “Does the answer correctly use the provided information?” However, that introduces a second model in the loop. In any case, the scoring will incorporate this: **an answer gets full credit only if supported by context evidence**.

This addresses the problem of hallucinations and makes the evaluation stricter. As noted, LLMs often produce answers that sound plausible but are not grounded in any source (hallucinations)[[21]](https://arxiv.org/html/2505.01563v1#:~:text=for%20generating%20on,world). By requiring evidence, we guard against accepting those. It’s similar to how some QA benchmarks require a system to point to a source text that contains the answer. It also differentiates between genuine in-context learning vs. cheating: if we anonymize terms correctly, a model passing this check is likely actually using the context to derive the answer.

1. **Consistency Check (Re-pick / Self-Consistency):** This is an additional gate to ensure the model’s reasoning is sound and the answer is not a fluke. There are a few ways to implement a consistency check:
2. **Self-Consistency (Re-sampling):** We can pose the same question or prompt multiple times (with some variations or allowing the model to generate multiple reasoning paths) and see if the answers converge[[22]](https://www.promptingguide.ai/techniques/consistency#:~:text=Self,select%20the%20most%20consistent%20answer). The Self-Consistency method (Wang et al., 2022) suggests sampling diverse chains-of-thought and then taking the most common answer as the final[[23]](https://www.promptingguide.ai/techniques/consistency#:~:text=The%20idea%20is%20to%20sample,select%20the%20most%20consistent%20answer). In our simulator, we could do something similar behind the scenes: after the model gives an answer, we could prompt it again (or prompt a few replicas if we have the resources) to see if it reliably gives the same answer. If not, that indicates uncertainty – the answer might not be firmly learned. For efficiency, we might not do this for every question, but it could be an option for certain evaluation phases. If enabled, the rule could be: an answer “passes” the consistency gate if the model, when asked in a slightly different way (or when thinking step-by-step), still arrives at that answer.
3. **Reasoning Verification:** Another form of consistency is to ask the model to explain its answer and then verify the explanation. For instance, after an answer, the tutor might ask *“Why?”* or *“How did you get that?”*. If the model can produce a coherent explanation that references the context appropriately and leads to the answer, then it’s a good sign of understanding. If it cannot, maybe it guessed. We could score the explanation for logical consistency (possibly via another automated check or heuristic). Essentially, *the answer and explanation should not contradict each other and should both align with the given information*. This is like requiring the student to show their work. Some studies in self-reflection have found that models can identify and correct their own errors when asked to critique their chain-of-thought[[24]](https://arxiv.org/html/2405.06682v1#:~:text=Also%20similar%20to%20humans%2C%20LLM,37%2C%2014%20%2C%20%2039)[[25]](https://arxiv.org/html/2405.06682v1#:~:text=CoT,37%2C%2014%20%2C%20%2039). We can leverage that: if the model’s reasoning process (as a witness) doesn’t lead to the answer it gave, then the answer might not get full credit.
4. **Cross-Verification via an Alternate Prompt:** We might also use an alternate prompt or an automated “checker agent.” For example, we could feed the question, context, and the model’s answer into a separate verification model prompt like: *“Given the above context, is the answer correct and supported? Answer yes or no.”* This would double-check with another pass (though again, using an LLM to verify an LLM has its own reliability issues). Alternatively, if multiple models are available (e.g., GPT-4 as student, GPT-4 or another as verifier), this could be a form of ensemble consistency.

In the system design, the **Evaluation Module** will incorporate these checks after the model produces an answer. The result could be a tuple like (correctness\_passed, evidence\_passed, consistency\_passed) flags. Only if all are true do we record it as a fully correct outcome. We might still log partial success (e.g. answer was right but not from context) as that’s useful information.

Witness-gated scoring encourages behaviors like citing the provided notes, and it penalizes relying on parametric knowledge. It effectively operationalizes the “closed-book” mandate: even a correct answer can be disqualified if it wasn’t *from the book (notes)*. In practice, if a model tries to answer something that requires a certain fact not in the context (because we anonymized it out), it should fail evidence check – which is what we want.

