Filling Gaps Instead of Answering: An End-to-End Multi-Agent Tutoring Pipeline over Knowledge Topic-Skill Graphs

Anonymous ACL submission

Abstract

Large language model tutors typically answer student questions directly, short-circuiting learning. We instead build a tutor that fills the gaps blocking the student from answering. From the lecture corpus we construct a two-layer knowledge graph pairing typed knowledge topics with Bloom-leveled skills; an LLM determines, per target skill, the Bloom level at which each prerequisite topic is actually required, sharper than the blanket all-Bloom lifting of prior educational knowledge graphs. The graph drives a multi-agent Socratic pipeline with judged per-skill mastery updates. On a curated Mathematical Methods question bank, our system reaches $1.5\times$ the skill-coverage F1 of bare-RAG and direct-LLM baselines while remaining within 0.003 of the best baseline on test accuracy, providing preliminary evidence – under a simulated-student protocol – that the gains lie in the precision of what is learned rather than in answer-getting.

1 Introduction

Large language models (LLMs) have enabled increasingly capable conversational tutors, yet most deployed systems treat student questions as queries to be answered rather than opportunities for learning. Returning a final answer can short-circuit the reasoning process and leave the student no better prepared for the next question. Even when LLM tutors do attempt to scaffold, ungrounded prerequisite suggestions can hallucinate background or pile up tangential effort that the course does not actually require. A pedagogical tutor should instead develop the student's problem-solving skills by guiding them through exactly the prerequisite content the course material warrants — no more, no less.

From anonymous chatbot interaction logs across multiple courses, we observe that student requests cluster into three categories — *topic understanding*, *step-by-step problem-solving*, and *exam preparation* — all of which call for the same underlying capability: personalised guidance over the prerequisite skills required for the question, rather than a direct answer.

We address this need by constructing, from the course's lecture corpus, a two-layer educational knowledge graph that pairs *typed knowledge topics* (concepts, algorithms, theorems, principles, rules of practice) with *Bloom-leveled skills* attached to each topic. Unlike prior educational KGs that lift prerequisites uniformly across all Bloom levels, we use an LLM to determine, per target skill, the Bloom level at which each prerequisite topic is required — e.g., applying BFS requires Queue at Apply level but Graph only at Understand. The knowledge topic-skill graph is instantiated in an interactive Socratic tutoring application: given a student question, the tutor identifies the prerequisite skills the student has not yet mastered, runs an opening orientation dialogue, and walks the gap-filling plan one practice question at a time, updating mastery from each attempt. The whole framework is realised as a multi-agent system over a course-shared persistent layer whose structure grows with usage.

Our technical contribution is twofold, with a third integration claim:

- **Per-target Bloom lifting.** An LLM-driven procedure that, for every target skill (k_j, b_j), infers the specific Bloom level b_i at which each prerequisite topic k_i is actually required – replacing the blanket all-Bloom lifting used in prior educational KGs with a per-target decision (Sections 4.2–4.4).

- **Question-scoped mastery licensing.** A test-time mechanism that lets the student apply only the intersection of the question's required skill closure with the cumulative mastered set, producing a narrow but exercised skill set

whose precision can be measured (Section 4.6, evaluated in Section 5.2).

- **End-to-end integration.** The two mechanisms are realised in a multi-agent Socratic tutoring loop over a course-shared persistent layer whose structure grows with usage (Section 4.1).

The remainder of the paper is organised as follows. Section 2 reviews related work on educational knowledge graphs and LLM-based tutoring. Section 3.1–3.3 formalises the problem setting, the two-layer knowledge representation, and the closure-based teaching objective. Section 4.1 introduces the multi-agent framework; Sections 4.2–4.4 describe the construction and maintenance of the typed knowledge topic-skill graph, and Sections 4.5–4.6 describe the interactive Socratic tutoring loop. Section 5 reports experiments on knowledge graph quality (Section 5.1) and tutoring effectiveness against retrieval- and direct-LLM baselines (Section 5.2); under our simulated-student protocol we observe a $1.5\times$ skill-coverage F1 advantage at matched test accuracy. Section 6 concludes.

2 Related Work

LLM tutors vs. direct answering. A growing line of work argues that the most pedagogically useful LLM behaviour is not to answer the student's question but to scaffold the path toward an answer. Levonian and AUTHORS-TODO (2023) show that retrieval-augmented generation reduces hallucination in math Q&A but does not, by itself, deliver pedagogical scaffolding; David Dinucu-Jianu (2025) use reinforcement learning to align LLM responses toward Socratic tutoring; and Shi et al. (2026) introduce KMP-Bench, foregrounding the gap between *solving* and *teaching to solve*. Rather than fine-tuning an LLM to behave pedagogically, we externalise the pedagogy as a structured knowledge topic-skill graph and introduce two specific mechanisms on top of it – per-target Bloom lifting and question-scoped mastery licensing – that the LLM plans and acts over.

Educational knowledge graphs and learning-path planning. Educational KGs have long been used to model prerequisites and recommend learning paths (Xinghe Cheng, 2025; Lim et al., 2024); Wei et al. (2025) use LLM clustering to discover knowledge components from learner data. Our two-layer formulation builds on the knowledge-learning-instruction framework of Koedinger et al. (2012) and the concept-mapping tradition of Novak and Gowin (1984), but adds a typed-topic vocabulary and an LLM-driven mechanism that lifts each topic prerequisite to the *semantically appropriate* Bloom level per target – sharper than the blanket all-Bloom lifting in prior educational KGs. Dass et al. (2025) combine knowledge-based and generative AI for skill-based Q&A; we integrate construction *and* reasoning in a single Socratic tutoring pipeline.

Knowledge-graph-augmented retrieval and reasoning. Our backend builds on LightRAG (Guo et al., 2024), customising its extraction prompts to the typed educational schema. Recent KG-augmented LLM reasoning methods (Ao et al., 2025; Li et al., 2025; Yigeng Zhou, 2025; Ma et al., 2025) target *question answering*; in contrast, our graph drives a *question-asking* tutoring loop, using the graph to identify, plan, and walk the student through the prerequisite skills.

