Survey of Agent SDKs for Production-Grade Agentic Systems

- OpenAI
- Google ADK
- Agno
- CrewAI
- LangChain

By: Pramod

Sources for this Survey / References

- **OpenAI Agents SDK:** openai.github.io/openai-agents-python (agents, running, tracing, tools, handoffs, MCP).
- **Google ADK:** google.github.io/adk-docs (Python quickstart, streaming, tools).
- **Agno:** docs.agno.com (introduction, AgentOS, production)
- **LangChain (Agents):** docs.langchain.com/oss/python/langchain/agents (create_agent, model, tools, middleware, structured output, memory); Graph API.
- **LangGraph:** langchain-ai.github.io/langgraph (overview, persistence, multi-agent); LangGraph Cloud.
- **CrewAI:** docs.crewai.com (agents, crews, tasks, flows, quickstart).
- **AutoGen:** microsoft.github.io/autogen (AgentChat, Core, extensions, getting started).
- This report is ready to use for your production framework decision and can be updated in this file as the SDKs evolve.

Framework Overview

Aspect=>	OpenAI Agents SDK	Google ADK	Agno	LangChain (Agents)	CrewAI
Source=>	openai/openai-agents-python	google/adk-docs, google-adk	agno-agi/agno	docs.langchain.com/agents	joaomdmoura/crewAI
License=>	MIT	Apache 2.0	Apache 2.0	MIT	MIT
Primary language=>	Python (also JS/TS)	Python	Python	Python	Python
Positioning=>	Lightweight multi-agent; handoffs, guardrails	Real-time voice/video; Vertex AI	SDK + Engine + AgentOS; full stack	Production-ready agent API on LangGraph; ReAct loop, middleware	Role-based crews; tasks and flows
Model binding=>	Provider-agnostic. Supports all major providers.	Provider-agnostic. Supports all major providers.	Provider-agnostic. Supports all major providers.	Provider-agnostic. Supports all major providers.	Provider-agnostic. Supports all major providers.

Agents

Multi-Agent Pattern

Metrics

Tools

Guardrails

Core Primitives

Context, Memory, and Sessions

Deployment and APIs

Observability

Execution and Running

Knowledge / RAG

Agents:

OpenAI:

```
Agent(
    name=...,
    instructions=...,
    model=...,
    tools=[...],
    handoffs=[...]
)
```

CrewAI:

```
Agent(
    role=...,
    goal=...,
    backstory=...,
    llm=...,
    tools=[...]
)
```

Google ADK:

```
Agent(
    name=...,
    description=...,
    instruction=...,
    model=...,
    tools=[...]
)
```

Agno:

```
Agent(
    name=...,
    id=...,
    model=...,
    tools=...,
    instructions=...,
    db=...,
    markdown=...,
    post_hooks=...
)
```

LangChain (Agents):

```
create_agent(
    model,
    tools,
    system_prompt=...,
    name=...,
    middleware=[...]
)
```

Tools:

OpenAI:
`@function_tool` with automatic schema from signature; Pydantic validation. Tools can be MCP servers (`mcp_servers`). Agents-as-tools via `agent.as_tool(tool_name=..., tool_description=...)`

LangChain (Agents):
`@tool` decorator (or plain functions/coroutines); tools passed to `create_agent(model, tools=[...])`.

CrewAI:
Agents get `tools=[...]` (CrewAI tool interface).
Built-in memory and tool caching.

Google ADK:
Plain Python functions as tools (e.g. `get_current_time(city: str)`). Streaming tools: async functions returning `AsyncGenerator` for real-time streams (e.g. stock, video). Tools are listed in `Agent(..., tools=[...])`.

Agno:
`@agno.tools.tool` and `Toolkit` (grouped tools with optional `instructions`, `include_tools`/`exclude_tools`). Tools receive `RunContext` for session/run.

Multi-Agent Pattern:

OpenAI:
Two main patterns: (1) **Handoffs** — peer agents delegate; runner passes conversation to the new agent. (2) **Manager (agents as tools)** — one agent invokes sub-agents via `as_tool()`. Run-level options: `handoff_input_filter`, `nest_handoff_history` (beta).

CrewAI: Crews = set of **Agents** (role, goal, backstory) + **Tasks** (description, assigned agent). **Processes**: sequential, hierarchical, or hybrid. **Flows** = event-driven, single-LLM orchestration for precise control.

Google ADK: Single `root_agent` per project; multi-agent is via composition of tools or separate ADK apps. Focus is on one primary agent with rich tools and streaming.