An example of witness gating: suppose the model is given a mini history text about “CountryX” and it has to answer who founded it. If the model answers correctly with a name *that was not mentioned in the text*, then it clearly used outside knowledge. The simulator’s evidence check catches that and marks it as unsupported. The tutor could even respond with *“Your answer is not supported by the notes. Please refer to the notes for the correct information.”* – pushing the model to actually use the context. Only when the model answers with the name given in the notes (say the notes said “Explorer Y established the colony of CountryX in 1700s”) will it get full credit.

The witness mechanism can also be used to implement *adaptive prompting*: if a model fails the evidence check, the system might follow up by highlighting the relevant part of the notes as a hint and asking the question again. This way, the simulator not only evaluates but also guides the model toward context-grounded answers.

In summary, the **witness-gated scoring** is a multi-layer filter that enforces correct, context-based, and consistent answers. It aligns with best practices observed in recent research: - It addresses hallucinations by requiring evidence[[21]](https://arxiv.org/html/2505.01563v1#:~:text=for%20generating%20on,world). - It relates to the *inverse frequency effect* from cognitive science: error-driven learning implies learning more from surprising examples[[26]](https://aclanthology.org/2025.naacl-long.586/#:~:text=this%20paper%2C%20we%20introduce%20a,display%20the%20IFE%2C%20with%20the)[[27]](https://aclanthology.org/2025.naacl-long.586/#:~:text=effect%20being%20stronger%20in%20larger,line%20processing). Our consistency check can be seen as ensuring the model *updates its approach* after an error. If a model gets something wrong (especially a rare pattern), and then we provide feedback, we expect it to adjust – the error-driven learning hypothesis for ICL suggests models do implicitly compute an error signal[[28]](https://aclanthology.org/2025.naacl-long.586/#:~:text=effect%20,at%20least%20in%20the%20case)[[27]](https://aclanthology.org/2025.naacl-long.586/#:~:text=effect%20being%20stronger%20in%20larger,line%20processing). Our simulator could incorporate this by perhaps weighting errors on rare contexts more, or simply by observing through the logs if performance jumps after encountering a novel example (which would mimic the inverse frequency effect where a rare exposure has a big impact). - It also connects to the idea of *calibrated confidence*: if a model is not consistent, it might be indicating low confidence. By requiring consistency, we indirectly prefer answers the model “strongly knows” from context.

Implementing this robust scoring will give us a more nuanced picture of ICL capabilities. Instead of just accuracy, we can measure **context-utilization accuracy** and **consistency rate**, etc., for each model and condition.

## Experiments and Benchmarking Protocols

With the platform built as above, we will design a series of experiments and a benchmarking suite to evaluate in-context learning across different domains, tasks, and model types, under the controlled closed-book setting. Below are the key experiments and protocols we propose:

### 1. **Learning Curve Measurement (Few-Shot Scaling)**

**Objective:** Quantitatively measure how performance improves as the number of context examples increases (the hallmark of in-context learning).  
**Protocol:** For each domain-task combination, run the simulator with 0-shot (no examples, only the question), 1-shot, 2-shot, … up to *k*-shot examples in the prompt. Ensure the examples are drawn from the same distribution and anonymized. Record accuracy or error rate as a function of number of shots. Plot learning curves for each model.  
**What to Measure:** The shape of the curve (diminishing returns or steady improvement), the “effective few-shot capacity” (how many examples until performance plateaus), and differences between models. We expect larger models to have steeper improvement with more examples, as seen in literature (e.g., GPT-3 showed significant gains from 0 to 32-shot in many tasks[[29]](https://proceedings.neurips.cc/paper_files/paper/2023/file/cda04d7ea67ea1376bf8c6962d8541e0-Paper-Conference.pdf#:~:text=To%20test%20whether%20the%20effects,curve%20for%20all%20these%20models)). We can also fit simple learning curve models or scaling laws to this data[[30]](https://proceedings.neurips.cc/paper_files/paper/2023/file/cda04d7ea67ea1376bf8c6962d8541e0-Paper-Conference.pdf#:~:text=%5BPDF%5D%20Meta,curve%20for%20all%20these%20models). A key metric could be *shots to reach X% accuracy* or *area under the learning curve*. If human data is available for a domain (like how quickly humans learn a skill), we can compare the model’s curve to it, similar to TutorGym’s approach[[31]](https://arxiv.org/html/2505.01563v1#:~:text=match%20at%20L245%20how%20this,or%20incorrect%2C%20and%20these%20experiences)[[19]](https://arxiv.org/html/2505.01563v1#:~:text=succeeded%20in%20closely%20aligning%20with,19%2C%2020%2C%2049).