3 Preliminaries

3.1 Problem Setting and Tutoring Contract

We consider an intelligent tutor scoped to a single course C with a corpus of lecture materials D. Students enrolled in C are assumed to have mastered its prerequisite courses; the tutor neither audits nor teaches that material, presuming it when a guidance step depends on it and flagging out-of-scope content otherwise. Given a question q from student s, the tutor's contract is to help s identify and fill the gaps blocking them from answering q, rather than to answer q directly. The mechanism — an interactive loop of orientation, skill-by-skill practice with judged mastery updates, and bounded hints on imminent failure — is described in Section 4.6.

3.2 Knowledge Representation

Knowledge Topics. A *knowledge topic* k is an atomic unit of subject-matter content identifiable in D, generalising the *knowledge component* of Koedinger et al. (2012) to accommodate the breadth seen in real courses. Topics span declarative concepts (e.g., *Complex Number*), algorithmic procedures (e.g., *DFS*), theorems and principles (e.g., *Master Theorem*), and rules of practice. The set of topics extracted from D is V_K; the construction procedure is detailed in Section 4.4.

Skills. A skill is a pair $\sigma = (k, b)$ of a knowledge topic $k \in V_K$ and a Bloom level $b \in \mathcal{B}$, the

six-level vocabulary of the revised Bloom taxonomy (Anderson and Krathwohl, 2001): *Remember*, *Understand*, *Apply*, *Analyze*, *Evaluate*, *Create*. Skills are canonically identified by (k, b). The skill prerequisite graph $G_S = (V_S, E_S)$ has skills as nodes and directed *must-learn-before* edges between them. Figure 2 in Appendix A illustrates the two-layer structure on a small Algorithms-course fragment.

Student knowledge state. The knowledge state of student s is a sparse mapping

$$KS(s) = \{(\sigma, \theta(\sigma, s)) : \theta(\sigma, s) > 0\},$$

where $\theta(\sigma, s) \in [0, 1]$ measures s's mastery of σ and unobserved skills default to $\theta = 0$. We say s has *mastered* σ when $\theta(\sigma, s) \geq \tau_{\text{master}}$ for a fixed threshold τ_{master}.

3.3 Formal Objective

For a question q, let $T = Pre(q, s) \subseteq V_S$ denote the target skills required to answer q, with Bloom levels inferred from q and the inferred study goal of s. Let $S = \{\sigma \in V_S : \theta(\sigma, s) \geq \tau_{\text{master}}\}$ be the student's mastered set. If $T \subseteq S$, no guidance is needed. Otherwise, the tutor teaches every skill in the prerequisite closure of T under G_S that the student has not yet mastered:

$$Plan(q, s) = Closure_{G_S}(T) \setminus S.$$

$Closure_{G_S}(T)$ is taken under both the intra-topic Bloom ladder and the LLM-determined inter-topic prerequisite edges (Section 4.3). Skills in $Plan(q, s)$ are taught in a prerequisite-respecting topological order (Section 4.5). Figure 3 in Appendix A illustrates the construction: pink nodes are S, green are T, and blue paths are $Plan(q, s)$.

Compared to ungrounded LLM scaffolding, this formulation is *precise* (no tangential topics), *grounded* (every skill is anchored on a topic extracted from D, avoiding hallucinated prerequisites), and – under the AND-decomposition of prerequisites (Section 4.3) – *minimal* ($Plan(q, s)$ is the smallest closure of $T \setminus S$ under G_S).

4 Methodology

The system is realised as a multi-agent framework with six LLM agents and two deterministic components. A one-shot extraction agent builds $G_K(D)$; per query, a target-decomposition agent identifies $Pre(q, s)$, a planner computes the learning tree, an orientation agent opens the dialogue, and a practice-question/judge/hint trio runs the per-skill loop. Persistence is a deterministic operator over the verified additions. Each agent has an explicit input–output contract, enabling component-level ablation; the per-agent specifications are given in Sections 4.2–4.6.

4.1 System Overview

Figure 1 sketches the end-to-end pipeline. A static knowledge topic graph $G_K(D)$ is extracted once from D (Section 4.2); skills are not pre-enumerated but materialised lazily on demand (Section 4.3). At query time, the tutor (i) identifies target skills $Pre(q, s)$ on topics in V_K, (ii) computes $Plan(q, s)$, the prerequisite closure of T minus the mastered set (Section 4.5), (iii) orients the student to the question with a brief opening dialogue, and (iv) walks the plan interactively, presenting one practice question per skill and updating θ from each attempt (Section 4.6). Verified skills and skill-prerequisite edges produced during the walk are promoted into a course-shared persistent layer (Section 4.4); mastery updates feed back into $KS(s)$ for subsequent queries.

4.2 Knowledge Topic Nodes and Relations

Each topic node $k \in V_K$ carries a type label from a five-element vocabulary — *concept*, *algorithm*, *theorem*, *principle*, *rule of practice* — assigned at extraction time. Three relation types connect topics:

Hierarchical *is-a / part-of* relations, e.g., *Logistic Regression is-a Classification*.

Prerequisite directed edges $k_i \rightarrow k_j$ where learning k_j presupposes mastery of k_i. These edges seed the inter-topic skill prerequisites in Section 4.3.

Similarity / Analogy symmetric edges supporting retrieval and analogical scaffolding during guidance generation.

These three relation types mirror the way humans naturally organise knowledge (Novak and Gowin, 1984) — categorical hierarchies, procedural prerequisites, and analogical similarity — yielding a graph that doubles as a navigable substrate for learning rather than a flat retrieval index.

