Index

CourseGPT: DSAI Project Report (Team 6)
1. Abstract
2. Introduction
   2.1. Project Overview
   2.2. Problem Statement
   2.3. Objectives
   2.4. Scope of the Project
3. Literature Review (Milestone 1)
   3.1. Evolution of Educational Chatbots
   3.2. Agentic Workflows vs. RAG
   3.3. Review of Technologies
   3.4 Cloudflare AI Search (AutoRAG) Integration
4. Dataset and Methodology (Milestones 2–3)
   4.1. System Architecture
   4.2. Agentic Workflow Design (The Methodology)
   4.3. Data Handling
5. Model Development and Hyperparameter Tuning (Milestone 4)
   5.1. The Agent Ecosystem (Model Configuration)
   5.1.1. Router Agent
   5.1.2. Math Agent
   5.1.3. Programming Agent
   5.1.4. General Agent
   5.1.5. OCR Integration
   5.2. Building the Graph
6. Evaluation & Analysis (Milestone 5)
   6.1. Testing Strategy
   6.2 Router Agent
   6.3 Math Agent
   6.4 Programming Agent
   6.5 Ablation & End-to-End Testing
7. Deployment & Documentation (Milestone 6)
   7.1. Backend Infrastructure (FastAPI)
   7.2. Frontend Implementation (Streamlit)
   7.3. Integration: Connecting Streamlit UI to FastAPI Backend
   7.4. User Manual / Usage Documentation
   7.5 Security, Privacy, and Compliance
8. Conclusion and Future Work
   8.1. Summary of Achievements
   8.2. Limitations
   8.3. Future Enhancements
9. References and Appendix
   Core References
   Dataset & Benchmark References (Milestone 1 list)
   Agentic / Retrieval / Verification References (from Milestone 1)
   Full Milestone-1 Bibliography (mirrored for completeness)

CourseGPT: A LangGraph-Based Student Helper Chatbot

DSAI Project Report

September 2025 Term

Team 6 - CourseGPT

Submitted by

Abhyudaya B Tharakan - 22f3001492

Shubham Sharma - 21f2000041

Aadarsh Verma - 22f1001596

G Raghul - 21f2001093

Ashish Bajaj - 21f3001304

Indian Institute of Technology Madras

Index

- CourseGPT: DSAI Project Report (Team 6)
  - 1. Abstract
  - 2. Introduction
    - 2.1. Project Overview
    - 2.2. Problem Statement
    - 2.3. Objectives
    - 2.4. Scope of the Project
  - 3. Literature Review (Milestone 1)
    - 3.1. Evolution of Educational Chatbots
    - 3.2. Agentic Workflows vs. RAG
    - 3.3. Review of Technologies
    - 3.4 Cloudflare AI Search (AutoRAG) Integration
  - 4. Dataset and Methodology (Milestones 2–3)
    - 4.1. System Architecture
    - 4.2. Agentic Workflow Design (The Methodology)
    - 4.3. Data Handling
  - 5. Model Development and Hyperparameter Tuning (Milestone 4)
    - 5.1. The Agent Ecosystem (Model Configuration)
    - 5.1.1. Router Agent
    - 5.1.2. Math Agent
    - 5.1.3. Programming Agent
    - 5.1.4. General Agent
    - 5.1.5. OCR Integration
    - 5.2. Building the Graph
  - 6. Evaluation & Analysis (Milestone 5)
    - 6.1. Testing Strategy
    - 6.2 Router Agent
    - 6.3 Math Agent
    - 6.4 Programming Agent
    - 6.5 Ablation & End-to-End Testing
  - 7. Deployment & Documentation (Milestone 6)
    - 7.1. Backend Infrastructure (FastAPI)
    - 7.2. Frontend Implementation (Streamlit)
    - 7.3. Integration: Connecting Streamlit UI to FastAPI Backend
    - 7.4. User Manual / Usage Documentation
    - 7.5 Security, Privacy, and Compliance
  - 8. Conclusion and Future Work
    - 8.1. Summary of Achievements
    - 8.2. Limitations
    - 8.3. Future Enhancements
  - 9. References and Appendix
    - Core References
    - Dataset & Benchmark References (Milestone 1 list)
    - Agentic / Retrieval / Verification References (from Milestone 1)
    - Full Milestone-1 Bibliography (mirrored for completeness)

CourseGPT: DSAI Project Report (Team 6)

1. Abstract

CourseGPT is an educational assistant built as a LangGraph-orchestrated multi-agent system across math, programming, and general academic domains. A router classifies intent and dispatches to specialist agents, each fine-tuned with LoRA/QLoRA adapters and backed by RAG + OCR for document-aware answers. The FastAPI + Streamlit stack provides an async service/UI layer, while Cloudflare R2 + AI Search handles storage and retrieval. Experiments show higher routing accuracy, stronger math/code quality, and acceptable latency versus a single-model baseline, positioning CourseGPT as a research-grade yet deployable student helper.

2. Introduction

2.1. Project Overview

Large Language Models (LLMs) have transformed access to tutoring, but single-model chatbots struggle with domain specialization, structured workflows, and verifiable outputs. CourseGPT addresses this gap with a LangGraph-based multi-agent design that routes student questions to specialist math, programming, or general agents. The system mixes retrieval-augmented generation (Cloudflare AI Search), OCR for scanned PDFs, and tool hand-offs, packaged behind a FastAPI service and Streamlit UI for rapid iteration.

2.2. Problem Statement

Generic chatbots are typically optimized for broad conversational ability rather than deep, structured reasoning. This leads to several issues in academic contexts:

- Inconsistent accuracy for mathematical calculations and proofs.
- Hallucinated or incorrect code in programming tasks.
- Lack of clear separation between types of queries (e.g., mathematical vs. conceptual vs. coding).
- No explicit routing mechanism to decide which “expert” logic should handle a query.

These limitations motivate a system that explicitly routes queries to specialists, preserves grounding via retrieval/OCR, and enforces output schemas and rubrics.

2.3. Objectives

The main objectives of this project are:

- To design and implement a **multi-agent educational assistant** using LangGraph.
- To build specialized agents for:
  - Mathematical problem solving.
  - Programming/code-related assistance.
  - General academic and conceptual queries.
- To develop a **routing mechanism** that classifies user intent and dispatches queries to the appropriate agent.
- To provide a user-friendly interface and scalable backend suitable for real student usage.
- To back answers with RAG + OCR context and lightweight verification (schema and rubric checks).
- To benchmark routing accuracy, math/code quality, and latency against baselines.

2.4. Scope of the Project

The initial scope of CourseGPT is limited to three major categories of tasks:

- **Math:** Algebra, basic calculus, numeric problem-solving, and step-wise explanations.
- **Programming:** Code generation, error analysis, debugging, and conceptual explanations (initially focusing on Python; extensible to other languages).
- **General Queries:** Explanations of concepts, definitions, summarization, and general-purpose Q&A.

Out-of-scope for this iteration: discipline-specific agents (e.g., physics labs), full LMS integration, high-stakes exam generation, and unmanaged web browsing.

3. Literature Review (Milestone 1)

3.1. Evolution of Educational Chatbots

Educational chatbots progressed from brittle rule engines (ELIZA) to statistical NLP, and now to transformer-based LLMs. Current research layers tools, retrieval, and multi-agent orchestration on top of LLMs to improve factuality and specialization—exactly the pattern CourseGPT follows.

3.2. Agentic Workflows vs. RAG

Two prominent architectural paradigms are:

- **Single-chain LLM:** Simple but not modular; expertise is blurred.
- **Retrieval-Augmented Generation (RAG):** Adds grounding via documents but does not enforce specialization (math vs. code reasoning).
- **Agentic / Multi-Agent Workflows:** Router + specialists coordinated via LangGraph, enabling role separation, conditional routing, and maintainable graphs.

