🗓️ Code Review MAS - DAILY PLAN

🔷 Day 1 – MAS Design + Mesa Setup

✅ Goals:

- Define system architecture (Coder, Reviewer, later Architect)
- Install Mesa, structure model + scheduler

Tasks:

- Create project folder (`code_review_mas`)
- Install Mesa + basic dependencies
- Define `CoderAgent`, `ReviewerAgent` classes
- Create `CodeReviewModel` with `run_task()` logic

🔷 Day 2 – Instrument Agent Behavior

✅ Goals:

- Add OpenTelemetry spans to agent `step()` methods
- Export traces to stdout or local file

Tasks:

- Install OpenTelemetry SDK (`opentelemetry-sdk`)
- Use `tracer.start_as_current_span(...)` for:
  - Task implementation
  - Code review
- Add span attributes (task ID, output, errors)

🔷 Day 3 – Add Metrics + Internal Testing

✅ Goals:

- Add internal metrics like code/task similarity
- Inject synthetic errors

Tasks:

- Use `SentenceTransformer` or `sklearn` to compute cosine similarity between task + generated code
- Simulate edge cases (ambiguous task, bad code)
- Output final span with `{"similarity": 0.81, "errors": 1}`

🔷 Day 4 – LLM Task Generation

✅ Goals:

- Integrate GPT-4-turbo for `generate_task()` (OpenAI API)

Tasks:

- Set up API key, use `openai.ChatCompletion.create(...)`
- Prompt: “Give me a backend feature request”
- Extract output, send to `CodeReviewModel.run_task()`

🔷 Day 5 – LLM-based Workflow Planning

✅ Goals:

- Decompose LLM task into subtasks (future "PlannerAgent" role)

Tasks:

- Prompt LLM to create multi-step plan for ambiguous input
- Assign subtasks to coder/reviewer manually
- Store full trace of LLM plan + MAS execution

🔷 Day 6 – Stress Test the Pipeline

✅ Goals:

- Run 20–50 tasks through system, verify trace span richness

Tasks:

- Loop: `for task in generate_tasks(n=50): model.run_task(task)`
- Save results as JSONL or Pandas DataFrame
- Check span coverage: input → plan → code → review → outcome

---

🔷 Day 7 – OpenTelemetry Export to Jaeger

✅ Goals:

- Setup Jaeger locally, send spans from MAS to UI

Tasks:

- Install Jaeger via Docker (`docker run ...`)
- Add `OTLPSpanExporter` to your tracing pipeline
- Visualize agent timelines in Jaeger (span tree per task)

🔷 Day 8 – Add Architect Agent (Optional)

✅ Goals:

- Architect injects design constraints (e.g., "optimize for speed" vs "secure login")

Tasks:

- Create `ArchitectAgent`
- Feed contradictory goals into task
- Measure downstream error rate or agent conflict

🔷 Day 9–10 – MAST-style Analysis

✅ Goals:

- Apply metrics to trace logs:
  - Misalignment scores
  - Error propagation
  - Role-based span timelines

Tasks:

- Parse OTLP data (or span logs) into Pandas
- Cluster interactions with high error/similarity divergence
- Visualize agent interactions (networkx, Sankey, Seaborn)

🔷 Day 11–12 – Compare Experiments

✅ Goals:

- Compare:
  - With vs. without Architect

    - Ambiguous vs. precise task prompts
    - Reviewer strictness levels

Tasks:

- Run comparative batches
- Aggregate metrics (mean error, avg span length, task/code alignment)
- Save visuals and notebook summaries

🔷 Day 13 – Report & Visualizations

✅ Goals:

- Write markdown/PDF report (GitHub/Overleaf)
- Include code, spans, architecture diagram

Tasks:

- Document architecture, span schema, LLM prompts
- Add Jaeger screenshots or span timelines
- Summarize insights (e.g., “error rate rises with ambiguity”)

🔷 Day 14 – Final Polish

✅ Goals:

- Clean up repo
- Optional: submit as research artifact or internal demo

Tasks:

- Push code + data to GitHub
- Tag LLM prompts + responses
- Add README with system overview and run instructions