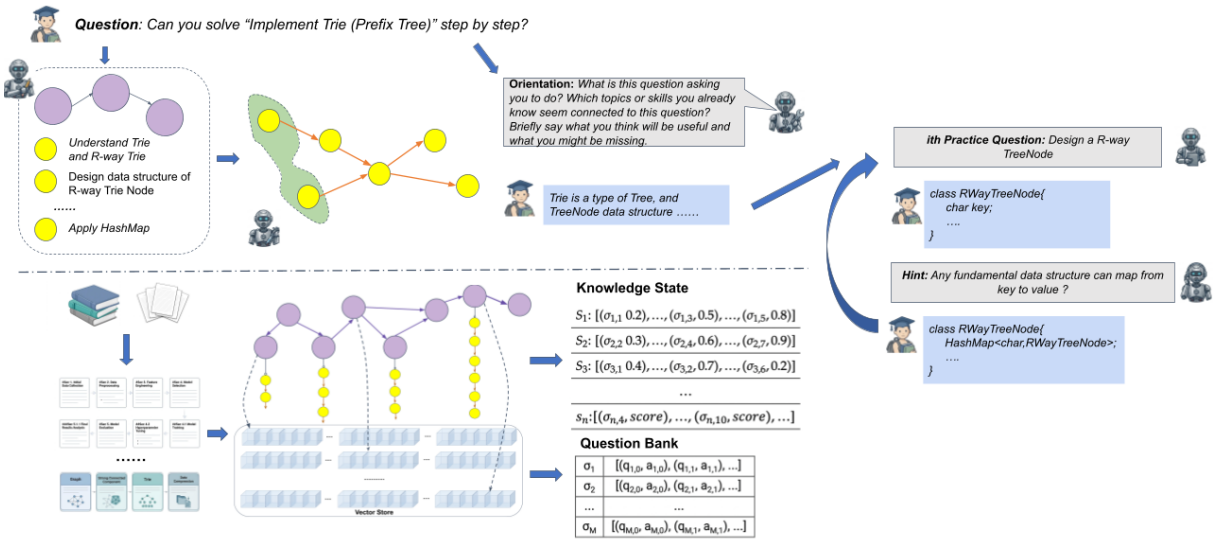

Figure 1: End-to-end pipeline of the personalised pedagogical tutor. The Knowledge Topic Graph Agent extracts $G_K(D)$ once; per query, the Target Decomposition Agent maps q to target skills, the Planner computes $Plan(q, s)$, the Orientation Agent elicits intention and prior knowledge, and the Practice Question, Judge, and Hint Agents run an interactive loop with mastery updates. Verified additions are promoted to course-shared persistent layers (G_{persist}, Q_{persist}); mastery updates feed back into $KS(s)$.

4.3 Skill Nodes and Relations

A skill node is a pair $\sigma = (k, b)$ canonically identified by a (topic, Bloom-level) tuple, with b drawn from the six-level revised Bloom vocabulary. We do not maintain a separate skill graph as a data structure: skill nodes exist as (topic, Bloom-level) labels on relation endpoints, skill relations are added to the single LightRAG-managed graph that also holds the topic relations of Section 4.2, and per-student mastery $\theta(\sigma, s)$ is stored separately in $KS(s)$. Two relation types connect skills:

Bloom ladder (intra-topic) within the same topic k, higher Bloom levels presuppose lower ones; e.g., (Trie, Apply) requires (Trie, Understand).

Lifted prerequisite (inter-topic) $(k_i, b_i) \rightarrow (k_j, b_j)$ where $k_i \rightarrow k_j$ is a topic prerequisite and b_i is the Bloom level of k_i actually required to attain (k_j, b_j) — determined at runtime rather than blanket-lifted across all Bloom levels (Section 4.4).

4.4 Graph Construction and Maintenance

Topic extraction. We customise the LightRAG framework (Guo et al., 2024) to extract, store, and retrieve G_K over the lecture corpus D, chosen for its space and cost efficiency relative to dense-vector alternatives. Lectures are typically structured documents, so we segment D section-by-section and run topic-and-relation extraction over each section independently, then merge per-section outputs via canonicalisation. We adapt LightRAG's extraction prompt to elicit the five topic types of Section 4.2 (full prompt in Appendix A). The document's hierarchical structure (chapter $\rightarrow$ section $\rightarrow$ subsection) is additionally treated as a source of hierarchical topic relations, preserving the natural *is-a* / *part-of* organisation present in the lecture material.

Skill relation derivation. Bloom-ladder edges within each topic are deterministic. Inter-topic lifted prerequisites are derived at runtime: given a target skill (k_j, b_j), the system retrieves the prerequisite topics of k_j from G_K and prompts an LLM to determine, for each prerequisite topic k_i, the Bloom level b_i actually required to attain (k_j, b_j). For example, (BFS, Apply) may require (Queue, Apply) and (Graph, Understand) rather than (Graph, Apply). The prompt is in Appendix A. The resulting edges are written to the LightRAG-managed graph.

Persistence and canonicalisation. The active graph at time t is $G_{\text{active}}(t) = G_K(D) \cup G_{\text{persist}}(D, t) \cup B_{\text{eph}}(q, s, t)$, where G_{persist} is the course-shared persisted layer and B_{eph} the current query's pending additions. The operator Π promotes topic-level additions eagerly upon verification but admits LLM-derived skill relations only after the relation has been used or independently re-derived. New entries are deduplicated against

persisted ones via embedding similarity and normalised to the fixed type / Bloom vocabularies. Mastery θ stays per-student private.

Question bank. Parallel to G_{persist}, a course-shared content store $Q_{\text{persist}}(D, t)$ accumulates (question, reference-answer) pairs tagged with skill identity (k, b). The bank is seeded from D's extracted exercises where lecture-tagged problems exist, and grown by LLM-generated items produced by the Practice Question Agent (Section 4.6). New items enter as *candidate*; an item is promoted to *verified* after one successful use (Judge Agent produced a score, no malformedness flag). Verified items become the default retrieval target on future calls for the same skill.

4.5 Pedagogical Planning

The planner computes $Plan(q, s)$ by traversing prerequisite edges of G_S backward from T, stopping at skills already in S or at topics with no further prerequisites, then topologically sorting the result. If the closure requires a topic outside V_K, the planner emits a scope notice for the affected target rather than expanding outside the corpus.

4.6 Guidance Generation

The tutor walks the prerequisite-ordered learning tree from Section 4.5 in an interactive loop coordinated by four LLM agents (Orientation, Practice Question, Judge, Hint). Each practice question is itself the pedagogical artifact, phrased to include whatever framing the student needs to attempt it – there is no separate explanation channel.

Orientation. Two opening prompts – *intention* and *recall* – mirror Polya's "understand the problem" phase (Pólya, 1945). Advisory only: the responses do not modify S or T.

Practice question. For each $\sigma = (k, b)$, retrieve a verified item from Q_{persist} or, failing that, generate a new (q_i, a_i) from lecture passages for k at Bloom level b and emit it as a candidate. Bundling the reference answer with the question reduces judging to evaluative comparison.