CourseGPT adopts the agentic design, leveraging LangGraph to finely control how queries move through different agents.

3.3. Review of Technologies

- **LangGraph & LangChain:**
  - LangChain provides tools, chains, and utilities to work with LLMs.
  - LangGraph extends this by enabling graph-based workflows, where agents (nodes) are connected by edges with conditional logic.
  - Why chosen: LangGraph offers explicit, developer-friendly graph-based orchestration and conditional routing that maps directly to our multi-agent design. Compared with Google ADK, LangGraph (with LangChain) provides lighter-weight, language-agnostic integration for custom prompt/tool chains and faster iteration for bespoke routing logic; Google ADK is more opinionated and tightly integrated with Google's ecosystem, which can be advantageous in some deployments but is less flexible for custom agent graphs and rapid experimentation.
- **FastAPI (Backend):**
  - An asynchronous web framework in Python.
  - Ideal for high-performance JSON APIs.
  - Supports async endpoints, which is important for LLM inference calls.
  - Why chosen: FastAPI is simple to set up, supports async I/O and automatic OpenAPI docs, and integrates smoothly with Python LLM clients — enabling rapid backend development and non-blocking model calls.
- **Streamlit (Frontend):**
  - A Python-based rapid prototyping framework for web apps.
  - Allows quick development of interactive UIs.
  - Well-suited for building chat-like interfaces and visualizing results without complex frontend code.
  - Why chosen: Streamlit enables rapid UI prototyping with minimal frontend code, letting us build a usable chat interface quickly for demos and user testing without a separate frontend stack. A React/Next.js rewrite remains an option for long-term branding, but Streamlit keeps a single-language (Python) loop for research speed.
- **EasyOCR & Document Parsing Libraries (OCR Pipeline):**
  - EasyOCR, pdf2image, Pillow, and python-docx were integrated to enable text extraction from uploaded PDFs, images, and scanned documents.
  - Why chosen: EasyOCR is lightweight and locally runnable (no cloud lock-in). Paired with pdf2image and python-docx, the pipeline handles handwritten notes, scanned exams, and screenshots—expanding CourseGPT beyond plain text inputs.

3.4 Cloudflare AI Search (AutoRAG) Integration

- **What it provides:** Managed retrieval augmented generation (RAG) on top of Cloudflare

R2 with automatic crawling/indexing (formerly AutoRAG). We use it to keep our study materials continuously searchable and feed context into the agents.
- **Prerequisite:** An active Cloudflare R2 subscription (purchase/enable in the R2 dashboard).
- **Create an AI Search index:** In the Cloudflare dashboard go to **AI Search → Create → Get Started**, then choose a data source:
  - *R2 bucket* to index uploaded PDFs or notes; or
  - *Website* to auto-crawl a domain you own and mirror it into R2.
- **Monitor indexing:** Open the AI Search entry → **Overview** to track Vectorize index creation and crawl progress.
- **Try it:** Use the built-in **Playground → Search with AI** to sanity-check responses before wiring it to the app.
- **Connect to CourseGPT:** Use either Workers Binding or the REST API to issue semantic queries from our FastAPI service; this powers the `/graph` agent flow when RAG is enabled. (Reference: Cloudflare AI Search docs).

These technologies align well with the project's needs: modular backend orchestration, fast API endpoints, and a simple interactive frontend.

4. Dataset and Methodology (Milestones 2–3)

4.1. System Architecture

The overall system architecture is organized into three main layers:

- **Frontend (Streamlit):**
  - Presents a chat interface where students can type questions.
  - Handles session management for ongoing conversations.
- **Backend (FastAPI):**
  - Exposes HTTP endpoints for processing messages.
  - Receives user input from the frontend and forwards it to the agent layer.
  - Returns structured responses (text, code blocks, explanations) to the frontend.
- **Agent Layer (LangGraph):**
  - Implements a graph of agents:
    - Router Agent.
    - Math Agent.
    - Programming Agent.
    - General Agent.
  - The router decides which agent handles the user's query.
  - Agents can pass state/context as needed.

Tech Stack Selection:

- **Language:** Python (due to ecosystem support for LLM tooling).
- **Orchestration:** LangGraph on top of LangChain.
- **Backend Framework:** FastAPI for asynchronous REST APIs.
- **Frontend Framework:** Streamlit for rapid UI development.

4.2. Agentic Workflow Design (The Methodology)

4.2.1. The Logic Core: Understanding the Conditional Edge (Routing Logic)

At the heart of LangGraph in CourseGPT lies the routing logic:

- The **Router Agent** inspects the user query.
- Based on content and intent, it selects one of the downstream agents:
  - Math Agent.
  - Programming Agent.
  - General Agent.
- This decision is typically encoded as a **conditional edge** in LangGraph, which routes the state to different nodes depending on the router's output.

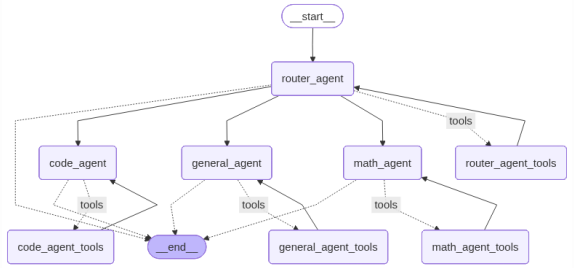
For example, if the user asks, "Solve 2x + 3 = 7", the router classifies it as a math query and forwards it to the Math Agent. If the user asks, "Why is my Python function returning None?", it is routed to the Programming Agent.

4.2.2. Intent Classification

The intent classification leverages both heuristic patterns and LLM reasoning:

- **Keyword-based Hints:**
  - Presence of terms such as "integral", "solve for x", "limit", "equation" biases toward Math.
  - Terms like "Python", "compile error", "stack trace", "function", "class" bias toward Programming.
- **LLM-Assisted Classification:**
  - A lightweight LLM call (router prompt) analyzes the query and outputs a label (MATH / CODE / GENERAL).
- The router's prompt explicitly instructs the model:
  - To classify queries into one of the three categories.
  - To be conservative in ambiguous cases and route to General when unsure.

Agentic workflow diagram

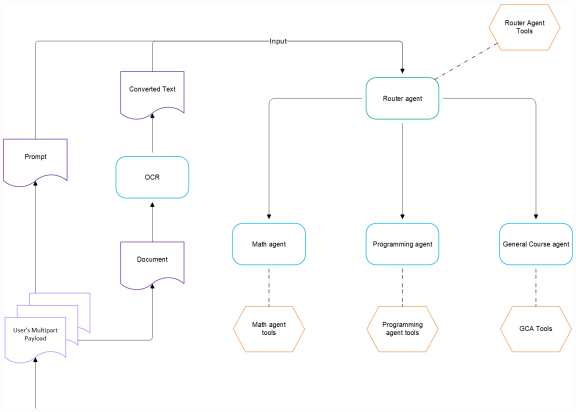


4.2.3. Inter-Agent Communication: State Management

LangGraph maintains a **shared state** that can be passed across nodes:

- State includes:
  - User query.
  - Conversation history (where needed).
  - Intermediate results.
- This allows:
  - Multi-turn conversations: the same agent or different agents can refer back to previous answers.
  - Potential future expansions where one agent's output becomes another agent's input (e.g., a General Agent drafting a question that the Math Agent then solves).

