ChatGPT

Exam-Prep Question Generator + Quiz Master

Abstract

We present *Exam-Prep Question Generator + Quiz Master*, a multi-agent system for automated creation and evaluation of educational quizzes. Our architecture comprises three specialized agents (Planner, Generator, Evaluator) that decompose exam topics into questions, generate multiple-choice questions (MCQs) via a fine-tuned LLM, and classify question difficulty. This modular design follows the simple, composable agent patterns advocated in recent AI systems research [1] [2]. For MCQ generation, we fine-tune a Falcon-7B (or GPT-J) model using Low-Rank Adaptation (LoRA) to efficiently adapt to an educational corpus. For difficulty estimation, we fine-tune a DistilBERT classifier on a small labeled dataset of question difficulty levels. We fabricate evaluation results for illustration: our generator achieves a BLEU-4 score of ~0.45 against reference questions, and the difficulty classifier attains ~80% accuracy on held-out data. A pilot user study rated the generated quizzes 4.2/5 on average for quality and diversity. We include visualizations such as training loss curves, confusion matrices, and system flow diagrams. The proposed system demonstrates that LLM agents can automate exam question creation and assembly into quizzes, with room for improvement through richer data and interactive refinement.

Introduction and Problem Background

Automated generation of exam questions is of growing interest in educational technology. High-quality practice problems and quizzes are vital for learning, but manually authoring them is labor-intensive and requires domain expertise. Large language models (LLMs) offer the potential to automate question generation (QG) [3], but producing pedagogically valid, well-calibrated questions remains challenging. In particular, controlling the *difficulty* of questions is crucial for adaptive learning; recent work highlights methods for difficulty-controllable question generation [4]. Simultaneously, advanced system designs emphasize breaking problems into simpler components using LLM-based agents [1].

Building on these trends, we design a multi-agent pipeline that takes a subject topic as input and produces a customized quiz. Following Anthropic's guidance to use *simple, composable patterns rather than complex frameworks* [1] [5], our system comprises three agents with distinct roles (planner, generator, evaluator). Each agent is an LLM enhanced with external knowledge access (retrieval) and memory, often called an *augmented LLM* [6]. An augmented LLM can retrieve domain facts or use tools to improve output relevance. We treat each agent call as one step in a chained workflow [7]. Early outputs from one agent become inputs to the next, similar to Anthropic's *prompt chaining* pattern [7]. This design balances complexity and reliability: a linear agent pipeline is a predictable workflow with high accuracy [2] [8]. If feedback or branching were needed, a routing workflow could dispatch tasks to specialized sub-agents [9], but in our case we maintain a fixed sequence.

In summary, this project investigates how to leverage LLMs for automatic quiz generation by combining a *planner* that outlines questions, a *generator* that writes each MCQ, and an *evaluator* that checks difficulty and quality. We connect relevant prior work on agent-based AI and question generation: Anthropic's frameworks for LLM agents [1] [7], architectural pattern catalogs for multi-agent design [10] [11], and

recent MCQ generation studies [3] [4]. Our contributions include (1) a detailed system architecture for exam-prep agents, (2) data preparation and fine-tuning methods for generating and classifying questions, and (3) experimental evaluation metrics for question quality and difficulty alignment. Fabricated illustrative results demonstrate the system's potential, and we discuss extensions such as RAG integration and user interfaces.

Related Work

Our approach builds on research in LLM-based agent systems and automated question generation.

LLM Agents and Architecture Patterns. Modern AI often uses *agentic* systems where models orchestrate tasks dynamically [12]. Anthropic et al. emphasize using simple LLM chains and workflows rather than monolithic frameworks [1] [5]. In particular, the *augmented LLM* (a base LLM with retrieval, tools, memory) is a common building block [6]. Two key workflow patterns are *prompt chaining* (sequentially decomposing a task) and *routing* (classifying inputs then delegating to specialized agents) [13] [9]. Our pipeline (planner→generator→evaluator) is essentially a sequential orchestration of LLM calls [7] [2]. The Azure architecture guide also notes that sequential orchestration "chains AI agents in a predefined, linear order" and is suited for step-by-step tasks [2]. Such multi-agent orchestration yields benefits like specialization, modularity, and scalability [14]. However, multi-agent systems introduce complexity: researchers note the need to consider interactions (e.g. agent coordination, possible collusion or correlated failures) when designing them [11] [14].

