GSDC

Global Skill Development Council

FREE FRAMEWORKS CHEAT-SHEET · 2026 EDITION

The Agentic AI Frameworks Cheat-Sheet

A free reference with side-by-side patterns, when-to-use guidance and copy-ready snippets for every major agentic framework — plus career, salary and governance context.

What’s inside

- LangChain, LangGraph, AutoGen, CrewAI & Semantic Kernel at a glance
- Function calling (LLM) + ReAct agent pattern templates
- Agent memory: vector store setup with Pinecone / Chroma / Redis
- Side-by-side comparison and when-to-use guidance
- Copy-ready snippets for every major framework
- Testing, deployment, security & governance checklists

Published by the Global Skill Development Council (GSDC) — a vendor-neutral certification body trusted by 2,50,000+ professionals across 100+ countries. Snippets are illustrative; APIs evolve, so check each library’s current docs.

01 · ORIENTATION

The agentic stack at a glance

Most agentic systems stack the same layers, whichever framework you choose:

- ✔ **Model** — an LLM that reasons and decides
- ✔ **Tools** — functions/APIs the agent can call
- ✔ **Pattern** — function calling, ReAct, plan-and-execute
- ✔ **Memory** — short-term context + a vector store for recall
- ✔ **Orchestration** — single agent, graph or multi-agent crew
- ✔ **Ops** — testing, observability, deployment, guardrails

This cheat-sheet moves through each layer with a template you can copy. Pair it with the certification's 34 hands-on builds to turn these snippets into shipped systems.

02 · SIDE-BY-SIDE

Frameworks at a glance

Framework	Best for	Core idea
LangChain	Chains, RAG, tool agents	Composable LCEL pipelines
LangGraph	Stateful, cyclic agents	Graph of nodes & edges
AutoGen	Multi-agent conversations	Conversable agents
CrewAI	Role-based teams	Agents + tasks + crew
Semantic Kernel	Skills & planners (Py/.NET)	Plugins + planners

They overlap heavily — many production systems combine two (e.g. LangChain tools inside a LangGraph state machine).

02 · WHEN TO USE WHICH

Choosing a framework

- ✔ **Reach for LangChain** when you need quick RAG, tool-calling agents and a huge integration ecosystem.
- ✔ **Reach for LangGraph** when control flow matters — loops, retries, human-in-the-loop and explicit state.
- ✔ **Reach for AutoGen** when agents should converse to solve a task, including code execution.
- ✔ **Reach for CrewAI** when you want clear roles and tasks with minimal boilerplate.
- ✔ **Reach for Semantic Kernel** when you live in the Microsoft / .NET world or want planner-driven skills.

Rule of thumb: prototype with the simplest option, then graduate to an explicit graph as reliability and control requirements grow.

03 · PATTERN TEMPLATE

Function calling (LLM)

Define a tool, bind it to the model, let the LLM decide when to call it.

```
# Function calling with LangChain
from langchain_core.tools import tool
from langchain_openai import ChatOpenAI

@tool
def get_weather(city: str) -> str:
    "Return current weather for a city."
    return fetch_weather(city)

llm = ChatOpenAI(model="gpt-4o").bind_tools([get_weather])
msg = llm.invoke("What is the weather in Pune?")
print(msg.tool_calls)   # name + args the model chose
```

The model returns a structured tool call; your code executes it and feeds the result back for the final answer.

03 · PATTERN TEMPLATE

ReAct agent (Reason + Act)

A loop where the model reasons, calls a tool, observes the result, and repeats.

```
# Prebuilt ReAct agent (LangGraph)
from langgraph.prebuilt import create_react_agent
from langchain_openai import ChatOpenAI

agent = create_react_agent(
    model=ChatOpenAI(model="gpt-4o"),
    tools=[search, calculator],
)

result = agent.invoke(
    {"messages": [("user", "Find X, then compute Y")]}
)
print(result["messages"][-1].content)
```

ReAct is the workhorse pattern for tool-using agents — simple, transparent and easy to debug.

04 · FRAMEWORK

LangChain — composable pipelines

LCEL pipes prompt → model → parser into one runnable chain.

```
# LCEL chain
from langchain_core.prompts import ChatPromptTemplate
from langchain_core.output_parsers import StrOutputParser
from langchain_openai import ChatOpenAI

prompt = ChatPromptTemplate.from_template("Summarise: {text}")
chain  = prompt | ChatOpenAI(model="gpt-4o") | StrOutputParser()

print(chain.invoke({"text": document}))
```

RAG | Tool agents | Huge ecosystem

04 · FRAMEWORK

LangGraph — stateful agents

Model the agent as a graph of nodes and edges with explicit state.

```
# Stateful agent graph
from langgraph.graph import StateGraph, START, END

g = StateGraph(AgentState)
g.add_node("plan", plan_step)
g.add_node("act",  act_step)
g.add_edge(START, "plan")
g.add_edge("plan", "act")
g.add_edge("act",  END)

app = g.compile()
app.invoke({"input": task})
```

Loops & retries | Human-in-the-loop | Control

04 · FRAMEWORK

AutoGen — conversable agents

Agents converse to solve a task, optionally executing code.

```
# Two-agent conversation
from autogen import AssistantAgent, UserProxyAgent

assistant = AssistantAgent("assistant")
user = UserProxyAgent(
    "user",
    code_execution_config={"work_dir": "out"},
)
user.initiate_chat(assistant, message="Build a CSV parser")
```