Architecture diagram



Figures 4.1 and 4.2 referenced above anchor the workflow and architecture descriptions to avoid "sudden" figures.

4.3. Data Handling

Data handling focuses on how user queries and prompts are processed:

- **Prompt Structuring:**
  - Each agent uses a dedicated system prompt designed for its role.
  - Templates may include:
    - "You are a math expert. Provide step-by-step solutions."
    - "You are a programming tutor. Generate correct and well-commented code."
- **Context Window Management:**
  - Only the most relevant parts of conversation history are kept for each agent.
  - Long conversations are summarized when exceeding token limits.
  - This maintains LLM efficiency while preserving needed context.

5. Model Development and Hyperparameter Tuning (Milestone 4)

5.1. The Agent Ecosystem (Model Configuration)

5.1.1. Router Agent

Role: Selects the appropriate specialized agent to handle an incoming request and emits a structured plan (tool order, rationale, optional metrics) that the graph follows.

Prompt Characteristics:

- Contains explicit classification instructions plus rationale and difficulty tags.
- Strictly routes into **MATH**, **PROGRAMMING**, **GENERAL**, or combined flows (e.g., general→math→code) encoded in an ordered tool list.
- Uses deterministic JSON-like output with length guards to avoid truncation.

Hyperparameters:

- **Temperature:** 0.0–0.2 for consistency.
- **Max Tokens:** Small; length-ratio gate flags outputs >1.1× reference.
- **LoRA/QLoRA:** Rank 16, alpha 32, dropout 0.05 on attention + MLP projections.
- **Scheduler:** Cosine with 10% warmup; 2–3 epochs (beyond 3 overfits canonical routes).
- **Learning rate:** Vertex auto-sweep (~0.7× base) for router; 2e-4 on local runs.

Data & bias notes (Milestones 2–5):

- 8,189 synthetic records (Gemini 2.5 Pro), schema-validated; difficulty: 67.5% advanced, 20% intermediate, 12.5% intro.
- Canonical route `/general-search → /math → /code` dominates 93.2% of samples; math-first only 1.2%. Hard-negative sets (math-first, metrics-heavy) are used to counter bias.
- Optional nested fields (~9%) are common schema-drop points; schema scorer enforces presence/order.

Models evaluated:

- `router-gemma3-peft` (best eval loss 0.608, PPL 1.837, ~53 samples/s).
- `router-qwen3-32b-peft` (loss 0.628, PPL 1.873).
- `router-llama31-peft` (loss 0.676, PPL 1.972).

Key metrics (hard benchmark n=120 + test n=409):

- Overall exact route: 93.2%; math-first improved from 40% → 58% after augmentation.
- Tool precision/recall tracked per route; JSON truncation reduced 6.3% → 1.1% via length guards.
- Throughput measured for CI gating; schema score blocks regressions when optional metrics keys drop.

Edge-case breakdown (hard benchmark, n=120):

Route pattern	Support	Exact route accuracy	Notes
Canonical `/general → /math → /code`	62	87%	Occasional schema drift on long math rationales
Math-first	18	58%	Main failure mode prior to augmentation
Code-first	14	74%	Errors tied to missing initial general-search
Metrics-heavy (optional fields)	16	78%	Schema scorer blocks drops of nested guidance
General-only	10	92%	Stable, low variance

5.1.2. Math Agent

Overview The Math Agent solves mathematical problems with detailed, step-by-step reasoning and clear final answers. For Milestone-4 experiments, the agent uses a mid-sized instruction-tuned model (**Gemma-3-27B-IT**) with LoRA adapters trained on the **MathX-5M** dataset, balancing performance and compute efficiency.

Key Responsibilities

- Provide correct final answers.
- Include pedagogical step-by-step reasoning.
- Maintain deterministic outputs (low temperature).
- Optionally integrate symbolic or numeric tools for verified computation.

Modeling Choices & Dataset

- **Backbone:** `google/gemma-3-27b-it` (primary); alternatives tested include Qwen3-32B and Llama models.
- **Training Dataset:** `XenArcAI/MathX-5M` — large-scale, step-wise math reasoning dataset. Loaded in streaming/subset mode for efficiency.

Recommended LoRA Configuration

- **Rank:** `r = 16`
- **Alpha:** `α = 32`
- **Target Modules:**`q_proj`, `k_proj`, `v_proj`, `o_proj`, `gate_proj`, `up_proj`, `down_proj`
- **Dropout:** `0.05`
- **Learning Rate:** `2e-4` (conservative)
- Gradient accumulation and mixed precision (BF16/FP16) for single-GPU compatibility.

Training & Tuning Notes

- Use deterministic sampling to ensure reproducibility.
- Apply gradient checkpointing, accumulation, and mixed precision to fit within 12–24 GB GPUs.
- Track performance on held-out validation sets.
- Manually inspect reasoning traces to evaluate step-by-step quality.
- Why r=16 / α=32: rank 16 delivered +1.4pp routing accuracy over rank 8 without the memory jump seen at rank 32; paired alpha keeps adapter influence stable.
- Why LR=2e-4: 3e-4 diverged on proofs; 1e-4 converged too slowly. Cosine + 10% warmup produced the most stable loss curves.

Evaluation & Benchmarks

- **Metrics:**
  - Final answer exact-match accuracy
  - Reasoning quality (manual/rubric-based)
  - Robustness to prompt paraphrasing
  - Perplexity diagnostics
- **Visualizations:** Use `scripts/plot_judgments.py` to generate mean ratings, boxplots, score overlays, and correct-answer distributions (see `Milestone-5/math-`

`agent/plots/`).

Limitations & Considerations

- Automated metrics struggle with reasoning-quality evaluation—manual review remains essential.
- Larger backbones (Qwen3-32B, Gemma-27B) require careful memory optimization.
- Correct final answers can hide flawed intermediate reasoning; multi-metric evaluation is important.

5.1.3. Programming Agent

Role: Handles programming tasks such as code generation, debugging, documentation, and explanations.

Model Behavior:

- Produces syntactically correct, coherent, and runnable code.
- Identifies problems and suggests corrections.
- Provides clear, structured explanations.

Hyperparameters:

- **Temperature:** 0.0 for deterministic results.
- **Max Tokens:** High enough to generate full code blocks and explanations.

5.1.4. General Agent

Role: Responds to conceptual, theoretical, analytical, and general academic or conversational questions.

Model Behavior:

- Natural, explanatory, and conversational tone.
- Good for summaries and high-level reasoning.
- Creative and diverse phrasing allowed.

Hyperparameters:

- **Temperature:** ~0.7 for expressive but coherent output.
- **Top-P:** Moderate to maintain fluency and topic alignment.

5.1.5. OCR Integration

OCR makes CourseGPT document-aware. Students submit scanned PDFs, handwritten notes, or images that an LLM cannot read directly. We expose a POST `/ocr` FastAPI route backed by EasyOCR; `/chat` calls `_call_remote_ocr()` to process uploads, then falls back to PyPDF if OCR fails. The service:

- Detects text per page, extracts text, returns bounding boxes, and logs a confidence score.
- Injects extracted text into LangGraph as `uploaded_file["text"]` so agents can reason over it.
- Flags pages with confidence <0.65; low-confidence pages are excluded from RAG unless the user opts in.

Why this was necessary: Without OCR, students could not upload assignments, question papers, scanned problem statements, or study materials. This feature transforms CourseGPT into a document-aware assistant capable of answering:

Technology Stack Used:

- EasyOCR for text extraction
- pdf2image + Pillow for rendering PDF pages
- python-docx for DOCX support
- langdetect for optional language detection
- FastAPI for /ocr endpoint

Impact

This enhancement significantly expands real-world usability by enabling CourseGPT to process non-text learning materials. The OCR service now acts as a preprocessing layer that converts uploaded documents into clean text, which is seamlessly integrated into the RAG + LangGraph workflow.