Educational Question Generation. Automated question generation (QG) has a long history in NLP and AI for education. Recent work leverages powerful LLMs for MCQ generation, but notes challenges of hallucination and difficulty alignment. For instance, Yao et al. highlight that GPT-4 and similar models struggle with specialized exam-style MCQs due to outdated knowledge and hallucinations [3]. They propose an iterative *self-refine* method (LLM-as-critic and self-correction) to improve question quality and difficulty in medical exams [15]. Similarly, Tomikawa & Uto (2024) develop methods for *difficulty-controllable* MCQ generation using item response theory, enabling the generation of questions tailored to a target difficulty level [4]. These works underscore our motivation to explicitly classify question difficulty. They also suggest evaluating generation quality not just by text overlap (BLEU/ROUGE) but by pedagogical relevance.

Other AI education systems also use multi-agent setups. Zhang et al. introduce *EduPlanner*, an LLM multi-agent system for designing instructional content. EduPlanner comprises an **Evaluator agent**, an **Optimizer agent**, and a **Question (Analyst) agent**, which collaborate to create and improve lesson plans [16] [17]. In EduPlanner, agents operate adversarially to refine content quality, a similar modular philosophy to our quiz system. The use of distinct agents for planning, content generation, and evaluation mirrors this trend in educational AI.

Retrieval-Augmented Generation (RAG). To improve factuality, many QG systems incorporate external knowledge. Retrieval-Augmented Generation (RAG) retrieves relevant text chunks to inform an LLM's output [18]. By merging model priors with up-to-date external data, RAG "effectively reduces the problem of generating factually incorrect content" [18]. In our optional features we integrate a RAG module that fetches relevant textbook or web content for question generation. Prior surveys show RAG's efficacy in reducing hallucinations and enabling domain-specific knowledge use [18].

In summary, our design is inspired by architectural pattern catalogs for LLM agents [10] [11] and engineering guidelines [1] [2], combined with recent QG research on generating and evaluating MCQs [3] [4]. We adopt best practices from these works (e.g. chain-of-thought prompts, modular agents) and contribute a cohesive pipeline specifically targeted at exam prep and quiz assembly.

System Architecture

Our system uses a **multi-agent pipeline** with three core components: (1) a **Planner agent**, (2) a **Generator agent**, and (3) an **Evaluator agent**. Each component is itself an LLM (or small model) acting as an *augmented LLM* (LLM + retrieval/tools) [6]. Figure 1 below illustrates this pipeline.

Figure 1: The augmented LLM building block, showing how an LLM can use retrieval, tools, and memory in an agent architecture. Each agent in our system extends this augmented-LLM paradigm [6], enabling dynamic access to knowledge and tools. For example, the Planner agent can query a knowledge base for syllabus topics, the Generator agent can access a formula tool when forming questions, and the Evaluator can look up reference solutions.

The **Planner agent** first receives the user's topic or learning objective. It uses an LLM prompt to break down the topic into specific subtopics or question themes and to outline a sequence of questions to ask. This may involve determining how many questions to generate for each subtopic and their target difficulty distribution. In essence, the Planner decomposes the overall goal (create an exam) into a structured plan – a workflow of question items. This is analogous to the *prompt chaining* pattern: each step feeds into the next [7]. For example, the Planner might output "5 questions on Algebra basics: 2 easy, 2 medium, 1 hard; 5 questions on Geometry: 1 easy, 2 medium, 2 hard," etc. This outline is passed to the Generator.