Agno:
Teams (multiple agents collaborating), **Workflows** (deterministic + agentic steps: `Workflow(steps=[researcher, writer])`). Your app uses one agent per product, not Team/Workflow.

LangChain (Agents): Single-agent by default; the agent is a **LangGraph subgraph** (model + tools nodes). For **multi-agent**, use LangGraph multi-agent: add agents as subgraphs and connect them; optional `name` on `create_agent` for node identity.

Execution and Running:

OpenAI:
Runner drives the loop: `Runner.run()` (async), `Runner.run_sync()`, `Runner.run_streamed()`. Loop: LLM → tool calls → run tools → re-run; or handoff → switch agent and re-run; or final output → done.
RunConfig for guardrails, tracing, model overrides, session settings, tool error formatter, etc. Optional Responses WebSocket transport and `responses_websocket_session()` for multi-turn reuse.

Agno:
```
agent.run(input,
stream=True|False,
stream_events=..., user_id=...,
session_id=...,
session_state=...)
```
→ `RunOutput` or stream of run events (`RunContentEvent`, `ToolCallStartedEvent`, etc.). No separate Runner class; run is on the agent with full event set for observability.

Google ADK:
Run via CLI `adk run my_agent` or **ADK Web** (`adk web --port 8000`). Web is for development only. Production: deploy with FastAPI/SSE or Vertex AI Agent Engine. Streaming is first-class (Gemini Live API, bidirectional voice/video).

CrewAI:
`crew.kickoff(inputs=...)` for Crew execution. **Flows** for event-driven, single-LLM steps. Streaming and async supported. CLI and **CrewAI Visual Agent Builder** for no-code; enterprise deployment options.

LangChain (Agents):
`agent.invoke({"messages": [...]})` or `agent.stream(..., stream_mode="values")`; follows LangGraph Graph API. State update is message-based. Use LangSmith for tracing. Same deployment story as LangGraph (you build the server, or LangGraph Cloud).

Context, Memory, and Sessions

OpenAI: Context = dependency injection: any Python object passed to `Runner.run(..., context=...)`, available to agents, tools, handoffs. **Sessions** (when used) via run config (`session_settings`, `session_input_callback`). No built-in persistent memory layer; you implement via context or external stores.

Agno: Sessions and **memory** are first-class. Agent can use `db` (e.g. Postgres) for session and run storage. Memory layer with best practices (e.g. `update_memory_on_run`, cheaper model for memory ops, pruning). Your app uses Postgres for sessions and custom metrics; no Agno memory in use.

Google ADK: Session management for streaming (isolated user contexts). Memory and state are model/session-scoped; no built-in long-term memory API in the quickstart/docs.

CrewAI: Memory built in (agent interaction memory); **knowledge** systems. Per-agent and crew-level context. No first-class "session" in the core; you manage identity/context in task inputs.

LangChain (Agents): State includes `messages` (conversation history) and optional custom fields via `state_schema` (TypedDict extending `AgentState`) or middleware `state_schema`. Short-term memory in state; long-term memory via LangChain patterns. Runtime **context** (e.g. `context_schema`) and **Store** (e.g. `InMemoryStore`) for cross-turn data.

Knowledge / RAG

OpenAI:
No built-in RAG primitive; you add retrieval as tools or in context.

Agno:
Knowledge + **KnowledgeTools** (search, think, analyze) over a **VectorDb**. Your app uses OpenSearch via `MultiIndexOpenSearchVectorDb` and per-product indices.

Google ADK:
No built-in RAG in core ADK; Vertex AI has separate RAG/grounding features.

LangChain (Agents):
No built-in RAG; add LangChain retrievers as tools or use retrieval in middleware/state. RAG = tool or pre-step.

CrewAI:
Knowledge management and integrations; RAG can be wired via tools or knowledge sources. Data-focused use cases supported through tooling.

Guardrails

OpenAI:
Input and output guardrails via `RunConfig.input_guardrails` and `RunConfig.output_guardrails`.
Each guardrail is a function that returns a `GuardrailFunctionOutput` with `tripwire_triggered`; if true, the SDK raises `InputGuardrailTripwireTriggered` or `OutputGuardrailTripwireTriggered` and stops the run.

Google ADK:
Safety via Gemini/Vertex AI model settings (e.g. harm categories, block thresholds). No first-class guardrail or approval API in the ADK; you rely on model-level filters and application logic.