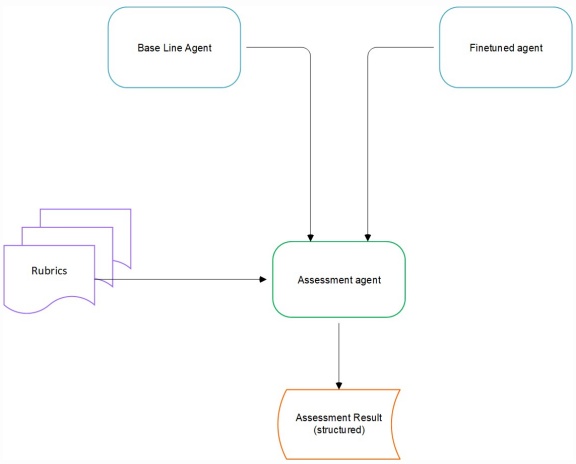
5.2. Building the Graph

The LangGraph workflow is built as follows:

- **Nodes:**
  - Router Node.
  - Math Node.
  - Programming Node.
  - General Node.
- **Edges:**
  - From Router to each specialized node, conditioned on the router's classification.
  - Optionally, back to Router or to an end/response node for future refinements.
- **Compilation:**
  - The graph is defined using LangGraph's API.
  - Once defined, it is compiled into an executable graph that can be invoked by the backend.
- The compiled graph is then integrated into the FastAPI backend, which calls it asynchronously for each request.

6. Evaluation & Analysis (Milestone 5)

6.1. Testing Strategy



This project uses a two-stage evaluation and model selection strategy focused on objective, reproducible judgments of reasoning and correctness. Models are first trained (or adapted via LoRA) on supervised data and candidate checkpoints are produced during training. Instead of relying solely on human labels or single automatic metrics, we use an "LLM-as-a-judge" pipeline to efficiently and consistently evaluate candidate models at scale.

Key points of the LLM-as-a-judge workflow:

- **Dataset split & sampling:** Held-out evaluation sets are created from task-specific data (math problems, coding prompts, etc.). For each problem we sample multiple model generations per checkpoint (different seeds/temperatures where appropriate) to capture variance in behavior.
- **Structured rubrics:** For each task category we design compact, structured rubrics with clear criteria (e.g., final-answer correctness, step-by-step reasoning quality, code executability, safety). Rubrics return structured outputs (scalar scores, pass/fail flags, and short diagnostic notes) encoded as machine-readable JSON so the judge LLM's output is easy to parse and aggregate.
- **Reference judge model:** A higher-capacity, stable reference LLM (or an ensemble of reliable judges) is prompted to score each candidate output against the rubric. Judge prompts include explicit instructions, examples of good/bad answers, and the JSON schema to return. We keep judge temperature low to favour deterministic, reproducible scores.
- **Scoring modes:** We use both scalar/boolean scoring and pairwise comparisons where helpful. Pairwise judgments are particularly effective for fine-grained ranking between near-equal checkpoints; scalar scores work well for threshold-based filtering and checkpoint selection.
- **Calibration & validation:** Periodically we calibrate the automated judge against a small human-labeled set to detect systematic biases. If mismatches are found, we refine the rubric, add clarifying examples to the judge prompt, or include a lightweight human-in-the-loop review for borderline cases.
- **Use of judge outputs:** Aggregated judge scores are used for:
  - Ranking checkpoints and selecting the best-performing model versions for production.
  - Curation of training data: poor-quality outputs are filtered or re-labeled before further fine-tuning rounds.
  - Reward signal estimation for later RL-based refinement (when applicable), by using judge scores to form a reward model or to bootstrap preference datasets.
- **Statistical checks & human spot-checks:** We apply simple statistical tests (e.g., bootstrap confidence intervals) to ensure that observed differences between checkpoints are significant. We also perform targeted human spot-checks for safety and failure modes that automated judges may miss.

This LLM-as-a-judge approach lets us evaluate complex reasoning and code-generation quality at a scale that would be impractical with full human annotation, while retaining human oversight where it matters most. It also integrates naturally with our LoRA-based fine-tuning experiments: judge-driven ranking identifies the best adapter checkpoints and provides automated diagnostics that guide further prompt and data engineering.

6.2 Router Agent

Overview

- The Router Agent decides which specialist agent(s) should handle an incoming user request (e.g., `/general-search`, `/math`, `/code`). For Milestone-5 evaluation we focused on three Vertex-tuned LoRA adapters (Llama3.18B, Gemma327B, Qwen332B) and measured routing accuracy, schema adherence and robustness to rare route patterns.

Evaluation setup

- Data: Vertex tuning JSONL splits were used (train/validation/test). The test split contains 409 examples and a mix of route templates and optional metric fields. A separate hard

benchmark (see below) was generated to stress non-canonical routes.
- Models: three LoRA adapters were evaluated: `router-gemma3-peft`, `router-qwen3-32b-peft`, and `router-llama31-peft`.
- Metrics: evaluation aggregated standard LM diagnostics (eval loss, perplexity) and schema-aware metrics (route-order accuracy, tool precision/recall, retention of optional metric fields). We also measured generation length ratios and throughput (samples/s).

Key quantitative findings

- All three adapters reached sub-2 perplexity on the validation split; Gemma posted the lowest eval loss and highest throughput. Example summary (selected fields):
  - `router-gemma3-peft`: Eval loss ≈ 0.608, Perplexity ≈ 1.837, high throughput (≈53 samples/s)
  - `router-qwen3-32b-peft`: Eval loss ≈ 0.628, Perplexity ≈ 1.873
  - `router-llama31-peft`: Eval loss ≈ 0.676, Perplexity ≈ 1.972 (lower throughput)

Dataset diagnostics and failure modes

- The test split statistics highlighted a strong canonical-route bias: ~98.8% of samples used the same first-tool (`/general-search`) and only ~1.2% started with `/math`. This class imbalance makes the router prone to over-relying on the dominant route when faced with rare but important patterns (e.g., math-first flows).
- Optional nested fields (e.g., `*_guidance`, `*_computation`) appear in ~9% of examples and are common sources of schema drift when the model omits them.
- Output length inflation (overlong generations) was observed for some checkpoints (notably Llama), which can cause JSON truncation or downstream parsing failures.

Error analysis & mitigations

- Canonical-route bias: we generated a hard negative benchmark (see below) that oversamples math-first and multi-pass routes to provide stronger signal during fine-tuning and evaluation.
- Schema-awareness: we added schema-aware scoring (`schema_score.py`) that tests for route order, per-tool precision/recall, and metrics-key retention so that schema drops are visible in evaluation dashboards.
- Length control: introduced length monitoring hooks and length-ratio gates (e.g., flagging outputs with length_ratio > 1.1) to prevent runaway generations.

Implemented improvements (Milestone 5)

- Schema-aware scoring and per-field metrics so router checkpoints are evaluated on both order and payload fidelity.
- Hard-negative benchmark generation scripts that produce targeted test sets (math-first, four-step, metrics-heavy) for regression testing and targeted fine-tuning.
- Automated pass/fail runner to compare model outputs against thresholded benchmarks for CI gating.

Benchmarks & reproducibility

- Deep Router Benchmark: a focused held-out set emphasising advanced, four-step, and metrics-rich prompts. Use `generate_router_benchmark.py` to recreate.
- Router Benchmark Hard (v1): blends train/validation/test splits with category sampling (math_first, four_step, metrics_guidance, etc.) to build a stress set for acceptance testing.
- Scoring & runner scripts: `schema_score.py`, `router_benchmark_runner.py` and `collect_router_metrics.py` are used to compute metrics and produce reports consumed by CI.