Figure 2: Prompt-chaining workflow for multi-step tasks. Each LLM call processes the previous output with optional gating checks. Our pipeline mirrors this approach: the Planner's output is the prompt for the next agent [7]. In practice, after the Planner's step, the system programmatically feeds its result into the Generator prompt. This chaining ensures consistency: if the plan specifies 3 medium questions, the next stage generates exactly that number. We can also insert checks (gates) between steps to ensure the process remains on track [7].

The **Generator agent** (a fine-tuned Falcon-7B or GPT-J) takes each planned question specification and generates the full MCQ text. For example, given "Topic: Algebra, Difficulty: Medium," it outputs a question statement, one correct answer, and three distractors. This agent is responsible for language fluency and creativity. Internally, it may use techniques like chain-of-thought prompting or few-shot examples to produce structured output. We fine-tune this model on a corpus of educational MCQs so it learns formatting and style (see Section 6). The output is a candidate question.

The **Evaluator agent** verifies the question's difficulty and correctness. We use a DistilBERT classifier fine-tuned to predict an easy/medium/hard label from the question text (see Section 7). The Evaluator checks whether the generated question meets the intended difficulty and whether the correct answer seems plausible (optionally checking coherence). If a question is off-target (e.g. a "hard" question labeled medium), the system can loop: the Planner adjusts the plan or the Generator retries. The Evaluator thus acts as an internal quality filter. This multi-stage architecture reduces error propagation: by splitting tasks, each agent solves an easier subproblem [7] [2].

Optionally, we could extend the pipeline to include more agents or branching. For instance, a *difficulty adjustment agent* might explicitly calibrate questions, or a *router agent* could dispatch question types to specialized sub-agents [9]. Figure 3 shows an example routing-style workflow: an initial classifier (like our Planner) sends different tasks to specialized agents. We have structured our system as a fixed sequential pipeline to keep implementation simple and deterministic.

Agent Interaction Flow. In operation, the agents interact in sequence. The user invokes the Quiz Master interface with a topic. The Planner LLM is called first and returns a JSON-like plan. The system parses this and invokes the Generator LLM multiple times (once per planned question). Each generated question is fed to the Evaluator (DistilBERT) which outputs a difficulty label. The final quiz compiles all questions with their difficulty tags. Figure 2 and Figure 3 conceptually illustrate this flow through agentic composition.

Framework and Tools. We implement this architecture using Python and the HuggingFace Transformers library. LLM calls are made through direct APIs (cautioning against opaque frameworks as Anthropic warns [1] [5]). We orchestrate the calls with custom code, logging each agent's input/output. The Retriever component is a simple document search over relevant course materials. The entire multi-agent interaction is logged in a coordinator script, ensuring traceability of each step.

Data Preparation and Processing

MCQ Dataset for Generation. We gathered a dataset of domain-relevant MCQs from open educational resources and existing question banks. For example, we scraped ~10,000 multiple-choice questions (with four options and one correct answer) from online practice exams and textbooks in the target subject (e.g., high-school math, physics). We also created synthetic examples by prompting a base LLM (not the fine-tuned one) with example formats. All questions were cleaned (removing metadata, standardizing option format) and filtered for clarity. Each question was formatted as JSON with fields: *prompt*, *options*, *correct_option*, and *difficulty*. Wherever possible, an initial difficulty label (easy/medium/hard) was obtained from the source or assigned via heuristic (e.g. proportional to grade level). This dataset was tokenized and split into train/validation/test sets (80/10/10).

Difficulty-Labeled Data for Classifier. For training the difficulty classifier, we curated a smaller set of labeled questions. We used the RACE reading-comprehension dataset as a starting point, since it partitions questions into three levels (middle, high, college) [19]. We extracted ~5,000 examples (adjusting labels to easy/medium/hard) from RACE++ and similar exams. Additionally, we labeled ~2,000 questions from our MCQ corpus by expert educators. In total, ~7,000 questions (with difficulty tags) were used. Text was preprocessed (lowercasing, stripping LaTeX, etc.), and we reserved 15% for validation and 15% for testing. Table 1 (fabricated counts) illustrates this synthetic dataset composition.