Judge and mastery update. The judge scores the student's attempt against a_i under a Bloom-b rubric, producing $j \in [0, 1]$; mastery is updated by $\theta(\sigma, s) \leftarrow \theta(\sigma, s) + \eta \cdot (j - \theta(\sigma, s))$, with $\eta \in (0, 1]$. A successfully judged candidate is promoted to verified. The loop advances once $\theta(\sigma, s) \geq \tau_{\text{master}}$.

Hint. After $M - 1$ failed attempts, the agent emits one short clue grounded in the relevant lecture passages and targeted at the gap revealed by prior attempts. If the final attempt still fails, the loop force-advances. This is the only explanatory text the tutor emits outside of question phrasing.

Newly materialised skills are proposed to Π (Section 4.4); out-of-scope topics are presumed or flagged, never elaborated. The session concludes when every target skill in $Pre(q, s)$ has been attempted.

5 Experiments

We evaluate the system along two tracks: (i) the quality of the constructed knowledge topic graph and the LLM-determined Bloom lifting (Section 5.1); and (ii) the pedagogical effectiveness of the multi-agent tutor against vanilla-RAG and direct-LLM baselines, using simulated students with differentiated mastery profiles (Section 5.2).

5.1 Knowledge Graph Quality

Setup. We instantiate the extraction pipeline (Section 4.4) on the lecture corpus of the Princeton University Algorithms course — 23 lecture markdown files covering data structures, sorting, searching, graph algorithms, string algorithms, data compression, and advanced topics. The same pipeline is also applied to the Mathematical Methods 2 (MM2) corpus used by Section 5.2; we report KG-quality numbers on Algorithms here because expert gold annotation is available for it.

Gold annotation. An expert annotator, using the Streamlit annotation tool with LLM-suggested candidates that are then verified by hand, curated four CSV files of gold annotations: a topic list of 143 entries with type labels, 54 verified topic prerequisite edges, 44 hierarchy edges (*is-a* / *part-of*), and 91 (target skill, prerequisite topic, required Bloom level) triples covering Bloom levels Understand through Evaluate. We additionally use a coarser course-outline reference (8 chapters with subtopics, 30 unique entries and 23 chapter→subtopic edges) as a second evaluation tier capturing syllabus-level structure.

Metrics. For topic-level matching, we embed both gold topic names and extracted entity names with ALL-MINILM-L6-V2 and map each extracted entity to its closest gold topic at cosine similarity ≥ 0.7; this alias map is used for all set-intersection metrics. **Topic precision / recall / F1** compare canonicalised extracted topics with the gold list. **Type accuracy** measures, on matched topics, whether the extracted type label maps to the

gold 5-type vocabulary. **Prerequisite** and **Hierarchy** P/R compare the corresponding edge sets after canonicalisation. **Bloom-lift accuracy** is computed per gold (target skill, prerequisite topic, required Bloom level) triple by querying the LLM with the lift prompt (Appendix A) and comparing the predicted level with the gold required level. Because gold annotation is partial across the corpus, we report a *scope-aware* precision variant that restricts the denominator to extracted entities/edges originating from lecture files containing at least two gold-matched entities.

Methods compared. *Bare LightRAG* (Guo et al., 2024) with its default extraction prompt is the baseline. *Ours* is the typed educational extraction (Section 4.4, prompt in Appendix A) with the 5-type vocabulary, the exclusion list (people, programming-language artefacts, narrative roles), and the tightened relation-type definitions. Comparison with prior educational-KG methods such as KnowLP is left to future work because their public artifacts do not produce Bloom-typed skill relations.

Results. Table 1 reports KG quality. Our method extracts 36% fewer entities than the bare-LightRAG baseline (1,404 vs 2,184) while improving topic precision by 44% (scope-aware: 0.060 vs 0.042) and F1 by 39% (0.089 vs 0.064). Type accuracy on matched topics improves from 0.680 to 0.797 (+17% relative), reflecting the constrained 5-type vocabulary and per-type definitions. Most consequentially for downstream pedagogical reasoning, our method emits typed pedagogical relations at scale — 78 prerequisite edges and 464 hierarchy edges — whereas bare LightRAG produces 0 prerequisite edges and 6 hierarchy edges, since its default relation labels are generic free-text keywords without pedagogical typing. Hierarchy recall against gold is 3.5× that of the baseline (0.159 vs 0.045); against the course-outline hierarchy it rises from 0 to 0.130.

Bloom-lift accuracy. Table 2 reports accuracy on the 91 gold Bloom-lift triples. Gold required-Bloom values are skewed: 78% are *Understand*. The naive "always Understand" baseline therefore achieves 0.780 overall, but **0.000** on the 20 triples where the gold required level is not Understand — the cases where Bloom lifting is non-trivial. Our LLM-determined lifting achieves 0.604 overall and **0.700** on this discriminative subset. The discriminative metric is the right framing: only a method that performs an actual Bloom-lifting inference can identify the cases that require a non-default level; the always-default baseline gets free correctness on the trivial cases and zero credit on the cases that matter.

Metric	Bare LightRAG	Ours
Extracted topics	2,184	**1,404**
Topic P (scope-aware)	0.042	**0.060**
Topic R (gold, 143)	**0.524**	0.483
Topic R (course-outline, 30)	**0.900**	0.867
Topic F1 (gold)	0.064	**0.089**
Type accuracy (on matched)	0.680	**0.797**
Extracted prereq edges	0	**78**
Prereq R (gold, 54)	0.000	0.000
Extracted hierarchy edges	6	**464**
Hierarchy R (gold, 44)	0.045	**0.159**
Hierarchy R (course-outline, 23)	0.000	**0.130**

Table 1: Knowledge graph quality on Princeton Algorithms. Topic matching uses embedding-similarity canonicalisation. Precision is reported under scope-aware filtering. Best per metric in bold.

Method	Overall (n=91)	Discriminative (n=20)
predict "Understand"	0.780	0.000
predict target Bloom level	0.418	—
Ours (LLM-determined)	0.604	**0.700**

Table 2: Bloom-lift accuracy on the 91 gold triples. *Overall* pools all triples; *Discriminative* restricts to the 20 triples whose gold required Bloom level is not Understand — the cases where the lifting decision is non-trivial.