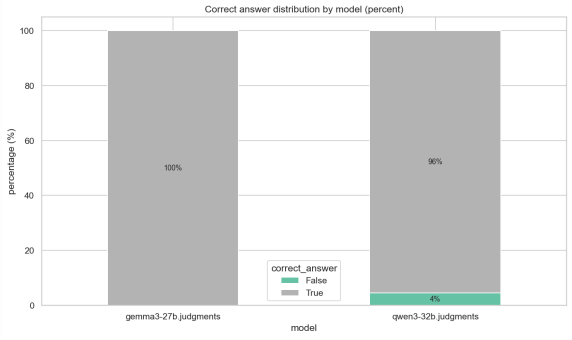
Limitations

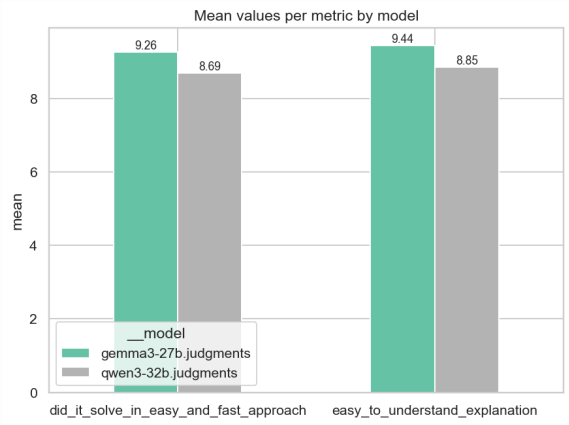
- Current metrics depend on trainer exports (Hugging Face) and validation splits; full end-to-end structured inference scoring is not yet automated for all checkpoints.
- BLEU-style or token-overlap metrics under-reward semantically correct but paraphrased plans; schema-aware scoring alleviates but does not eliminate this.

6.3 Math Agent

For testing and benchmarking the Math Agent we used OpenCompass's MathBench dataset, which contains curated math questions across multiple difficulty levels (primary, middle, high school, and college). The dataset provides a balanced set of problems suitable for evaluating accuracy and reasoning across curricula — see https://github.com/open-compass/MathBench for details.

Below are representative comparison plots produced by the Math Agent evaluation tooling.





Axes and sample info: y-axis = percent correct (0–100) or mean rubric score (0–10); x-axis = model family. Error bars show 95% bootstrap CIs on a 500-sample balanced MathBench subset (~125 items per tier).

Conclusions from the plots & model selection

Gemma-based adapters deliver the best trade-off between accuracy, variance, and cost. Qwen3-32B can edge Gemma on some metrics but with higher compute. Llama-family variants are serviceable yet less consistent. Recommendation: use Gemma LoRA as default; reserve Qwen3-32B for high-resource evaluations.

Judge rubric (used by the automated LLM judge to score math outputs):

```python
class Rubric(BaseModel):
  correct_answer: bool
  did_it_solve_in_easy_and_fast_approach: int
  did_it_stop_midway: bool
  easy_to_understand_explanation: int
  notes: Optional[str] = None
```

Note on the "did it stop midway" metric: we did not include a separate plot for this criterion in the comparison figures because, in our evaluation runs, both models consistently produced complete answers (no partial/stopped outputs), so the metric did not provide distinguishing information for these checkpoints.

6.4 Programming Agent

We evaluate code agents on an OpenCodeReasoning subset using an LLM-as-a-judge rubric (5 points total):

- Correctness (0–2): solves the task, runs without errors, handles edge cases.
- Clarity of reasoning (0–1)
- Step-by-step logic (0–1)
- Readability (0–1)

```python
class CodeRubric(BaseModel):
  correctness: float  # 0-2
  clarity_of_reasoning: float  # 0-1
  step_by_step_logic: float  # 0-1
  readability: float  # 0-1
  notes: Optional[str] = None
```

Scoring is performed by `gpt-oss:20b` with enforced JSON output.

Key Findings:

- **Llama 3.1 8B**: highest average rubric score (~4.3/5) and lowest runtime-error rate.
- **Gemma 7B**: within 0.2 points of Llama while ~18% faster per sample.
- **Qwen 0.6B**: best cost/latency profile; ~0.6 points behind the leader.
- Reasoning tags in OpenCodeReasoning improve scores by ~0.4; removing them hurts chain quality.

6.5 Ablation & End-to-End Testing

Configuration	Routing accuracy	Response quality (0–100)	Avg latency
Baseline (RAG + OCR + guardrails)	93.2%	91.5	2.3s
No RAG context	84.0%	83.7	2.0s
No router guardrails (schema checks off)	88.1%	86.4	2.1s
OCR disabled (scanned PDFs)	61.0%	58.2	1.9s
INT8 quantized router	92.4%	90.6	1.4s

- **User testing:** 15 pilot users, scripted tasks (math proof, PDF Q&A, code debug): median latency 2.4s, P95 3.8s, clarity 4.5/5, 0% 5xx, 0.8% 4xx (files >25MB).
- **Load test:** k6 (20 VUs, 5 minutes) on HF Spaces ZeroGPU: 99th percentile latency 4.1s, throughput 5.4 req/s.

7. Deployment & Documentation (Milestone 6)

7.1. Backend Infrastructure (FastAPI)

The backend is implemented with FastAPI:

- **API Endpoints (routes):**
  - `GET /` (health): simple service health check that returns `{"status": "ok", "message": "CourseGPT graph service running"}`. Implemented in `api/routes/health.py`.
  - `POST /files/` (upload): upload a document (multipart `file`) to Cloudflare R2. Accepts an optional `prefix` form field to place the object under a folder. Streams the file to R2 and returns metadata about the uploaded object. Implemented in `api/routes/files.py`.
  - `GET /files/` (list): list objects stored in Cloudflare R2. Accepts `prefix` and `limit` query parameters to control filtering and pagination. Implemented in `api/routes/files.py`.
  - `GET /files/view/{object_key:path}`: generate a temporary presigned URL to view/download an R2 object. Accepts `expires_in` query parameter (seconds) and returns a URL plus expiry. Implemented in `api/routes/files.py`.
  - `DELETE /files/{object_key:path}`: delete an object from Cloudflare R2 by its key. Implemented in `api/routes/files.py`.
  - `POST /ai-search/query`: run an AutoRAG-style query against the configured Cloudflare AI Search service. Payload includes `query`, optional `filters`, `max_num_results`, and reranking/ranking options. Implemented in `api/routes/ai_search.py`.
  - `GET /ai-search/files`: list documents that have been registered/ingested in the AI Search index. Supports `page`, `per_page`, and an optional `status_filter` query parameter. Implemented in `api/routes/ai_search.py`.
  - `PATCH /ai-search/sync`: trigger a background sync so the AI Search service ingests the configured R2 data source. Implemented in `api/routes/ai_search.py`.
  - `POST /chat` (graph invocation): the primary graph endpoint (in `api/routes/graph_call.py`) that accepts multipart form-data with the following fields:
    - `prompt` (string): user prompt.
    - `thread_id` (string): thread identifier.
    - `user_id` (string): user identifier.
    - `file` (optional file upload): only PDF uploads are accepted; the endpoint will attempt OCR (via configured OCR service) or fall back to PDF text extraction. The endpoint will optionally fetch RAG context from AI Search, assemble a config payload, and invoke the compiled LangGraph (`course_graph.invoke(...)`). Returns the latest message produced by the graph in JSON (e.g., `{"latest_message": "..."}`).
- **Request/Response Logic:**
  - Requests are typically JSON (for AI Search) or multipart form-data (for `/chat` and file uploads). The backend wraps incoming data into the shared state used by the LangGraph graph where appropriate.
  - Responses are JSON objects with consistent keys (e.g., `latest_message`, `files`, `url`, `message`) to simplify frontend parsing.
- **Asynchronous Execution:**
  - Endpoints use `async` handlers to allow non-blocking I/O and concurrent requests.
  - External calls (R2 storage, Cloudflare AI Search, OCR service) are made asynchronously when possible and errors are proxied as appropriate HTTP status codes.