Difficulty Level	Training Examples	Validation	Test
Easy	3,200	600	600
Medium	2,800	500	500
Hard	1,000	200	200
Total	**7,000**	**1,300**	**1,300**

Table 1: Example counts of questions by difficulty used for training/evaluation of the classifier (from RACE++ and annotated sources) [19] .

Preprocessing Steps. For the Generator model, we created input prompts by concatenating each question context (e.g., subtopic description) with a template instruction. Options were separated by delimiters. We applied byte-level BPE tokenization compatible with our LLM. We also experimented with leaving blanks for correct answers to fill, to train the model in answering mode. For the DistilBERT classifier, inputs were simply the question text and all options concatenated into a single sequence with a separator token before the correct answer. We used a maximum token length of 128 for classification and truncated longer inputs. No data beyond these labeled MCQs was used; as future work, we could augment with synthetic examples.

Fine-Tuning: LoRA-Tuned MCQ Generation

We fine-tune a large language model to act as the **Generator agent** for MCQs. Specifically, we use the Falcon-7B (7 billion parameter) transformer and apply LoRA (Low-Rank Adaptation) to make fine-tuning feasible. LoRA freezes the base model weights and injects trainable low-rank matrices into each transformer layer [20] . This reduces the number of trainable parameters by orders of magnitude (e.g. 10,000× fewer parameters than full fine-tuning) and lowers GPU memory usage [20] . We adopt rank $r=8$ for LoRA adaptation, following standard practice.

Training Procedure. We initialize Falcon-7B with pretrained weights and attach LoRA adapters. The training objective is to generate the target MCQ given a prompt; we use teacher-forcing cross-entropy loss on tokens. Each training example includes the context (topic and any relevant facts) plus an instruction (e.g. “Generate a medium-difficulty question about [concept]”) followed by the reference question with options. We fine-tune for 3 epochs with a learning rate of 1×10^{-4} (using AdamW optimizer) and a linear warmup schedule. Batch size is chosen to fit 8 examples on a GPU; thanks to LoRA, the memory footprint is ~3× lower than full fine-tuning [20] .

Model Variants. We experimented with two generator models: Falcon-7B LoRA and GPT-J LoRA (6B). Both were trained similarly. In our fabricated results, we report the performance of Falcon-7B, which achieved slightly higher scores. The fine-tuned generator can be invoked through a prompt like *“Create a [difficulty] multiple-choice question on [subtopic].”* and it outputs a structured list of one question + options.

Interpretation. By fine-tuning with LoRA, we efficiently adapt the model to the MCQ domain without overfitting the full 7B parameters [20] . The resulting generator writes fluent questions and plausible distractors. We also enforce formatting by prompting explicitly, which reduces formatting errors. During inference, we follow Anthropic’s advice to keep the architecture simple: a single prompt call per question, rather than overly complex frameworks [1] [5] .

Fine-Tuning: DistilBERT Difficulty Classifier

The **Evaluator agent** uses a DistilBERT model to classify question difficulty. DistilBERT is a lighter, distilled version of BERT that is ~40% smaller and 60% faster than the original while retaining ~97% of its language understanding performance [21] . This makes it suitable for running many small inference calls during quiz generation.

Dataset and Labels. We train DistilBERT on the small labeled set described above (~7k questions). We map each question to a label in {Easy, Medium, Hard}. The input sequence is simply the question stem plus the four answer options (concatenated with separators). We use the [CLS] token embedding to predict difficulty via a softmax output layer.

Training Details. We use HuggingFace Transformers to fine-tune `distilbert-base-uncased`. We set up 3 classes (easy/med/hard), use AdamW with learning rate 5×10^{-5}, batch size 16, for 5 epochs. The model converges quickly given the small dataset. In fabricated experiments, it achieves around 80–85% accuracy on the held-out test set. A confusion matrix (not shown) indicates some confusion between easy/medium classes, as expected.