### 2. **Explanation and Reflection Benefit**

**Objective:** Evaluate the impact of explanations and self-reflection on learning outcomes.  
**Protocol:** Take a set of tasks (e.g., math word problems). Run two conditions: (A) *No explanations*: the model is prompted to give final answers only, and if wrong, it’s just marked wrong (no feedback beyond correct/incorrect). (B) *Explanation & reflection*: the model is prompted to “think aloud” (chain-of-thought) and after answering, if it’s wrong, the simulator provides the correct solution explanation or asks the model to reflect and try again. Both conditions eventually expose the model to the correct reasoning (either implicitly through its own CoT in B, or not at all in A beyond seeing it got it wrong). Then compare how performance on subsequent similar problems differs. For example, maybe in condition B the model learns to avoid a certain trap because it reflected on it.  
**What to Measure:** Final accuracy after a learning session of N problems under each condition; speed of error reduction; types of mistakes made. We anticipate that enabling reflection will yield higher eventual accuracy – e.g., Renze *et al.* (2024) found that self-reflecting LLM agents significantly improved their problem-solving after reflecting on mistakes[[11]](https://arxiv.org/html/2405.06682v1#:~:text=models%20%28LLMs%29%20on%20problem,determine%20their%20individual%20contribution%20to). We can also measure the quality of explanations themselves (do they improve over time, indicating internal learning?). This experiment will inform how much *explanation helps in-context learning*.

### 3. **Spacing and Position Effects**

**Objective:** Test how the ordering and spacing of examples affect the model’s retention and application of information.  
**Protocol A (Primacy/Recency):** For a given set of few-shot examples plus a target question, create multiple prompt orderings. E.g., one where the most relevant example is at the top vs at the bottom vs in the middle of the context. Evaluate the model’s performance in each case. This directly probes position bias. We expect, based on “Lost in the Middle” findings, that placing relevant info at the start or end yields better performance than burying it[[8]](https://cs.stanford.edu/~nfliu/papers/lost-in-the-middle.arxiv2023.pdf#:~:text=performance%20curve%E2%80%94models%20are%20better%20at,2023%3B%20Schick).  
**Protocol B (Spaced vs Massed):** If the simulator supports multi-turn sessions, we can intermix different topics to simulate spacing. For example, teach concept A with one example, then switch to concept B, then come back to a second example of A later. In a massed condition, do two examples of A back-to-back then B. At the end, test the model on both A and B. Compare accuracy – does the spaced schedule improve retention on A? This mirrors human spaced repetition experiments (which consistently show spaced learning outperforms massed practice in long-term retention[[13]](https://pmc.ncbi.nlm.nih.gov/articles/PMC3399982/#:~:text=,schedules%20than%20massed%20learning)[[14]](https://pmc.ncbi.nlm.nih.gov/articles/PMC1876761/#:~:text=,amount%20of%20information%20massed%20together)). Since LLMs don’t have long-term memory in the same sense, our hypothesis might be that within a single session, spacing might *hurt* (because the model may forget the earlier example by the time we return to concept A, given context length limits). However, if our memory policy kept the older example in context (maybe summarized), we might simulate the reminder effect. This is an open question that the simulator can help answer: how do LLMs handle interleaved context vs block context?  
**What to Measure:** Accuracy on test questions for each ordering/spacing condition. Also measure model’s confidence (if available) or consistency. For position, we might find a drop in accuracy for middle-placed info, quantifying how severe the “middle context loss” is (e.g. model accuracy might drop X% when the relevant example is not recent). For spacing, measure differences in error rates between spaced and massed practice conditions.