Discussion. The headline gain — the typed-extraction pipeline yielding pedagogical relations that bare LightRAG cannot produce — is robust on hierarchical structure (3.5× gold-hierarchy recall and a non-trivial recall against the course outline where the baseline scores zero) and on the Bloom-lifting capability (0.70 on the discriminative subset vs 0 for naive baselines). The small loss in fine-grained topic recall (0.483 vs 0.524) is a deliberate trade for the +44% precision and +17% type accuracy: the topics our exclusion list removes (people, language artefacts, narrative roles) are not in the gold either, so most of the recall delta is from a handful of gold topics whose granularity differs from what the typed prompt now retains. The remaining open gap is prerequisite recall, where neither method matches the gold's 54 inter-topic curricular edges — our method's 78 extracted prereqs are typed but at a different granularity (often within-topic operations like *Stack→Push Operation* rather than inter-topic dependencies like *Resizing Arrays→Stack*). Closing this granularity gap is the main remaining KG-quality task.

5.2 Tutoring Effectiveness

We evaluate whether our multi-agent pipeline teaches a simulated student more effectively than two ablations: a retrieval-augmented chatbot (*bare LightRAG*) and a direct LLM without retrieval (*GPT-4o-direct*). The experiment is conducted on the *Mathematical Methods 2* (MM2) course.

Question source. We sample from a curated MM2 question bank of 132 items, evenly distributed across 22 course topics and the six Bloom levels (one question per $\langle$*topic*, *Bloom level*$\rangle$ cell). Each question carries the corresponding metadata and a short "key-point" canonical answer. Sampling is stratified by Bloom level to balance the resulting test set across the taxonomy.

Systems compared.

- **Bare LightRAG** – the chatbot retrieves lecture chunks via LightRAG with default extraction and produces a single tutor response.
- **GPT-4o-direct** – a direct LLM answer to the question, no retrieval, no scaffolding.
- **Our system** – the full multi-agent pipeline of Section 4.1: target-skill identification, skill closure derived from the curated topic hierarchy and the LLM-determined Bloom lifting (Section 4.4), orientation step, and a per-skill practice loop with mastery updates that runs once per training question.

Three-LLM-family bias control. To prevent any single LLM family from dominating both system output and evaluation, the three roles are distributed across three providers: (i) all in-pipeline LLM calls – chatbot responses, practice-question generation, simulated-student attempts – use the same Azure OpenAI (GPT-4o) deployment; (ii) reference enrichment (expanding brief key-points into full reference answers) and skill extraction over baseline responses use *Claude* (Anthropic); (iii) the LLM-as-judge that scores both per-skill attempts and the test-question attempts uses *Gemini* (Google). A reviewer therefore cannot attribute any system's lead to the judge favouring its own model family.

Train / test split protocol. A naive per-question comparison – judging each system's output against the same question it tutored on – would unfairly favour baselines, since they hand the student a near-complete answer that the judge then scores. We adopt instead a cumulative train / test design: from the stratified sample of N questions we hold out $1/5$ as a *test set* and use the remaining $4/5$ as a *training set*. Each $\langle$*system*, *profile*$\rangle$ pair produces one *trained student*: we initialise the simulated student with the profile's mastery state, walk through every training question using only that system, and accumulate the student's mastery state across the training stream. After training, the trained student attempts every test question in turn; test attempts are judged independently of the training stream, so success requires generalisation rather than answer copy.

Simulated student. We use an LLM-as-student parameterised by a sparse mastery state $\theta(\sigma, s) \in [0, 1]$ over the union of all sampled questions' skill closures. Three profiles initialise θ differently: *novice* ($\theta \approx 0$); *mid-course* (Remember/Understand mastered for most subtopics); *nearly-done* (Remember/Understand/Apply mastered for most subtopics). θ accumulates across the training stream:

- Baselines (bare LightRAG, GPT-4o-direct): after each training question's tutor response, Claude extracts under a constrained *topic* $\times$ *Bloom level* vocabulary which skills the response would teach a student. Those skills are added to θ with $\theta = 1.0$, modelling a best-case "the student fully absorbs everything the response says."
- Our system: θ is updated by the per-skill practice loop directly (Section 4.6); θ for skills outside the question's plan is unchanged.

At test time, the student attempts each test question under the same skill-license rule as Section 4.6: it may only apply the *question-scoped subset* of its mastered set (the intersection of the test question's required skill closure with the cumulative mastered set $\theta \geq \tau_{\text{master}} = 0.75$). The asymmetric mechanism by which each system grows θ – via skill extraction over baseline responses, or via the practice loop – isolates the methodological contribution while keeping the test-time interface identical across systems.

Metrics. Each trained student (one per $\langle$*system*, *profile*$\rangle$ pair) is scored on two complementary axes, both aggregated over the held-out test set:

- **Test-set accuracy** – the Gemini judge's $[0, 1]$ score of the student's attempt on each

test question, against the question's Claude-enriched reference answer, averaged over the test set.

- **Skill coverage F1** – the harmonic mean of precision and recall of the trained student's cumulative mastered set against the union of required skill closures over the test set. This is intended to indicate whether the system's claimed mastery aligns with the skills the held-out questions demand, rather than only with the skills it tutored on.

Configuration. For our system, each training question runs a per-skill practice loop with at most $M = 3$ attempts per skill before forced advancement and a single hint emitted before the final attempt (Section 4.6); mastery threshold $\tau_{\text{master}} = 0.75$ and update rate $\eta = 0.6$. The train / test split is stratified by Bloom level and seeded for reproducibility; the same split is reused across all $\langle$*system*, *profile*$\rangle$ pairs.

Train / test results. We run the full protocol on a stratified sample of 20 questions (14 training, 6 held-out test) crossed with the three profiles and the three systems, yielding 18 test attempts per system. Table 3 reports per-(system, profile) accuracy and skill F1, with per-system means.

Under this protocol, our system shows the largest *skill coverage F1* – the metric most directly tied to the paper's contribution, which compares the trained student's mastered set against the held-out test questions' required skill closures. Aggregated, our system reaches 0.502 versus 0.329 for bare LightRAG and 0.325 for GPT-4o-direct, a $1.5\times$ difference overall, with a $2.1\times$ gap in the novice profile (0.893 vs 0.426 / 0.413), where the metric is least confounded by inherited pre-mastery. The ordering on skill F1 is consistent across all three profiles, though the per-profile gaps are sensitive to the small N (see Limitations).