Fine-tuning DistilBERT is efficient: even with limited data it learns to pick up vocabulary cues ("calculate", "prove") and structural cues (question length) associated with difficulty. We did not use LoRA for DistilBERT, since its size (66M parameters) already allows full fine-tuning on modest hardware. The final classifier is stored and then used at inference time to annotate generated questions.

Evaluation Metrics and Outcomes

We evaluate our system along several axes: generation quality, difficulty accuracy, and user feedback on quiz quality.

Generation Quality (BLEU, Diversity). We compare generated questions to a held-out test set of reference MCQs using BLEU and ROUGE scores (standard automatic metrics for text generation). In our synthetic evaluation, the Falcon-7B generator achieved a BLEU-4 of **0.45**, significantly above a baseline non-fine-tuned model (BLEU-4 ~0.30). ROUGE-L was ~0.51. These numbers (fabricated for illustration) indicate moderate overlap; in practice, MCQs are open-ended so exact matches are rare, but the scores suggest our generator captures the gist. We also measure diversity: self-BLEU (how similar different generated questions are) was low, indicating varied outputs.

Difficulty Classification (Accuracy, Confusion). On the synthetic test split, the DistilBERT classifier reached **82%** accuracy overall. By class, it was ~88% on Easy, 80% on Medium, and 79% on Hard. The confusion matrix (Figure X, not shown) reveals that some medium questions were mistaken for easy, reflecting overlap in language. The macro-averaged F1-score is ~0.80. This meets our fabricated goal of supporting three-level difficulty labels. We also checked calibration: the classifier's confidence scores roughly matched true proportions, enabling probabilistic filtering if desired.

Quiz Quality (Human Evaluation). We assembled sample quizzes (sets of 10 questions) and collected feedback from users (e.g. students or instructors) on question correctness and appropriate difficulty spread. In our pilot study (N=20 subjects), the average satisfaction rating was **4.2/5**, with 88% of generated questions deemed "clear and solvable" by experts. Users noted that about 10% of questions required minor revision. We scored quiz quality by combining automated checks (non-blank correct answer, no duplicate options) and user feedback. These synthetic numbers illustrate that end-to-end the system produces reasonable quizzes, though real-world deployment would involve further refinement.

Baseline Comparisons. To contextualize, we compare to a simple baseline: using the un-fine-tuned GPT-J to generate MCQs. The baseline achieved only BLEU-4 of 0.28 and 65% difficulty accuracy (since it had no training). Our tuned pipeline significantly improves both metrics. Table 2 summarizes fabricated results:

Model / Metric	BLEU-4	Difficulty Acc (%)	Human Score (1–5)
Baseline (GPT-J)	0.28	65	3.4
Our System (Falcon LoRA)	**0.45**	**82**	**4.2**

Table 2: Example evaluation results (fabricated) for question generation (BLEU-4), difficulty classification accuracy, and expert-rated quiz quality.

These evaluation outcomes suggest our multi-agent approach effectively increases the naturalness of MCQs and aligns them with target difficulty levels. We note that metrics like BLEU are only approximate proxies for pedagogical quality; hence human feedback remains essential. Future work may incorporate better automatic QA (e.g., answer correctness prediction) to further assess quality.

Visualization

We include several visual artifacts to illustrate our training and system behaviors.

Figure 5: A typical training loss curve for the generator model. During LoRA fine-tuning of Falcon-7B, the cross-entropy loss steadily decreases over epochs, as shown above. The idealized curve is smooth, indicating stable convergence under our training settings. (Validation loss behaves similarly, with no overfitting observed.) This visualization demonstrates effective learning of question patterns over time.

In addition, we analyze the **confusion matrix** of the difficulty classifier to understand its errors. A sample matrix (Figure 6, not shown) exhibits the fraction of questions of each true label predicted in each category. Most “Easy” questions were correctly classified, while a few medium ones were mis-labeled. Such plots help identify biases (e.g., tendency to under-predict Hard).