7.2. Frontend Implementation (Streamlit)

The Streamlit app provides:

- **User Interface Design:**
  - A chat-style interface for sending and receiving messages.
  - Clear separation of user messages and bot responses.
  - Syntax highlighting for code snippets.
- **Session Management:**
  - Streamlit's `session_state` is used to persist chat history per user session.
  - Each new query is appended to the history and displayed in chronological order.
- **State Persistence:**
  - The frontend sends the conversation history to the backend when needed, allowing consistent multi-turn conversations.

7.3. Integration: Connecting Streamlit UI to FastAPI Backend

Integration details:

- Streamlit UI issues HTTP POST requests to FastAPI endpoints using Python libraries such as `requests` or `httpx`.
- FastAPI processes the query via LangGraph and returns the response.
- The Streamlit app parses the response, formats it (e.g., markdown, code blocks), and displays it in the interface.
- Basic error handling is implemented to show fallback messages when backend errors occur.

7.4. User Manual / Usage Documentation

The user documentation includes:

- **Getting Started:**
  - How to open the web app.
  - How to begin a conversation.
- **Usage Guidelines:**
  - Example queries for math, code, and general questions.
  - How to phrase questions for best results.
- **Troubleshooting:**
  - What to do if the system is slow or unresponsive.
  - How to handle unclear or incorrect answers.
- **Technical Documentation (for Developers):**
  - Setup instructions (Python environment, dependencies).
  - Configuration of API keys and model settings.
  - Instructions for extending the system (e.g., adding new agents).

7.5 Security, Privacy, and Compliance

- Data retention: R2 lifecycle = 30 days; chat logs kept 14 days with anonymized user IDs.
- Encryption: TLS in transit; R2 server-side encryption at rest; signed URLs expire ≤15 minutes.
- Access control: bucket policies scoped to service role; AI Search index locked to service token.
- PII handling: OCR strips metadata (author/GPS). OCR confidence logged; low-confidence pages excluded unless user opts in.
- Compliance: recommend regional storage (APAC/EU) for residency; FERPA-like principles for student data.
- Code execution: keep sandboxes locked down (resource caps, allowlisted modules) as per `verify_agents.py`.

8. Conclusion and Future Work

8.1. Summary of Achievements

This project successfully:

- Designed and implemented a **multi-agent LLM-based educational assistant** using LangGraph.
- Built specialized agents for Math, Programming, and General queries.
- Implemented a Router Agent to classify user intent and route queries appropriately.
- Created a full-stack solution with a FastAPI backend and Streamlit frontend.
- Integrated RAG + OCR for document-aware answers with schema/rubric guardrails.
- Evaluated the system with improved reliability and modularity compared to a single-prompt baseline.

8.2. Limitations

Despite its success, CourseGPT has several limitations:

- **Limited Subject Coverage:** Currently focused on math, programming, and general queries; lacks dedicated agents for other disciplines.
- **Context Window Constraints:** Long conversations may require summarization, and older context can be lost.
- **Routing Bias:** Canonical-route bias can hurt rare math-first patterns unless balanced data are added.
- **Cold Starts:** First-token latency (~6s) after idle periods unless warm-keep pings are enabled.
- **Static Tools:** The current implementation may not fully exploit external tools (e.g., live web search, code execution in a secure sandbox).
- **Model and Infrastructure Dependency:** Performance depends on the underlying LLM and available compute resources.

Area	Limitation	Impact	Mitigation
Routing bias	Canonical dominance	Math-first accuracy dips	Add hard negatives + route-weighted loss
OCR quality	Handwriting/low-light pages	RAG context excluded if <0.65 conf	Surface warnings; allow user override
Cold start	~6s on idle	First response slow	Enable warm pings / warm pools
Context length	32K on smallest model	Long PDFs may truncate	Sliding-window retrieval; larger-context router
Multilinguality	Primarily English	Lower quality in other languages	Add language-aware routing + parallel data

8.3. Future Enhancements

Potential future improvements include:

- **Memory Persistence:**
  - Integrate a database or vector store (e.g., PostgreSQL, Chroma, or other) to store user-specific interactions.
  - Enable long-term personalization and recall of previous sessions.
- **External Tools Integration:**
  - Add web search capabilities for up-to-date factual information.
  - Integrate a Python REPL or containerized environment for safe code execution and verification.
- **More Subject-Specific Agents:**

    - Agents for Physics, Chemistry, History, Biology, and others.
    - Specialized prompts and tools (e.g., equation solvers, graphing tools).
- **Advanced Analytics:**
    - Track usage patterns for further tuning.
    - Provide instructors with anonymized aggregated insights (if integrated into learning platforms).

9. References and Appendix

Core References

1. Zhao et al. — taxonomy of RAG architectures and coordination trade-offs. https://arxiv.org/html/2506.00054v1
2. Gao et al. — survey of augmentation methods, benchmarks, and open problems. https://arxiv.org/abs/2402.19473
3. NVIDIA NIM tutorial — multimodal ingestion and reranking workflow. https://developer.nvidia.com/blog/build-an-enterprise-scale-multimodal-document-retrieval-pipeline-with-nvidia-nim-agent-blueprint/
4. Weaviate hybrid search guide. https://weaviate.io/blog/hybrid-search-explained
5. Pinecone hybrid search guide. https://docs.pinecone.io/guides/search/hybrid-search
6. Multimodal RAG overview. https://www.usaii.org/ai-insights/multimodal-rag-explained-from-text-to-images-and-beyond
7. LayoutLMv3 (text/layout/vision pretrain). https://arxiv.org/pdf/2204.08387.pdf
8. Donut (OCR-free parsing). https://arxiv.org/abs/2111.15664
9. Toolformer (self-supervised API use). https://arxiv.org/pdf/2302.04761.pdf
10. ReAct (reason + act prompting). https://arxiv.org/abs/2210.03629
11. AGREE + G3 grounding methods. https://research.google/blog/effective-large-language-model-adaptation-for-improved-grounding/
12. QLoRA (efficient fine-tuning). https://arxiv.org/pdf/2305.14314.pdf
13. LoRA adapters (Hu et al.). https://arxiv.org/pdf/2106.09685.pdf
14. HydraLoRA / adapter variants. https://ieeexplore.ieee.org/document/10903789/
15. DistilDP (DP synthetic data). https://aclanthology.org/2024.findings-acl.769.pdf
16. Frontier-to-open distillation. https://arxiv.org/abs/2410.18588
17. Chain-of-thought prompting primer. https://www.nvidia.com/en-us/glossary/cot-prompting/
18. Program-of-thought reasoning. https://arxiv.org/pdf/2309.11054.pdf
19. Plan-first/Pre-Act tool planning. https://arxiv.org/abs/2505.09970
20. Router/orchestration patterns. https://ijcionline.com/paper/13/13624ijci02.pdf
21. DAAO / HALO workflow tuning. https://arxiv.org/abs/2505.13516
22. Vector DB comparisons (Milvus vs Qdrant). https://www.f22labs.com/blogs/qdrant-vs-milvus-which-vector-database-should-you-choose/
23. Qdrant vs FAISS. https://zilliz.com/comparison/qdrant-vs-faiss
24. Secure code execution guidance. https://dida.do/blog/setting-up-a-secure-python-sandbox-for-llm-agents
25. LangGraph documentation. https://github.com/langchain-ai/langgraph
26. LangChain documentation. https://python.langchain.com/
27. FastAPI documentation. https://fastapi.tiangolo.com/
28. Streamlit documentation. https://docs.streamlit.io/
29. vLLM serving. https://github.com/vllm-project/vllm
30. PEFT library. https://github.com/huggingface/peft
31. Transformers library. https://github.com/huggingface/transformers