### 4. **Error-Driven Learning and Inverse Frequency Effect**

**Objective:** Investigate whether the model’s in-context learning exhibits **error-driven characteristics** – i.e., does it learn more from mistakes and rare experiences, analogous to how humans show the inverse frequency effect (rare patterns can produce larger adjustments)[[26]](https://aclanthology.org/2025.naacl-long.586/#:~:text=this%20paper%2C%20we%20introduce%20a,display%20the%20IFE%2C%20with%20the)[[27]](https://aclanthology.org/2025.naacl-long.586/#:~:text=effect%20being%20stronger%20in%20larger,line%20processing).  
**Protocol:** Design a structural priming or pattern learning task. For instance, create a simple grammatical transformation task where the model sees example input-output pairs. Include either a high-frequency pattern (shown many times) or a low-frequency/rare pattern (shown once) and then test on that pattern. If ICL is like error-driven gradient descent, a rare example might influence the model’s output disproportionately (Zhou et al. 2025 observed LLMs show a stronger priming effect from less frequent examples, akin to humans[[28]](https://aclanthology.org/2025.naacl-long.586/#:~:text=effect%20,at%20least%20in%20the%20case)[[32]](https://aclanthology.org/2025.naacl-long.586/#:~:text=involve%20error,line%20processing)). Concretely, we can do: show the model 4 sentences converting active voice to passive and 1 sentence converting something in a slightly different way (like a rare structure). Then ask it to convert a new sentence of each type. Does it do better on the rare one than expected given frequency? Another approach: deliberately cause the model to make an error on a certain type of question, then immediately give an example or explanation for that same type, then test a similar question – see if it corrects the mistake (learning from the error). Measure the *error correction rate*.  
**What to Measure:** If testing inverse frequency, measure accuracy on infrequent pattern vs frequent pattern relative to baseline. If testing error correction, measure the probability the model gets a question right *after* just having gotten a similar one wrong and seeing the solution. A high correction rate would indicate it integrated the feedback (which would support the notion of an implicit error-driven update happening in-context). This experiment connects to theoretical discussions of whether ICL mimics gradient descent updates internally[[26]](https://aclanthology.org/2025.naacl-long.586/#:~:text=this%20paper%2C%20we%20introduce%20a,display%20the%20IFE%2C%20with%20the). Our simulator can supply empirical evidence for this across tasks.

### 5. **Alias/Notation Transfer**

**Objective:** Test the model’s ability to transfer learned concepts to new aliases or notations, ensuring it’s learning the underlying concept, not just the surface token.  
**Protocol:** During training examples, refer to a concept with one term, and in the test query, use a different term (that was defined as equivalent). For instance, in a policy domain, all examples might refer to “Country Z” and in the final question we instead say “the nation” – does the model realize it’s the same entity from context? In math, show a formula with certain symbol names, then ask using different symbol names (units, variables). Or use a cipher: provide a note like “In this context, ‘florp’ means ‘apple’.” Then ask a question about “florp” – see if the model applies the note. Essentially, evaluate *symbol binding* in context. The anonymization itself is a form of aliasing; we can push it further by changing aliases mid-stream to see if the model gets confused or can adapt.  
**What to Measure:** Accuracy on queries that require understanding alias mapping. If a model truly learned from context, it should handle synonyms or re-phrasings introduced in the context. A failure would mean it over-fitted to the exact wording of examples. For example, if all notes say “dog = a type of animal that barks” and the question asks “Which creature is known to bark?”, do we get “dog”? Good performance here indicates flexible application of context knowledge, similar to how humans can transfer learning to synonyms.

### 6. **Cross-Domain and Model Benchmarking**

Finally, we will set up a **benchmark suite** where multiple models are evaluated on a battery of tasks under strict closed-book conditions. This serves as a comparison of ICL proficiency: - **Domains:** We select a diverse set (e.g. elementary math, code debugging, medical Q&A, policy analysis, common-sense reasoning, etc.), each with their anonymized context and tasks. - **Tasks:** Within each domain, a variety of tasks (MCQ, short answer, etc. as appropriate) are included. - **Models:** Include a range of models: smaller ones like the hypothetical “GPT-5-nano” (a lightweight GPT-5), mid-size like Llama-3, and large ones like GPT-4 or beyond. If available, include a “reasoning agent” that might use tools or chain-of-thought explicitly (to see if tool use helps ICL). - **Procedure:** Each model is evaluated on the same tasks with the same supporting context. We fix a protocol (say 5-shot examples provided for each task, reflection off to keep it consistent, or perhaps whatever default yields best performance for that model). All domain terms are anonymized so no model has an inherent advantage of prior knowledge. We then measure overall accuracy as well as the other metrics (e.g. average witness score, consistency, etc.).

