LangGraph Cheat Sheet

Production-Ready Agentic AI Patterns | From the Oracle 4.91/5.0 Workshop

1. Core Concepts

StateGraph: A directed graph where nodes are functions and edges define transitions

State: A typed Python dict/TypedDict shared across all nodes

Node: A Python function: state_in !' state_out (partial updates OK)

Edge: Transition between nodes — can be conditional (dynamic routing)

Checkpointer: Persists state between runs (MemorySaver, SqliteSaver, RedisSaver)

Compile: Converts graph definition into a runnable Pregel execution engine

2. StateGraph API — Quick Reference

```
from langgraph.graph import StateGraph, END
graph = StateGraph(MyState)
graph.add_node("name", fn)
graph.add_edge("a", "b")
graph.add_conditional_edges("a", router, {"x": "b", "y": END})
graph.set_entry_point("start")
app = graph.compile(checkpointer=saver)
result = app.invoke({"key": val}, config)
```

3. Graph Patterns

Sequential:

```
A !' B !' C !' END
```

- Best for: linear pipelines, chain-of-thought

Branching (Conditional Edge):

```
A !' router_fn !' {"yes": B, "no": C}
```

- router_fn returns a string key matching the edge dict

Cycle / ReAct Loop:

```
agent !' tools !' agent !' ... !' END
```

- Use interrupt_before=["node"] to add HITL approval step

4. State Definition

```
from typing import TypedDict, Annotated
from langgraph.graph import add_messages

class AgentState(TypedDict):
    messages: Annotated[list, add_messages]
    step: int
    tool_output: str | None
```

- add_messages reducer: appends, never overwrites messages list
- All fields optional in updates — only return changed keys

5. Tool Calling Node

```
from langgraph.prebuilt import ToolNode
from langchain_core.tools import tool

@tool
def search(query: str) -> str:
    """Search the web for query."""
    return run_search(query)

tools = [search]
tool_node = ToolNode(tools)
graph.add_node("tools", tool_node)
```

- ToolNode auto-handles ToolMessage creation
- Bind tools to LLM: llm.bind_tools(tools)

6. Human-in-the-Loop (HITL)

```
# Compile with interrupt
app = graph.compile(
    checkpointer=MemorySaver(),
    interrupt_before=["dangerous_tool"]
)

# Resume after human approval
app.invoke(None, config=thread_cfg)
```

- State is persisted at interrupt point via checkpointer
- Call invoke(None, config) to resume from saved state

LangGraph Cheat Sheet — Page 2

Checkpointing · Multi-Agent · Streaming · Common Patterns

7. Checkpointing — Persistence

```
# In-memory (dev/testing only)
from langgraph.checkpoint.memory import MemorySaver
checkpointer = MemorySaver()

# SQLite (local production)
from langgraph.checkpoint.sqlite import SqliteSaver
checkpointer =
SqliteSaver.from_conn_string("db.sqlite")
# Thread config (isolates sessions)
config = {"configurable": {"thread_id": "user-123"}}

# Inspect saved state
snapshot = app.get_state(config)
print(snapshot.values)
```

- thread_id isolates state per user/session
- Use get_state_history(config) to replay runs

8. Multi-Agent — Supervisor Pattern

```
from langgraph.graph import StateGraph

# Each sub-agent is its own compiled graph
researcher = researcher_graph.compile()
writer     = writer_graph.compile()

# Supervisor routes between them
def supervisor(state):
    decision = llm.invoke(state["messages"])
    return {"next": decision.next_agent}

graph.add_conditional_edges(
    "supervisor",
    lambda s: s["next"],
    {"researcher": "researcher",
     "writer": "writer", "DONE": END}
)
```

- Parallel: add_edge("supervisor", ["agent1", "agent2"])
- Use Send() for dynamic fan-out to N worker agents

9. Streaming Output

```
# Stream node updates
for chunk in app.stream(input, config):
    for node, update in chunk.items():
        print(f"{node}: {update}")

# Stream LLM tokens (astream_events)
async for event in app.astream_events(
        input, config, version="v2"):
    if event["event"] == "on_chat_model_stream":
        print(event["data"]["chunk"].content)
```

- stream_mode="values": full state after each node
- stream_mode="updates": only changed keys (more efficient)

10. Patterns & Pitfalls

' DO	Return only changed state keys from nodes
' DO	Use Annotated reducers for lists (add_messages)
' DO	Always set entry point (set_entry_point or START)
' DO	Compile once, invoke many times
' DO	Use MemorySaver for dev, RedisSaver for prod
'L DON'T	Mutate state in-place — always return a new dict
'L DON'T	Use mutable defaults in TypedDict fields
'L DON'T	Put I/O inside conditional edge functions
'L DON'T	Forget interrupt_before when tools are destructive

11. Key Imports — Copy-Paste Ready

```
from langgraph.graph import StateGraph, END, START
from langgraph.graph import add_messages
from langgraph.prebuilt import ToolNode,
tools_condition
from langgraph.checkpoint.memory import MemorySaver
from langchain_core.messages import HumanMessage,
AIMessage
from langchain_openai import ChatOpenAI
```