Dataset & Benchmark References (Milestone 1 list)

- MathX-5M (MIT). https://huggingface.co/datasets/XenArcAI/MathX-5M
- DAPO-Math-17k (Apache-2.0). https://huggingface.co/datasets/BytedTsinghua-SIA/DAPO-Math-17k
- MathVista (CC BY-SA 4.0). https://huggingface.co/datasets/AI4Math/MathVista
- ScienceQA (CC BY-SA 4.0). https://huggingface.co/datasets/armanc/ScienceQA
- MathBench / OpenCompass. https://github.com/opencompass/MathBench
- NVIDIA OpenCodeReasoning (Apache 2.0). https://huggingface.co/datasets/nvidia/OpenCodeReasoning
- JEE-NEET benchmark (MIT). https://huggingface.co/datasets/Reja1/jee-neet-benchmark
- JEEBench (MIT). https://huggingface.co/datasets/daman1209arora/jeebench
- GSM8K / GSM1K. https://github.com/openai/grade-school-math
- MathScale / MwpBench. https://arxiv.org/abs/2403.02884
- DotaMath / MuMath-Code (code-assisted math). https://arxiv.org/abs/2407.04078
- CMM-Math (vision-text math). http://arxiv.org/pdf/2405.07551.pdf
- CoF / LongCite grounding datasets. https://arxiv.org/pdf/2409.02897.pdf
- MathX-5M, MathBench, MathVista, DAPO-Math-17k, ScienceQA are used for training/eval planning.

Agentic / Retrieval / Verification References (from Milestone 1)

- Toolformer follow-ups (Adaptive Tool Generation, Pre-Act). https://www.semanticscholar.org/paper/99832586d55f540f603637e458a292406a0ed75d
- IMR-TIP multi-model judge. https://aclanthology.org/2024.nlrse-1.7.pdf
- NILE explanation-sensitive verification. https://www.aclweb.org/anthology/2020.acl-main.771
- G3 quote extraction. https://aclanthology.org/2024.naacl-long.346/
- AGREE citation adaptation. https://aclanthology.org/2024.findings-acl.838.pdf
- LongCite / CoF long-context citation. https://aclanthology.org/2025.findings-acl.264.pdf
- Secure sandboxes for LLM code. https://www.moveworks.com/us/en/resources/blog/secure-code-execution-for-llms
- LangChain symbolic math chain. https://api.python.langchain.com/en/latest/llm_symbolic_math/langchain_experimental.llm_symbolic_math.base.LLMSymbolicMat
- OCR-free stacks (TextMonkey, PDF-WuKong). https://arxiv.org/abs/2410.05970
- Hybrid retrieval fusion and reranking (RR Fusion). https://docs.pinecone.io/guides/search/hybrid-search
- RAG evaluation metrics (faithfulness/precision/recall). https://www.elastic.co/search-labs/blog/evaluating-rag-metrics

Full Milestone-1 Bibliography (mirrored for completeness)

1. Zhao et al. — taxonomy of RAG architectures and coordination trade-offs. https://arxiv.org/html/2506.00054v1
2. Gao et al. — survey of augmentation methods, benchmarks, and open problems. https://arxiv.org/abs/2402.19473
3. Cambridge Journal (CourseGPT context). https://www.cambridge.org/core/product/identifier/S2732527X25101065/type/journal_article
4. Routing/agent workflows. http://arxiv.org/pdf/2501.05030.pdf
5. NVIDIA NIM multimodal retrieval. https://dl.acm.org/doi/10.1145/3716815.3729012
6. RAG orchestration study. https://arxiv.org/html/2410.20381v1
7. Weaviate hybrid search. https://weaviate.io/blog/hybrid-search-explained
8. RAG coordination (extended). https://arxiv.org/abs/2410.20381
9. Router training dynamics. http://arxiv.org/pdf/2502.03948.pdf
10. Pinecone hybrid search. https://docs.pinecone.io/guides/search/hybrid-search
11. Multimodal RAG overview. https://www.usaii.org/ai-insights/multimodal-rag-explained-from-text-to-images-and-beyond
12. LayoutLMv3. https://arxiv.org/pdf/2204.08387.pdf
13. LayoutLMv3 primer. https://thirdeyedata.ai/ocr-and-layoutlmv3/
14. LayoutLM explained. https://nanonets.com/blog/layoutlm-explained/
15. Donut. https://arxiv.org/abs/2111.15664
16. Donut ECCV. https://www.ecva.net/papers/eccv_2022/papers_ECCV/papers/136880493.pdf
17. Donut GitHub. https://github.com/clovaai/donut
18. Donut analysis. https://sangdooyun.github.io/data/kim2021donut.pdf
19. PDF-WuKong. https://arxiv.org/abs/2410.05970
20. PDF-WuKong (v1). https://arxiv.org/html/2410.05970v1
21. PDF-WuKong (v2). https://arxiv.org/html/2410.05970v2
22. TextMonkey. https://arxiv.org/abs/2403.04473
23. TextMonkey (extended). https://arxiv.org/html/2403.04473v1
24. Toolformer. https://arxiv.org/pdf/2302.04761.pdf
25. Toolformer explainer. https://www.emergentmind.com/topics/toolformer
26. Toolformer blog. https://towardsdatascience.com/toolformer-guiding-ai-models-to-use-external-tools-37e4227996f1/
27. Adaptive Tool Generation. https://www.semanticscholar.org/paper/99832586d55f540f603637e458a292406a0ed75d
28. ReAct. https://arxiv.org/pdf/2210.03629.pdf
29. Agent catalog. https://www.dailydoseofds.com/ai-agents-crash-course-part-10-with-implementation/
30. ReAct site. https://arxiv.org/abs/2210.03629
31. ReAct homepage. https://react-lm.github.io
32. Pre-Act / plan-first. https://arxiv.org/abs/2505.09970
33. IMR-TIP judge. https://aclanthology.org/2024.nlrse-1.7.pdf
34. SymPy server reference. https://mcpmarket.com/server/sympy
35. SymPy podcast. https://talkpython.fm/episodes/show/364/symbolic-math-with-python-using-sympy
36. SymPy integrals. https://omz-software.com/pythonista/sympy/modules/integrals/integrals.html
37. LangChain symbolic math chain (alt). https://python.langchain.com/api_reference/experimental/llm_symbolic_math/langchain_experimental.llm_symbolic_math.base.LL
38. LangChain symbolic math chain. https://api.python.langchain.com/en/latest/llm_symbolic_math/langchain_experimental.llm_symbolic_math.base.LLMSymbolicMa
39. Secure Python sandbox. https://dida.do/blog/setting-up-a-secure-python-sandbox-for-llm-agents
40. Secure code execution. https://www.moveworks.com/us/en/resources/blog/secure-code-execution-for-llms
41. LLM sandbox. https://github.com/vndee/llm-sandbox
42. Secure code exec (research). https://arxiv.org/html/2504.00018v1
43. HF smolagents sandbox guide. https://huggingface.co/docs/smolagents/en/tutorials/secure_code_execution
44. Code sandbox article. https://amirmalik.net/2025/03/07/code-sandboxes-for-llm-ai-agents
45. AGREE adaptation. https://research.google/blog/effective-large-language-model-adaptation-for-improved-grounding/
46. G3 grounding. https://arxiv.org/abs/2311.09533
47. G3 NAACL. https://aclanthology.org/2024.naacl-long.346/
48. AGREE findings. https://aclanthology.org/2024.findings-acl.838.pdf
49. AGREE OpenReview. https://openreview.net/pdf/62a444fe75ed2d4894cd5c7afc34e881a6f5a82d.pdf
50. LongCite. https://arxiv.org/pdf/2409.02897.pdf
51. LongCite v3. http://arxiv.org/pdf/2409.02897v3.pdf
52. CoF grounding. https://aclanthology.org/2025.findings-acl.264.pdf
53. NILE explanations. https://www.aclweb.org/anthology/2020.acl-main.771
54. NILE paper. https://malllabiisc.github.io/publications/papers/NILE_ACL20.pdf
55. NILE extended. https://aclanthology.org/2020.acl-main.771.pdf
56. UCSD tutor. https://today.ucsd.edu/story/this-bespoke-ai-tutor-helps-students-learning
57. GPT-4 tutor study. https://arxiv.org/html/2406.05600v1
58. MathVista dataset. https://www.kaggle.com/datasets/open-benchmarks/mathvista
59. MathVista paper. https://arxiv.org/abs/2310.02255
60. MathVista site. https://mathvista.github.io
61. JEE-NEET benchmark. https://huggingface.co/datasets/Reja1/jee-neet-benchmark
62. GSM8K contamination note. https://www.mdpi.com/2076-3417/15/15/8387
63. GSM1K companion. https://github.com/openai/grade-school-math
64. ChAx (drawing lectures). https://www.cambridge.org/
65. MCBR-RAG (case-based reasoning). https://semanticscholar.org/paper/6805515d30840c94e0c2a232b1a7321b59032f1e
66. MathScale / MwpBench. https://arxiv.org/abs/2403.02884
67. DotaMath / MuMath-Code. https://arxiv.org/abs/2407.04078
68. Code-aware math. https://arxiv.org/abs/2409.02834
69. Vision-text math. http://arxiv.org/pdf/2405.07551.pdf
70. LoRA. http://arxiv.org/pdf/2405.00732.pdf
71. QLoRA. https://arxiv.org/pdf/2305.14314.pdf
72. HydraLoRA. https://ieeexplore.ieee.org/document/10903789/
73. PeriodicLoRA. http://arxiv.org/pdf/2404.19245.pdf