**What to Measure (Benchmark metrics):**  
- *ICL Score:* Perhaps the primary metric is average accuracy under these closed-book conditions, which could be contrasted with an open-book or non-anonymized version to quantify how much the model relied on recall. We expect big models to drop in performance when forced into closed-book mode, but the degree to which they can recover via in-context info is the key measure. For instance, if GPT-4 achieves 80% on some task open-book but only 50% when all clues are obfuscated, that gap is interesting. A smaller model might have had 50% open-book and 40% closed-book, showing it had less memorized anyway and is closer to pure context performance. Prior research suggests that reducing model size or pruning weights disproportionately hurts memorized recall but leaves in-context learning relatively intact[[15]](https://openreview.net/forum?id=ldJXXxPE0L#:~:text=technique%20for%20reducing%20model%20size,Moderate%20pruning%20impairs%20LLM%E2%80%99s)[[16]](https://openreview.net/forum?id=ldJXXxPE0L#:~:text=abilities%20of%20LLMs%20differently,when%20replacing%20the%20original%20model) – in line with that, smaller models might narrow the gap between open and closed book (because they didn’t know much to begin with)[[33]](https://openreview.net/forum?id=ldJXXxPE0L#:~:text=TL%3BDR%3A%20Moderate%20down,scaling)[[16]](https://openreview.net/forum?id=ldJXXxPE0L#:~:text=abilities%20of%20LLMs%20differently,when%20replacing%20the%20original%20model). - *Learning Efficiency:* Compare learning curves across models. Perhaps define a metric like *ICL efficiency = (accuracy at 5-shot) – (accuracy at 0-shot)* to capture how much improvement the model gained from the provided examples. Larger or more meta-trained models might have higher ICL efficiency. We can tabulate this per domain. - *Domain Generality:* Does a model that excels in say programming ICL also excel in medical ICL, or are there domain-specific quirks? The benchmark results could show per-domain scores. This might reveal, for example, that a model struggles in domains requiring precise calculation (math) but does fine in those requiring factual recall if provided (history). Those differences can guide where to improve domain adapters or what kind of additional pretraining might help. - *Effect of Reasoning Agent:* If we include an agent that can, say, do tool use (e.g., run a calculator or search within the provided notes) vs a plain LLM, we can see if that yields better accuracy or consistency. It’s conceivable that a reasoning agent could better follow multi-hop context (like scanning a long reference) than a vanilla model. We’d measure that advantage. - *Closed-book vs Open-book:* As a sanity check, we might run the tasks in an open-book way (no anonymization, so a model could rely on its memory) to see the difference. The difference highlights how much the model was originally just recalling answers. For instance, TutorGym cited an example where adding distractors and renaming in GSM8K dropped accuracy dramatically[[3]](https://arxiv.org/html/2505.01563v1#:~:text=High%20accuracies%20on%20these%20benchmarks,solving), implying original performance was bolstered by memorization. Our benchmark formalizes this by measuring both modes. A truly good ICL simulator (or a model tuned for ICL) would have a smaller gap between open and closed book performance, because it always leans on provided info.

**Benchmark Output:** We will produce a report (and possibly a leaderboard) that compares models on this suite. Think of it like a *Big-Bench for ICL*, except focusing on closed-book prompt learning. It will provide insights such as “Model X achieves 70% witness-verified accuracy on our multi-domain ICL test, with particularly strong performance in programming tasks, whereas Model Y achieves 60% but has more consistent reasoning chains,” etc.

Additionally, we propose specific **experiments within the benchmark**: - **Scaling Trends:** Use versions of a model at different scales (e.g., Llama2-7B vs 70B, or GPT-3.5 vs GPT-4 vs a hypothetical GPT-5) on the same tasks to see how ICL ability scales. Prior works have noted emergent abilities at scale – we can check if witness-gated ICL is one of them (does a small model ever learn from context in a robust way, or does this only kick in beyond a size threshold?). - **Meta-ICL Training Effects:** If any model has been fine-tuned or meta-trained explicitly for in-context learning (like MetaICL in literature), compare it to base models. This could show whether specialized training leads to better performance under our rigorous evaluation (and if so, on which aspects – maybe it improves consistency or data efficiency).