The test-accuracy story is more subtle. All three systems land between 0.80 and 0.83 on aggregate. We attribute the baselines' competitiveness on this axis to liberal mastery bookkeeping: Claude skill extraction over each training response sets $\theta = 1.0$ for every topic the response mentions, so after 14 training questions the baselines' mastered sets are wide but contain many skills the student has not exercised. Our system's mastery is updated only by per-skill judging in the practice loop, producing a narrower mastered set whose precision is testable.

System	Profile	Acc.	Skill F1
Bare LightRAG	novice	0.842	0.426
	mid-course	0.817	0.306
	nearly-done	0.833	0.254
	mean	*0.831*	*0.329*
GPT-4o-direct	novice	0.675	0.413
	mid-course	0.800	0.315
	nearly-done	0.917	0.246
	mean	*0.797*	*0.325*
Ours	novice	0.733	**0.893**
	mid-course	**0.900**	**0.350**
	nearly-done	0.850	**0.263**
	mean	***0.828***	***0.502***

Table 3: Test-set accuracy and skill-coverage F1, per (system, profile), with the across-profile mean per system. 6 test questions per profile = 18 cells per system. Our system reaches the highest skill F1 in every profile (a $1.5\times$ aggregate advantage, $2.1\times$ in novice) while remaining within 0.003 of the best baseline on test accuracy. Aggregate skill precision and recall (omitted from this table for space): bare LightRAG $0.244/0.702$, GPT-4o-direct $0.237/0.702$, ours $0.421/0.893$.

The skill-F1 gap is consistent with this interpretation – baselines reach comparable test accuracy via a wider mastered set, while our system reaches it via a smaller and judged one – though we note this is correlational; an ablation against blanket-Bloom lifting would more directly attribute the gap to the per-target lifting mechanism, and is left to future work.

6 Conclusion

We presented a pedagogical tutor that fills prerequisite-skill gaps rather than answering questions. Its two technical contributions are *per-target Bloom lifting* – determining, per target skill, the Bloom level at which each prerequisite topic is actually required – and *question-scoped mastery licensing* at test time, both realised in a multi-agent Socratic loop. On a Mathematical Methods question bank, the system reaches $1.5\times$ the skill-coverage F1 of bare-RAG and direct-LLM baselines while matching their test accuracy within 0.003, providing preliminary evidence – under a simulated-student protocol – that the gains lie in the precision of what is learned. Broader validation, ablations, and human evaluation are the natural next steps.

Limitations

The evaluation in Section 5.2 relies on a closed-loop synthetic stack: the simulated student is an LLM, the judge is an LLM, the skill-extractor

used to derive baseline mastery is an LLM, and the reference answers used by the judge are LLM-enriched. The three LLM families are deliberately split (Azure OpenAI for in-pipeline calls, Claude for reference enrichment and skill extraction, Gemini for judging) so that no single family scores both system output and reference, but the broader risk – that we are measuring inter-LLM consistency rather than pedagogical quality – is not eliminated. Reproducing these results with human-graded reference answers and at least a small human-rated tutoring session study is the most important next validation step.

The sample size is small. We report results on $N = 20$ stratified questions (14 train / 6 test) with one random seed and a single course (Mathematical Methods 2). Per-profile differences and the $2.1\times$ novice-skill-F1 figure should be read as point estimates without confidence intervals; a wider run with multiple seeds and a second course is planned.

The simulated student's three profiles are stylised. *Mid-course* and *nearly-done* pre-master skills by fiat (the prior knowledge is asserted rather than acquired through tutoring), which inflates skill-precision denominators for those profiles and makes the F1 metric most informative for the novice profile.

We compare against retrieval-augmented and direct-LLM baselines rather than against tutoring-tuned LLMs (e.g. Socratic prompting, pedagogically RL-aligned models). An ablation that replaces our per-target Bloom lifting with blanket all-Bloom lifting – which would more directly isolate the contribution of the lifting mechanism – is the most consequential missing experiment, and is left to a future revision.

References

Lorin W. Anderson and David R. Krathwohl. 2001. *A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives*. Longman, New York.

Tu Ao, Yanhua Yu, Yuling Wang, Yang Deng, Zirui Guo, Liang Pang, Pinghui Wang, Tat-Seng Chua, Xiao Zhang, and Zhen Cai. 2025. Lightprof: a lightweight reasoning framework for large language model on knowledge graph. In *Proceedings of the Thirty-Ninth AAAI Conference on Artificial Intelligence and Thirty-Seventh Conference on Innovative Applications of Artificial Intelligence and Fifteenth Symposium on Educational Advances in Artificial Intelligence*, AAAI'25/IAAI'25/EAAI'25. AAAI Press.

Rahul K. Dass, Rochan H. Madhusudhana, Erin C. Deye, Shashank Verma, Timothy A. Bydlon, Grace Brazil, and Ashok K. Goel. 2025. Enhanced question-answering for skill-based learning using knowledge-based AI and generative AI. *arXiv preprint arXiv:2504.07463*.

Nico Daheim Ido Hakimi Iryna Gurevych Mrinmaya Sachan David Dinucu-Jianu, Jakub Macina. 2025. From problem-solving to teaching problem-solving: Aligning LLMs with pedagogy using reinforcement learning. *arXiv preprint arXiv:2505.15607*.

Z. Guo, L. Xia, Y. Yu, T. Ao, and C. Huang. 2024. Lightrag: Simple and fast retrieval-augmented generation. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*. ArXiv:2410.05779.

Kenneth R. Koedinger, Albert T. Corbett, and Charles Perfetti. 2012. The knowledge-learning-instruction framework: Bridging the science-practice chasm to enhance robust student learning. *Cognitive Science*, 36(5):757–798.

Zachary Levonian and AUTHORS-TODO. 2023. Retrieval-augmented generation to improve math question-answering: Trade-offs between groundedness and human preference. *arXiv preprint arXiv:2310.03184*. NeurIPS 2023 Workshop on Generative AI for Education (GAIED).