Multi-agent | Code execution | Tool use

04 · FRAMEWORK

CrewAI — role-based crews

Define agents with roles, assign tasks, run them as a crew.

```
# Role-based crew
from crewai import Agent, Task, Crew

researcher = Agent(role="Researcher", goal="Find facts")
writer     = Agent(role="Writer",     goal="Draft copy")

crew = Crew(
    agents=[researcher, writer],
    tasks=[Task(description="Research X", agent=researcher),
           Task(description="Write X",    agent=writer)],
)
print(crew.kickoff())
```

Clear roles | Low boilerplate | Teams

04 · FRAMEWORK

Semantic Kernel — skills & planners

Register functions as skills; a planner composes them toward a goal.

```
# Skill + invoke (Python SK)
import semantic_kernel as sk
from semantic_kernel.connectors.ai.open_ai import OpenAIChatCompletion

kernel = sk.Kernel()
kernel.add_service(OpenAIChatCompletion("gpt-4o", api_key=KEY))

summarise = kernel.add_function(
    prompt="Summarise: {{$input}}",
    function_name="sum", plugin_name="text")

print(kernel.invoke(summarise, input=document))
```

Plugins | Planners | Py / .NET

05 · AGENT MEMORY

Vector store setup: Pinecone / Chroma / Redis

```
# Pinecone (managed)
from pinecone import Pinecone
pc = Pinecone(api_key=KEY)
index = pc.Index("agents")
index.upsert([(doc_id, vector, {"text": chunk})])
hits = index.query(vector=q, top_k=5, include_metadata=True)

# Chroma (local, embedded)
import chromadb
col = chromadb.PersistentClient(path="./db") \
        .get_or_create_collection("agents")
col.add(ids=ids, embeddings=vecs, documents=chunks)
res = col.query(query_embeddings=[q], n_results=5)

# Redis (RediSearch vector index)
import redis
r = redis.Redis()
r.hset(key, mapping={"embedding": vec_bytes, "text": chunk})
```

Pick managed (Pinecone) for scale, embedded (Chroma) for local dev, Redis when you already run it.

05 · PATTERN TEMPLATE

RAG pipeline

Retrieve relevant chunks, add them to the prompt, then answer.

```
# Minimal RAG: retrieve -> stuff -> answer
docs = retriever.invoke(question)
context = "\n\n".join(d.page_content for d in docs)

prompt = f"Use the context to answer.\n{context}\n\nQ: {question}"
answer = llm.invoke(prompt)
print(answer.content)
```

Upgrade paths: add re-ranking, citation tracking, and an eval set to measure answer quality.

06 · ORCHESTRATION

Multi-agent communication patterns

- ✔ **Supervisor / worker** — a router delegates sub-tasks to specialists
- ✔ **Sequential pipeline** — output of one agent feeds the next
- ✔ **Group chat** — agents converse until a goal is met
- ✔ **Blackboard** — agents read/write a shared state store

Transport options

In-process function calls for tight loops; message queues, REST or gRPC for distributed agents; WebSockets for streaming, interactive sessions.

07 · BEST PRACTICE

Designing tools agents can trust

- ✔ **Clear names + docstrings** — the model chooses tools from these
- ✔ **Typed, validated args** — reject bad input before side effects
- ✔ **Least privilege** — scope each tool to the minimum it needs
- ✔ **Timeouts & retries** — never let a tool hang the loop
- ✔ **Idempotency** — safe to retry without duplicate effects
- ✔ **Guardrails** — allow-lists, output checks, human approval for risky actions

08 · RELIABILITY

Testing, debugging & observability

- ✔ **Unit-test tools** in isolation before wiring them to an agent
- ✔ **Simulate conversations** with scripted inputs and expected tool calls
- ✔ **Trace everything** — decision paths, tool calls, tokens and cost
- ✔ **Catch the loops** — detect repeated actions and hallucinated tools
- ✔ **Build an eval set** — measure quality so changes are comparable
- ✔ **Replay failures** from logs to reproduce and fix

09 · PRODUCTION

Deployment checklist

- ✔ **Containerise** with Docker; orchestrate with Kubernetes if needed
- ✔ **Expose an API** (and a frontend) in front of the agent
- ✔ **Rate limit, retry, circuit-break** around every model and tool call
- ✔ **Manage secrets** — keys in a vault, never in code
- ✔ **Autoscale** for bursty agent workloads
- ✔ **Monitor** latency, cost, error rate and tool success

10 · SAFETY & GOVERNANCE

Security, safety & AI governance

As agents take real actions, governance moves from optional to required — and into job descriptions:

- ✔ **Prompt-injection defence** — treat tool output and web content as untrusted
- ✔ **Tool allow-lists** & role-based access to sensitive actions
- ✔ **Human-in-the-loop** for irreversible or high-risk steps
- ✔ **PII handling & audit logs** for traceability and compliance

Related roles: AI Safety / Guardrails Engineer, AI Governance Specialist, Responsible AI Engineer, AI Compliance Analyst.

11 · CAREER & PAY

Why framework fluency pays

Job specs increasingly name these frameworks directly — fluency is a hiring filter and a salary lever.

$147K	$115–191K	$239K
median, agentic AI engineers (US, Glassdoor 2026)	typical range	top earners

2026 hiring trends

- ✔ Agentic over generative; production skills push offers up a band
- ✔ Portfolios of real framework builds beat buzzwords
- ✔ Governance and safety skills increasingly compensated

YOUR NEXT STEP

Turn these snippets into shipped systems

This cheat-sheet gives you the patterns; the certification gives you the reps — 34 hands-on builds across every framework here, plus voice-agent and full-stack chatbot capstones.

- ✔ 12 modules · 34 hands-on builds · capstones
- ✔ Daily live sessions & 1-on-1 mentoring
- ✔ Globally recognised · 7-day money-back guarantee

Offer is time-limited. Visit the program page for the current intake.

48 HOURS LEFT **Get certified across every framework — Claim your 50% offer now →**