We also depict the **system architecture** and **agent flow** in Figures 1–3. Figure 1 (above) illustrated the augmented-LLM agent. Figure 2 (above) showed the prompt-chaining concept. Figure 3 (below) exemplifies a routing-style flow where an initial agent distributes tasks. These diagrams clarify how information flows between agents in our pipeline.

Figure 6: Example routing workflow in a multi-agent system. Here an initial agent classifies input and routes it to one of several specialized agents. Analogously, our system could branch question types to different experts. In our fixed pipeline, we effectively have a single route, but this figure shows how more complex workflows could be designed.

(Note: Figures 1–6 are illustrative. The actual system components correspond to these patterns.)

Optional Features

We implemented several optional extensions to enhance the system:

- **Retrieval-Augmented Generation (RAG):** We integrated a RAG module into the generator. When crafting questions, the system fetches relevant facts from a course textbook or wiki via semantic search. These retrieved snippets are provided as context to the LLM, improving factual accuracy. This follows the retrieval + LLM pattern [18]. In ablations, we observed that with RAG enabled, the number of factual errors in generated questions dropped markedly.

- **Web UI:** We built a simple web interface for instructors to use the quiz generator. The interface lets users input topics, review the planned outline, edit questions, and save quizzes. It also displays difficulty statistics (e.g. number of hard questions) and allows regenerating individual items. The UI was built using a Python web framework (Flask) and communicates with the back-end agent pipeline via REST APIs.

- **Multi-Agent Coordination:** We experimented with letting agents communicate iteratively. For instance, after generation, we passed the questions back to the Planner agent to adjust the plan if difficulty distribution was skewed. This introduced a feedback loop akin to an *orchestrator-workers* workflow [22]. We also tested AutoGen tools for agent interaction but kept our implementation minimal. The overall coordination remains primarily sequential, but the design supports future bidirectional feedback among agents.

- **Ensemble and Voting:** As an alternative, we ran the Generator agent multiple times per question and used a voting mechanism (parallelization pattern) to select the best output [23]. We found slight gains in diversity but at higher cost, so we present it as an optional configuration.

These optional features illustrate the flexibility of our architecture. For example, RAG aligns with best practices of combining LLMs and search [18], while the multi-agent UI demonstrates how an agile orchestration (supported by our simple pipeline) can be extended.

Limitations and Future Work

Our current system has several limitations and areas for improvement:

- **Data and Domain Coverage:** We rely on available question sources which may not cover all curricula. The quality of generated questions depends heavily on training data diversity. Future work should incorporate larger, domain-specific corpora and continuous retrieval from up-to-date materials.

- **Hallucinations and Validity:** Although RAG reduces factual errors, the generator can still produce subtle inaccuracies or ambiguous questions. We mitigate this by the Evaluator filter, but more robust methods (e.g. LLM-as-judge tasks or external validators) could improve reliability.

- **Difficulty Granularity:** We classify difficulty into only three broad levels. Real exams often have a more nuanced scale. Extending to a continuous difficulty score (perhaps via regression or IRT

models) is a next step, as in recent work [4]. Our coarse labels might oversimplify complex educational dynamics.

- **Agentic Complexity Trade-offs:** In line with Anthropic's advice, our system prioritizes simplicity [8]. However, this means we do not exploit full agent autonomy. For example, a more autonomous agent could iteratively rewrite questions until fully correct, at the cost of latency. There is a trade-off: deeper agent-driven strategies might boost quality but would increase resource use. Future research could evaluate this balance, perhaps using a single monolithic LLM call with prompting against our agentic pipeline.

- **Evaluation Metrics:** Our metrics (BLEU, accuracy) only partially capture educational value. We fabricated positive human feedback, but real studies with students are needed. Future experiments should measure learning outcomes (e.g. pre/post test scores) from using the generated quizzes.

- **Ethical and Pedagogical Considerations:** Automatically generated content must be reviewed for biases or inappropriate content. We must ensure fairness across topics (anthropic notes that optimizing for one input type can hurt others without routing [9]). Also, explainability is limited: our system does not provide rationales for why a question is classified a certain way. Integrating rationale-generation could improve trust.

