# 🟥 Parallel Workflows in LangGraph

---

## 1. 🎯 Context & Recap

- This video continues the **Agentic AI using LangGraph** series.
- Previous video: learned about **Sequential (linear) workflows** using LangGraph.
- Today's focus: **Parallel workflows** with two examples:
  1. A **non-LLM cricket stats workflow** (simple, logical).
  2. An **LLM-based UPSC essay evaluation workflow** (advanced, real-world style).

---

## 2. 🌟 Key Learning Goals

- Understand **how to build parallel workflows** in LangGraph.
- Learn the difference between **sequential vs parallel execution**.
- Introduce three important concepts:
  1. **Parallel execution of nodes**.
  2. **Partial state updates**.
  3. **Structured outputs & reducer functions** for handling LLM responses.

---

## 3. 🏏 Example 1: Cricket Stats Parallel Workflow (Non-LLM)

### 📝 Problem

- Input: Cricket batsman's performance data → runs, balls faced, 4s, 6s.
- Output: Calculate 3 metrics **in parallel**:
  1. **Strike rate** = (Runs ÷ Balls) × 100
  2. **Boundary %** = (Runs from 4s+6s ÷ Total Runs) × 100
  3. **Balls per boundary (BPB)** = (Balls ÷ (4s+6s))

### ⚙️ Workflow

- **Start Node** → triggers 3 parallel nodes:
  - Calculate Strike Rate
  - Calculate Boundary %
  - Calculate BPB
- Outputs of these nodes → sent to a Summary node.
- **Summary Node** → combines results into one string.
- End of workflow.

### 📌 State Design

- Inputs: runs, balls, fours, sixes.
- Outputs: strike_rate, boundary_percent, balls_per_boundary, summary.

### ⚠️ Issue Encountered

- Error: **Invalid Update Error** due to **full state updates** in parallel.
- Why? Each node was returning the **entire state**, causing conflicts (LangGraph thought multiple nodes were trying to modify the same attributes simultaneously).

### ✅ Solution

- Use **Partial State Updates**:
  - Each node only returns the field it modifies (e.g., {"strike_rate": value} instead of whole state).
- This avoids conflicts and makes the workflow stable.

### 💡 Key Lesson

- Sequential workflows → can update the full state.
- Parallel workflows → **must update only relevant keys** (partial state).
- Recommended practice: *Always use partial updates* → works for both sequential & parallel.

---

## 4. 📖 Example 2: UPSC Essay Evaluation Workflow (LLM-based, Parallel)

### 📝 Problem

- Build a system to evaluate UPSC essays.
- Input: Essay text.

- Output:
    1. **Feedback** on multiple aspects.
    2. **Scores (0–10)** for each aspect.
    3. **Final summarized feedback** + **average score**.

📊 **Evaluation Aspects**

1. **Clarity of Thought**
2. **Depth of Analysis**
3. **Language Quality**

⚙️ **Workflow**

- **Start Node** → sends essay text to **3 parallel LLM nodes**:
    - Evaluate Clarity
    - Evaluate Analysis
    - Evaluate Language
- Each node returns:
    - Text feedback
    - Score (0–10)
- **Final Evaluation Node**:
    - Summarizes 3 feedbacks into one.
    - Computes average score from the 3 individual scores.
- **End Node** outputs final results.

📌 **State Design**

- Inputs:
    - essay_text (string).
- Outputs:
    - clarity_feedback (string).
    - analysis_feedback (string).
    - language_feedback (string).
    - individual_scores (list of integers).
    - overall_feedback (string).
    - average_score (float).

---

**5. 🔑 Important Concepts Introduced**

**1. Structured Outputs**

- Problem: LLMs may return inconsistent formats (e.g., “seven” instead of 7).
- Solution: Use **structured output schema** (via Pydantic models).
    - Define fields (e.g., feedback: str, score: int (0–10)).
    - LLM is instructed to return JSON matching schema.
- Ensures reliability and makes parsing easier.

---

**2. Reducer Functions**

- Problem: 3 parallel nodes produce **individual scores** → need to store in a single list.
- Default behavior = overwrite (lose data).
- Solution: Use **reducer function** to **merge** parallel results.
    - Example: operator.add → appends list results.
    - Ensures scores like [8,7,6] are combined correctly.
- Other reducers: max, mean, etc.

---

**3. Final Evaluation Node**

- Two tasks:
    1. **Summarized feedback**: Merge the 3 textual feedbacks (done via normal LLM call).
    2. **Average score**: Compute mean of individual_scores.

---

**6. 🌟 Key Takeaways**

- **Sequential workflows**: return full state is fine.
- **Parallel workflows**: must use **partial updates**.

- For LLM tasks:
  - Always prefer **structured outputs** for reliable parsing.
  - Use **reducer functions** when merging parallel node outputs into one attribute.
- Parallel workflows unlock **scalability & speed**, as multiple tasks run simultaneously.
- LangGraph + LangChain integration makes it easy to combine logic + LLM power.

---

**7. 🏁 Conclusion**

- Built two types of parallel workflows:
  1. **Simple cricket stats** (no LLM).
  2. **Advanced UPSC essay evaluator** (LLM-based, structured).
- Learnt critical practices: **partial updates, structured outputs, reducer functions**.
- This strengthens the foundation → next videos will build more **complex, production-grade workflows**.

---