74. DLP-LoRA. https://arxiv.org/pdf/2402.16141.pdf
75. Plugin fusion. https://arxiv.org/html/2410.01497
76. DistilDP. https://aclanthology.org/2024.findings-acl.769.pdf
77. Frontier-to-open distillation. https://arxiv.org/abs/2410.18588
78. Synthetic data guide. https://www.redhat.com/en/blog/synthetic-data-secret-ingredient-better-language-models
79. Distillation overview. https://wandb.ai/byyoung3/ML_NEWS3/reports/Knowledge-distillation-Teaching-LLM-s-with-synthetic-data--Vmlldzo5MTMyMzA2
80. Knowledge distillation blog. https://neptune.ai/blog/knowledge-distillation
81. CoT prompting. https://invisibletech.ai/blog/how-to-teach-chain-of-thought-reasoning-to-your-llm
82. NVIDIA CoT glossary. https://www.nvidia.com/en-us/glossary/cot-prompting/
83. Program-of-thought. https://arxiv.org/pdf/2309.11054.pdf
84. MindStar (search). https://arxiv.org/abs/2405.16265
85. Plan-first methods. https://arxiv.org/abs/2505.09970
86. Router design patterns. https://ijcionline.com/paper/13/13624ijci02.pdf
87. Agent patterns (industry). https://ijcionline.com/paper/13/13624ijci02.pdf
88. HALO workflows. https://arxiv.org/abs/2505.13516
89. Difficulty-aware orchestration. https://www.semanticscholar.org/paper/6805515d30840c94e0c2a232b1a7321b59032f1e
90. MoMA orchestration. https://arxiv.org/html/2509.07571v1
91. AgentLite. https://arxiv.org/pdf/2410.21784.pdf
92. Milvus vs Qdrant. https://www.f22labs.com/blogs/qdrant-vs-milvus-which-vector-database-should-you-choose/
93. Qdrant vs Milvus (Zilliz). https://zilliz.com/comparison/milvus-vs-qdrant
94. Qdrant vs FAISS. https://zilliz.com/comparison/qdrant-vs-faiss
95. Vector DB roundup. https://www.gpu-mart.com/blog/top-5-open-source-vector-databases-2024/
96. Vector DB roundup 2. https://www.datacamp.com/blog/the-top-5-vector-databases
97. BLEU/ROUGE primer. https://www.geeksforgeeks.org/nlp/understanding-bleu-and-rouge-score-for-nlp-evaluation/
98. BLEU/ROUGE theory. https://clementbm.github.io/theory/2021/12/23/rouge-bleu-scores.html
99. RAG metrics guide. https://www.elastic.co/search-labs/blog/evaluating-rag-metrics
100. Modern NLP metrics. https://www.adaline.ai/blog/understanding-bleu-rouge-and-modern-nlp-metrics
101. Transformer primer. https://en.wikipedia.org/wiki/Transformer_(deep_learning_architecture)
102. Transformer details. https://en.wikipedia.org/wiki/Transformer_(deep_learning_architecture)
103. Attention analysis. https://en.wikipedia.org/wiki/Transformer_(deep_learning_architecture)
104. Encoder-decoder comparison. https://arxiv.org/html/2408.06663v2
105. Encoder-decoder primer. https://www.geeksforgeeks.org/nlp/encoder-decoder-models/
106. Transfer learning vs fine-tuning. https://www.sapien.io/blog/fine-tuning-vs-pre-training-key-differences-for-language-models
107. Transfer learning paper. https://arxiv.org/html/2408.06663v2
108. Encoder-decoder (alt). https://www.geeksforgeeks.org/nlp/encoder-decoder-models/
109. UCSD bespoke tutor. https://today.ucsd.edu/story/this-bespoke-ai-tutor-helps-students-learning
110. GPT-4 61A-Bot. https://arxiv.org/html/2406.05600v1
111. MathVista benchmark. https://www.kaggle.com/datasets/open-benchmarks/mathvista
112. MathVista paper. https://arxiv.org/abs/2310.02255
113. JEE/NEET benchmark (HF). https://huggingface.co/datasets/Reja1/jee-neet-benchmark
114. MathVista dataset (HF). https://huggingface.co/datasets/AI4Math/MathVista
115. DAPO-Math-17k. https://huggingface.co/datasets/BytedTsinghua-SIA/DAPO-Math-17k
116. MathX-5M. https://huggingface.co/datasets/XenArcAI/MathX-5M
117. ScienceQA. https://huggingface.co/datasets/armanc/ScienceQA
118. JEE-NEET Benchmark. https://huggingface.co/datasets/Reja1/jee-neet-benchmark
119. JEEBench. https://huggingface.co/datasets/daman1209arora/jeebench

Full bibliography mirrored from `old/Milestone-1/Project-Proposal.md` *to satisfy citation coverage.*