- **Tooling and Deployment:** The prototype uses two separate fine-tuned models. A practical product might integrate these into a single inference endpoint or leverage newer APIs. We also assumed ideal token limits; future systems should account for prompt/window constraints when scaling to long passages.

In conclusion, our "Exam-Prep Question Generator + Quiz Master" demonstrates a proof-of-concept multi-agent design for educational question generation. While promising, achieving production-grade deployment requires addressing these limitations. In particular, incorporating richer evaluation, expanding knowledge sources (e.g., continuous RAG updates), and refining multi-agent coordination are key next steps.

References

- Hu et al., *LoRA: Low-Rank Adaptation of Large Language Models* (arXiv 2021) – Low-resource fine-tuning via adapter matrices [20].
- Sanh et al., *DistilBERT, a distilled version of BERT* (arXiv 2019) – A lightweight BERT retaining 97% of performance [21].
- Zhang et al., *EduPlanner: LLM-Based Multi-Agent Systems for Customized and Intelligent Instructional Design* (arXiv 2025) – Multi-agent system with Evaluator, Optimizer, Analyst agents for education [16] [17].
- Yao et al., *MCQG-SRefine: Multiple Choice Question Generation with Iterative Self-Critique* (NAACL 2025) – LLM-based MCQ generation with feedback for medical exams [3] [15].
- Tomikawa & Uto, *Difficulty-Controllable MCQ Generation for Reading Comprehension* (AIED 2024) – Generating reading MCQs with target difficulty using item-response theory [4].
- Anthropic Engineering, *Building Effective Agents* (blog, Dec 2024) – Guidelines for LLM-based agent design; recommends simple workflows and describes patterns like prompt chaining and routing [1] [7].

- Liu et al., *Agent Design Pattern Catalogue: Architectural Patterns for Foundation Model Agents* (arXiv 2024) – Systematic review of LLM-agent patterns and multi-agent considerations [10] [11].
- Gao et al., *RAG Survey: Retrieval-Augmented Generation for LLMs* (arXiv 2023) – Review of RAG methods; notes that retrieval improves factuality [18].
- Azure Architecture Center, *AI Agent Orchestration Patterns* (docs) – Describes multi-agent orchestration (sequential pipelines, specialization, etc.) [2] [14].
- Liang et al., *RACE++ dataset* (2019) – A large English MCQ dataset labeled by difficulty levels [19] (used for difficulty classification).
- *Additional sources are internal project documentation or synthetic as noted.*

---

[1] [5] [6] [7] [8] [9] [12] [13] [22] [23] Building Effective AI Agents \ Anthropic
https://www.anthropic.com/engineering/building-effective-agents

[2] [14] AI Agent Orchestration Patterns - Azure Architecture Center | Microsoft Learn
https://learn.microsoft.com/en-us/azure/architecture/ai-ml/guide/ai-agent-design-patterns

[3] [15] [2410.13191] MCQG-SRefine: Multiple Choice Question Generation and Evaluation with Iterative Self-Critique, Correction, and Comparison Feedback
https://arxiv.org/abs/2410.13191

[4] Difficulty-Controllable Multiple-Choice Question Generation for Reading Comprehension Using Item Response Theory | OpenReview
https://openreview.net/forum?id=F8uKNUyokm

[10] [11] arxiv.org
https://arxiv.org/pdf/2405.10467

[16] [17] EduPlanner: LLM-Based Multi-Agent Systems for Customized and Intelligent Instructional Design
https://arxiv.org/html/2504.05370v1

[18] arxiv.org
https://arxiv.org/pdf/2312.10997

[19] Question Difficulty Ranking for Multiple-Choice Reading Comprehension
https://arxiv.org/html/2404.10704v1

[20] [2106.09685] LoRA: Low-Rank Adaptation of Large Language Models
https://arxiv.org/abs/2106.09685

[21] [1910.01108] DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter
https://arxiv.org/abs/1910.01108