Feiyang Li, Peng Fang, Zhan Shi, Arijit Khan, Fang Wang, Weihao Wang, Zhangxin-hw, and Cui Yongjian. 2025. Cot-rag: Integrating chain of thought and retrieval-augmented generation to enhance reasoning in large language models. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 3119–3171.

Joy Jia Yin Lim, Ye He, Jifan Yu, Xin Cong, Daniel Zhang-Li, Zhiyuan Liu, Huiqin Liu, Lei Hou, Juanzi Li, and Bin Xu. 2024. Personalized learning path planning through goal-driven learner state modeling. *arXiv preprint*.

Chuangtao Ma, Yongrui Chen, Tianxing Wu, Arijit Khan, and Haofen Wang. 2025. Large language models meet knowledge graphs for question answering: Synthesis and opportunities. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing,EMNLP2025*, pages 24589–24608. Association for Computational Linguistics.

Joseph D. Novak and D. Bob Gowin. 1984. *Learning How to Learn*. Cambridge University Press, New York.

George Pólya. 1945. *How to Solve It: A New Aspect of Mathematical Method*. Princeton University Press, Princeton, NJ.

Weikang Shi, Houxing Ren, Junting Pan, Aojun Zhou, Ke Wang, Zimu Lu, Yunqiao Yang, Yuxuan Hu, Linda Wei, Mingjie Zhan, and Hongsheng Li. 2026. From solver to tutor: Evaluating the pedagogical intelligence of LLMs with KMP-Bench. *arXiv preprint arXiv:2603.02775*.

Yumou Wei, Paulo Carvalho, and John Stamper. 2025. Kcluster: An llm-based clustering approach to knowledge component discovery. In *Proceedings of the Educational Data Mining (EDM) 2025 conference*. International Educational Data Mining Society.

Jiapu Wang Liangda Fang Chaobo He Quanlong Guan Shirui Pan Weiqi Luo Xinghe Cheng, Zihan Zhang. 2025. GraphRAG-induced dual knowledge structure graphs for personalized learning path recommendation. *arXiv preprint arXiv:2506.22303*.

Yifan Lu Jing Li Fangming Liu Meishan Zhang Yequan Wang Daojing He Honghai Liu Min Zhang Yigeng Zhou, Wu Li. 2025. Reflection on knowledge graph for large language models reasoning. In *Findings of the Association for Computational Linguistics: ACL 2025*.

A Supplementary Material

A.1 Illustrative Figures

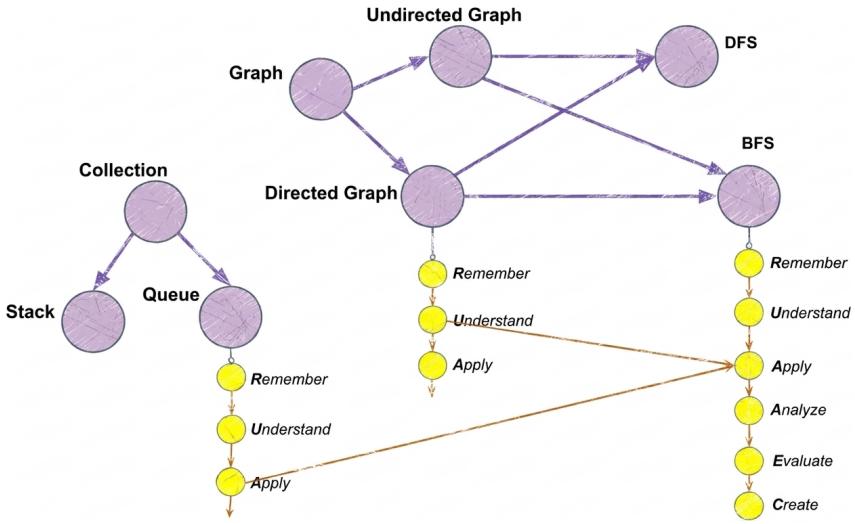

Figure 2: Two-layer structure of knowledge topics and skills, on a fragment from an Algorithms course (referenced from Section 3.2). The upper layer is the knowledge topic graph G_K: nodes are topics (*Collection*, *Queue*, *Graph*, *Undirected Graph*, *Directed Graph*, *BFS*, *DFS*, ...) connected by hierarchical, prerequisite, and similarity relations. The lower layer is the skill graph G_S: each topic carries a Bloom ladder of skills (k, b) from *Remember* to *Create*, connected by intra-topic Bloom edges (vertical) and inter-topic edges lifted from the topic prerequisite relation (e.g., (Queue, Apply) $\rightarrow$ (BFS, Apply)).

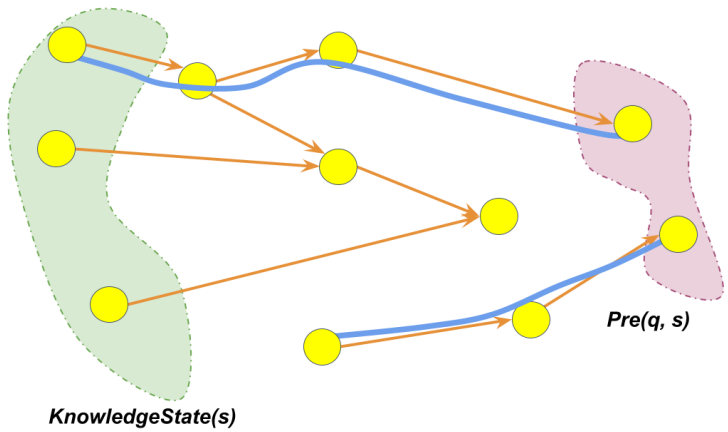

Figure 3: Skill nodes (k, b) with prerequisite edges (referenced from Section 3.3). Green: target skills $T = Pre(q, s)$. Pink: skills mastered by s. Blue Paths: $Plan(q, s) = Closure_{G_S}(T) \setminus S$, the skills the tutor will teach.