All experiments above rely on the robust logging and consistent framework we described. The results, with appropriate statistical analysis, will help answer key research questions the user likely has: *Can a single simulator handle all these diverse tasks? How well do today’s models actually learn in context when prevented from cheating? What strategies (explanations, spacing, etc.) yield the best in-context learning gains? And how to benchmark progress in ICL objectively across time?*

By following this roadmap, we will have transformed the tutor platform into a **powerful ICL simulation and research tool**. It will not only tutor AI agents in real-time across domains but also generate insights and data for understanding and improving in-context learning in AI. The modular architecture ensures it can evolve with new models (the day we have GPT-6 or a new reasoning algorithm, we can plug it in), and the evaluation protocols mean we can track improvements in a quantifiable way. Ultimately, this paves the way toward AI systems that can *truly learn new tasks on the fly* from whatever information is at hand – a key step toward more general and adaptable intelligence.

**Sources:**

* Brown et al., *Language Models are Few-Shot Learners*, NeurIPS 2020 – demonstrated prompt-based learning curves.
* Jin et al., *The Cost of Scaling Down LLMs: Memory vs In-Context Learning*, ICLR 2024 – found pruning harms recall more than in-context learning[[15]](https://openreview.net/forum?id=ldJXXxPE0L#:~:text=technique%20for%20reducing%20model%20size,Moderate%20pruning%20impairs%20LLM%E2%80%99s)[[16]](https://openreview.net/forum?id=ldJXXxPE0L#:~:text=abilities%20of%20LLMs%20differently,when%20replacing%20the%20original%20model).
* Weitekamp et al., *TutorGym: A Testbed for AI Tutors and Students*, 2025 – showed LLMs as students produce human-like learning curves and noted drops in LLM performance when problems are altered to prevent memorization[[3]](https://arxiv.org/html/2505.01563v1#:~:text=High%20accuracies%20on%20these%20benchmarks,solving)[[2]](https://arxiv.org/html/2505.01563v1#:~:text=initial%20evaluation%2C%20we%20find%20that,context%20learning).
* Renze & Guven, *Self-Reflection in LLM Agents Improves Performance*, 2024 – LLMs improved accuracy by reflecting on wrong answers[[11]](https://arxiv.org/html/2405.06682v1#:~:text=models%20%28LLMs%29%20on%20problem,determine%20their%20individual%20contribution%20to) and CoT prompting boosted reasoning[[12]](https://arxiv.org/html/2405.06682v1#:~:text=Like%20humans%2C%20large%20language%20model,7%20%2C%20%2032%2C%209).
* Liu et al., *Lost in the Middle: How LMs Use Long Contexts*, 2023 – identified primacy/recency biases with U-shaped performance for info retrieval from long prompts[[7]](https://cs.stanford.edu/~nfliu/papers/lost-in-the-middle.arxiv2023.pdf#:~:text=long%20input%20contexts,models%20use%20their%20input%20context)[[8]](https://cs.stanford.edu/~nfliu/papers/lost-in-the-middle.arxiv2023.pdf#:~:text=performance%20curve%E2%80%94models%20are%20better%20at,2023%3B%20Schick).
* Zhou et al., *ICL as Error-Driven Learning: Evidence from Inverse Frequency Effect*, NAACL 2025 – found LLMs learn more from rare examples, mirroring human structural priming[[26]](https://aclanthology.org/2025.naacl-long.586/#:~:text=this%20paper%2C%20we%20introduce%20a,display%20the%20IFE%2C%20with%20the)[[27]](https://aclanthology.org/2025.naacl-long.586/#:~:text=effect%20being%20stronger%20in%20larger,line%20processing).
* OpenAI, *HumanEval benchmark* – uses unit tests to evaluate code generation (functional correctness over surface similarity)[[4]](https://www.datacamp.com/tutorial/humaneval-benchmark-for-evaluating-llm-code-generation-capabilities#:~:text=function%20signature%2C%20docstring%2C%20body%2C%20and,7%20tests%20per%20problem)[[5]](https://www.datacamp.com/tutorial/humaneval-benchmark-for-evaluating-llm-code-generation-capabilities#:~:text=samples%20for%20a%20problem%20passes,the%20unit%20tests).
* Song et al., *LLMs as Theorem Proving Copilots in Lean*, 2024 – integrated LLMs with formal proof assistants, evaluating success by the number of proofs completed[[6]](https://arxiv.org/html/2404.12534v1#:~:text=assistants%20easier%20to%20use%20by,improving%20proof%20automation).
* Deng et al., *Evaluating Table Reasoning of LLMs*, ACL 2024 – studied LLM performance on table QA with different representations[[17]](https://aclanthology.org/2024.findings-acl.23.pdf#:~:text=of%20various%20LLMs%20in%20interpreting,the%20role%20of%20representation%20and)[[18]](https://aclanthology.org/2024.findings-acl.23.pdf#:~:text=1%20Introduction%20Recent%20years%20have,their%20effec%02tiveness%20on%20structured%20data).
* Additional cognitive science references on spaced repetition and learning efficacy[[13]](https://pmc.ncbi.nlm.nih.gov/articles/PMC3399982/#:~:text=,schedules%20than%20massed%20learning)[[14]](https://pmc.ncbi.nlm.nih.gov/articles/PMC1876761/#:~:text=,amount%20of%20information%20massed%20together).

[[1]](https://arxiv.org/html/2505.01563v1#:~:text=,agents%20directly%20learn%20from%20ITS) [[2]](https://arxiv.org/html/2505.01563v1#:~:text=initial%20evaluation%2C%20we%20find%20that,context%20learning) [[3]](https://arxiv.org/html/2505.01563v1#:~:text=High%20accuracies%20on%20these%20benchmarks,solving) [[10]](https://arxiv.org/html/2505.01563v1#:~:text=TutorGym%20provides%20exciting%20new%20opportunities,tutors%20and%20as%20simulated%20learners) [[19]](https://arxiv.org/html/2505.01563v1#:~:text=succeeded%20in%20closely%20aligning%20with,19%2C%2020%2C%2049) [[20]](https://arxiv.org/html/2505.01563v1#:~:text=match%20at%20L238%20succeeded%20in,19%2C%2020%2C%2049) [[21]](https://arxiv.org/html/2505.01563v1#:~:text=for%20generating%20on,world) [[31]](https://arxiv.org/html/2505.01563v1#:~:text=match%20at%20L245%20how%20this,or%20incorrect%2C%20and%20these%20experiences) TutorGym: A Testbed for Evaluating AI Agents as Tutors and Students

<https://arxiv.org/html/2505.01563v1>

[[4]](https://www.datacamp.com/tutorial/humaneval-benchmark-for-evaluating-llm-code-generation-capabilities#:~:text=function%20signature%2C%20docstring%2C%20body%2C%20and,7%20tests%20per%20problem) [[5]](https://www.datacamp.com/tutorial/humaneval-benchmark-for-evaluating-llm-code-generation-capabilities#:~:text=samples%20for%20a%20problem%20passes,the%20unit%20tests) HumanEval: A Benchmark for Evaluating LLM Code Generation Capabilities | DataCamp

<https://www.datacamp.com/tutorial/humaneval-benchmark-for-evaluating-llm-code-generation-capabilities>

[[6]](https://arxiv.org/html/2404.12534v1#:~:text=assistants%20easier%20to%20use%20by,improving%20proof%20automation) Towards Large Language Models as Copilots for Theorem Proving in Lean

<https://arxiv.org/html/2404.12534v1>

[[7]](https://cs.stanford.edu/~nfliu/papers/lost-in-the-middle.arxiv2023.pdf#:~:text=long%20input%20contexts,models%20use%20their%20input%20context) [[8]](https://cs.stanford.edu/~nfliu/papers/lost-in-the-middle.arxiv2023.pdf#:~:text=performance%20curve%E2%80%94models%20are%20better%20at,2023%3B%20Schick) cs.stanford.edu