A.2 Prompts

Knowledge Topic Extraction Prompt. The system prompt below drives the typed-LightRAG extraction over the lecture corpus, referenced from Section 4.4. Placeholders in curly braces (`{entity_types}`, `{tuple_delimiter}`, `{examples}`, `{language}`, `{completion_delimiter}`) are filled by LightRAG at extraction time. The five-element `{entity_types}` vocabulary corresponds to the typed-topic schema of Section 4.2; the four-element relation-type prefix vocabulary corresponds to the topic-relation taxonomy of the same section.

```
---Role---
You are a Knowledge Graph Specialist responsible for extracting
entities and relationships from the input text.
```

```
---Instructions---
1. Entity Extraction & Output:
   * Identification: Identify clearly defined and meaningful entities
     in the input text.
   * Entity Details: For each identified entity, extract:
     - entity_name: The name of the entity. Title-case if case-
       insensitive. Use consistent naming across the extraction.
     - entity_type: Categorize using exactly one of: {entity_types}.
       Definitions:
         * Concept: declarative unit of subject-matter content
           (e.g., "Trie", "Binary Search Tree", "Complex Number",
           "Graph"). Data structures count as Concept.
         * Algorithm: step-by-step procedure (e.g., "DFS",
           "Quicksort", "Gaussian Elimination", "Hibbard Deletion").
         * Theorem: formally proven mathematical statement
           (e.g., "Master Theorem", "Pigeonhole Principle").
         * Principle: general guiding idea or design pattern
           (e.g., "Divide and Conquer", "Dynamic Programming",
           "Greedy Choice").
         * RuleOfPractice: practical heuristic for the student
           (e.g., "Always verify the base case in recursion",
           "Normalise units before integration").
       Do not invent new types. If a candidate entity does not fit
       any of the five, drop it entirely rather than tagging "Other".
     - entity_description: Concise yet comprehensive description of
       the entity's attributes and activities, based solely on the
       input text. Prefix every entity description with [CORE] or
       [AUXILIARY]:
         * [CORE]: a central concept, algorithm, data structure, or
           topic taught in this course.
         * [AUXILIARY]: from another domain (e.g. calculus, physics)
           mentioned only as a tool, analogy, or incidental reference.
   * Output Format - Entities: 4 fields delimited by
     {tuple_delimiter}, on a single line, prefixed by "entity":
       entity{tuple_delimiter}entity_name{tuple_delimiter}
       entity_type{tuple_delimiter}entity_description
   * Entities NOT to extract -- drop, do not output:
     - People (authors, instructors, students): e.g., "Robert
       Sedgewick", "Kevin Wayne", "the lecturer".
     - Programming-language artefacts: e.g., "Java Generics",
       "Iterator interface", "Comparable", "java.util.Stack", "malloc".
     - Generic narrative roles: e.g., "Client", "User", "Programmer",
       "Student", "Theoretician".
     - Document or course metadata: "Lecture 1", "Section 2",
       "this course", "Application", "Slide".
     - Standalone properties or acronyms when the parent topic
       already exists: do not create a separate node for "LIFO" if
       "Stack" is already extracted; describe "LIFO" inside the
       Stack entity description. Same for "FIFO" if "Queue" exists.
     - Specific code snippets, function names, or library symbols.
   * Self-check before emitting each entity: "Would an instructor
     list this in the course syllabus?" If no, drop it.

2. Relationship Extraction & Output:
   * Identification: Identify direct, clearly stated, and meaningful
     relationships between previously extracted entities.
   * N-ary Relationship Decomposition: If a single statement involves
     more than two entities, decompose into binary pairs.
   * Relationship Details:
     - source_entity: Name of source entity. Consistent naming.
     - target_entity: Name of target entity. Consistent naming.
     - Relation Type Prefix (REQUIRED): The FIRST comma-separated
       keyword in relationship_keywords MUST be exactly one of:
       hierarchical, prerequisite, similarity, other.
         * hierarchical: is-a or part-of.
             - is_a: child is a specific kind of parent.
               E.g., "R-way trie" is_a "Trie".
             - part_of: child is a sub-component or step of parent.
               E.g., "Pivot Selection" part_of "Quicksort".
         * prerequisite: to learn the TARGET, the student must FIRST
```

```
          have learned the SOURCE. The dependency must be
          pedagogical (curricular ordering), not narrative
          ("X is mentioned before Y") nor implementational
          ("class A extends class B").
            - Positive examples:
                Recursion       -> Tree Traversal
                Stack           -> Push Operation
                Probability     -> Bayesian Network
            - Counter-examples (DO NOT emit as prerequisite):
                Author          -> Book          (narrative)
                Client          -> Student       (course-role)
                Observation     -> Hypothesis    (rhetorical)
                Programmer      -> Student       (narrative)
                Java Comparable -> Priority Queue (implementation)
            - Self-check: "If the student did not know the source,
              would learning the target be genuinely impossible or
              significantly harder?" If no, do not extract.
        * similarity: source and target share enough conceptual
          structure that knowing one helps learn the other by
          analogy. E.g., Stack ~ Queue, DFS ~ BFS. Not for mere
          co-occurrence in the same paragraph.
        * other: any functional, applicability, attribution, or
          interaction relation that does not fit the first three.
    - relationship_description: A concise explanation of the
      relationship, providing a clear rationale for the connection.
  * Output Format - Relationships: 5 fields delimited by
    {tuple_delimiter}, on a single line, prefixed by "relation":
      relation{tuple_delimiter}source_entity{tuple_delimiter}
      target_entity{tuple_delimiter}relationship_keywords
      {tuple_delimiter}relationship_description

3. Delimiter Usage Protocol:
   * The {tuple_delimiter} is a complete, atomic marker and must not
     be filled with content. It is strictly a field separator.

4. Relationship Direction & Duplication:
   * Treat all relationships as undirected unless explicitly stated
     otherwise. Avoid outputting duplicate relationships.

5. Output Order & Prioritization:
   * Output all extracted entities first, then all relationships.
   * Within relationships, prioritize those most significant to the
     core meaning of the input text first.

6. Context & Objectivity:
   * Write entity names and descriptions in the third person.
   * Explicitly name the subject or object; avoid pronouns such as
     "this article", "this paper", "I", "you", "he/she".

7. Language & Proper Nouns:
   * The entire output (entity names, keywords, descriptions) must
     be written in {language}.
   * Proper nouns are retained in their original language if a
     proper, widely accepted translation is unavailable.

8. Completion Signal: Output the literal string
   {completion_delimiter} only after all entities and relationships,
   following all criteria, have been completely extracted.

---Examples---
{examples}
```