<https://cs.stanford.edu/~nfliu/papers/lost-in-the-middle.arxiv2023.pdf>

[[9]](https://arxiv.org/html/2405.06682v1#:~:text=To%20improve%20their%20performance%2C%20we,42%2C%2010%20%2C%20%2043) [[11]](https://arxiv.org/html/2405.06682v1#:~:text=models%20%28LLMs%29%20on%20problem,determine%20their%20individual%20contribution%20to) [[12]](https://arxiv.org/html/2405.06682v1#:~:text=Like%20humans%2C%20large%20language%20model,7%20%2C%20%2032%2C%209) [[24]](https://arxiv.org/html/2405.06682v1#:~:text=Also%20similar%20to%20humans%2C%20LLM,37%2C%2014%20%2C%20%2039) [[25]](https://arxiv.org/html/2405.06682v1#:~:text=CoT,37%2C%2014%20%2C%20%2039) Self-Reflection in LLM Agents: Effects on Problem-Solving Performance

<https://arxiv.org/html/2405.06682v1>

[[13]](https://pmc.ncbi.nlm.nih.gov/articles/PMC3399982/#:~:text=,schedules%20than%20massed%20learning) Distributing Learning Over Time: The Spacing Effect in Children's ...

<https://pmc.ncbi.nlm.nih.gov/articles/PMC3399982/>

[[14]](https://pmc.ncbi.nlm.nih.gov/articles/PMC1876761/#:~:text=,amount%20of%20information%20massed%20together) Neurogenesis and the spacing effect: Learning over time enhances ...

<https://pmc.ncbi.nlm.nih.gov/articles/PMC1876761/>

[[15]](https://openreview.net/forum?id=ldJXXxPE0L#:~:text=technique%20for%20reducing%20model%20size,Moderate%20pruning%20impairs%20LLM%E2%80%99s) [[16]](https://openreview.net/forum?id=ldJXXxPE0L#:~:text=abilities%20of%20LLMs%20differently,when%20replacing%20the%20original%20model) [[33]](https://openreview.net/forum?id=ldJXXxPE0L#:~:text=TL%3BDR%3A%20Moderate%20down,scaling) The Cost of Scaling Down Large Language Models: Reducing Model Size Affects Memory before In-context Learning | OpenReview

<https://openreview.net/forum?id=ldJXXxPE0L>

[[17]](https://aclanthology.org/2024.findings-acl.23.pdf#:~:text=of%20various%20LLMs%20in%20interpreting,the%20role%20of%20representation%20and) [[18]](https://aclanthology.org/2024.findings-acl.23.pdf#:~:text=1%20Introduction%20Recent%20years%20have,their%20effec%02tiveness%20on%20structured%20data) aclanthology.org

<https://aclanthology.org/2024.findings-acl.23.pdf>

[[22]](https://www.promptingguide.ai/techniques/consistency#:~:text=Self,select%20the%20most%20consistent%20answer) [[23]](https://www.promptingguide.ai/techniques/consistency#:~:text=The%20idea%20is%20to%20sample,select%20the%20most%20consistent%20answer) Self-Consistency | Prompt Engineering Guide

<https://www.promptingguide.ai/techniques/consistency>

[[26]](https://aclanthology.org/2025.naacl-long.586/#:~:text=this%20paper%2C%20we%20introduce%20a,display%20the%20IFE%2C%20with%20the) [[27]](https://aclanthology.org/2025.naacl-long.586/#:~:text=effect%20being%20stronger%20in%20larger,line%20processing) [[28]](https://aclanthology.org/2025.naacl-long.586/#:~:text=effect%20,at%20least%20in%20the%20case) [[32]](https://aclanthology.org/2025.naacl-long.586/#:~:text=involve%20error,line%20processing) Is In-Context Learning a Type of Error-Driven Learning? Evidence from the Inverse Frequency Effect in Structural Priming - ACL Anthology

<https://aclanthology.org/2025.naacl-long.586/>

[[29]](https://proceedings.neurips.cc/paper_files/paper/2023/file/cda04d7ea67ea1376bf8c6962d8541e0-Paper-Conference.pdf#:~:text=To%20test%20whether%20the%20effects,curve%20for%20all%20these%20models) [[30]](https://proceedings.neurips.cc/paper_files/paper/2023/file/cda04d7ea67ea1376bf8c6962d8541e0-Paper-Conference.pdf#:~:text=%5BPDF%5D%20Meta,curve%20for%20all%20these%20models) [PDF] Meta-in-context learning in large language models - NIPS

<https://proceedings.neurips.cc/paper_files/paper/2023/file/cda04d7ea67ea1376bf8c6962d8541e0-Paper-Conference.pdf>