CrewAI:
Execution controls: max iterations (default 20), execution time limits, rate limiting. **Code execution** can be disabled, run in Docker (safer), or unsafe. No documented first-class input/output guardrail or approval API; you add checks in task logic or tools.

Agno:
Built-in guardrails (often as pre-hooks): **PII detection** (SSN, credit card, email, phone) with block or mask; **prompt-injection defense** (e.g. “Ignore previous instructions”); **content moderation** (e.g. OpenAI Moderation API for hate, violence, sexual content).
Approval flows: `@approval` (pause until user confirms) and `@approval(type="audit")` (log only); applied to tools with `requires_confirmation=True`. Approval state stored in DB; runs can pause and resume. **Pre/post hooks** allow custom input and output validation.

LangChain (Agents):
Middleware hooks: `before_model`, `after_model`, `wrap_model_call`, `wrap_tool_call`, `dynamic_prompt`. Use them to validate input (before model), filter or modify output (after model), or customize tool errors. No built-in PII/content-moderation primitives; implement guardrails in middleware.

Observability

OpenAI:
Tracing on by default. Traces dashboard at platform.openai.com. RunConfig: `trace_id`, `group_id`, `workflow_name`, `trace_metadata`, `tracing_disabled`, `trace_include_sensitive_data`. Custom processors/exporters.
Disable globally: `OPENAI_AGENTS_DISABLE_TRACING=1`.

CrewAI:
Observability via **AgentOps**, **Langfuse**, and similar; enterprise tracing and team management. You host and instrument your deployment.

Google ADK:
ADK Web for dev only. Production observability via Vertex AI and/or your own logging/metrics.

Agno:
AgentOS as production runtime: 50+ APIs, JWT auth, request isolation, background execution. **Tracing:** OpenTelemetry + `openinference-instrumentation-agno`; traces in your DB and visible on os.agno.com. Your app uses `tracing=True`, Postgres, and custom metrics

LangChain (Agents):
Same observability as LangGraph: **LangSmith** for tracing, debugging, and evaluation. Agent is a graph; use LangSmith to inspect runs. No built-in dashboard in the library; you deploy and use LangSmith or OTLP.

Metrics

OpenAI:
Traces (on by default) record runs, LLM calls, tool calls, handoffs, and guardrails. **Usage** object: `inputTokens`, `outputTokens`, `totalTokens`, `requests`, plus optional `inputTokensDetails`, `outputTokensDetails`, and per-request usage for cost. Traces dashboard at platform.openai.com; optional export via custom processors/exporters or OpenTelemetry (e.g. Langfuse, Agenta, OTLP backends). No built-in latency/error aggregates; you derive them from trace data or an external observability stack.

Agno:
Traces (with OpenTelemetry + `openinference-instrumentation-agno`) include runs, model calls, tool runs, errors, and token usage; stored in your DB and viewable on os.agno.com. **Trace API:** list traces (filter by run/session/user/agent/team/workflow, status, time range), trace statistics by session. Each trace has `trace_id`, status, duration (ms), span count, error count, and IDs. You can add custom metrics

Google ADK:
OpenTelemetry instrumentation (v1.17.0+) collects telemetry from agent actions; data can go to Cloud Logging, Cloud Monitoring, and OTLP. You can add custom metrics (e.g. response quality, validation failures, tool usage). No built-in token-usage or cost API in the ADK; you rely on Vertex/Gemini metrics and your own instrumentation.

CrewAI:
Observability via integrations: **OpenLIT** (cost, latency, prompts, compliance, dashboards), **OpenTelemetry** (e.g. SigNoz for traces and metrics), and **AgentOps**, **Langfuse**, **Langtrace**, **MLflow**, **Portkey**, **Weave**, etc. You get latency, token/cost, and custom metrics from the platform you plug in; CrewAI does not define a single built-in metrics schema.

LangChain (Agents):
Same as LangGraph: **LangSmith** receives traces from agent (graph) runs; token usage, cost tracking, trace tree, project stats, dashboards. No separate metrics API; use LangSmith or export (e.g. OTLP).

Deployment and APIs

OpenAI:
Library/SDK only. You build your own FastAPI/Flask app and deploy; Runner is in-process. No built-in control plane or hosted UI.

Agno:
AgentOS = FastAPI app with routers for agents, sessions, DB, health, knowledge, memory, metrics, traces, workflows, approvals, schedules. **AG-UI** (and os.agno.com) for chat; interfaces at `/agui/v1/<agent_id>`. Your app: `agent_os.serve(app="app:app")` or uvicorn, multi-product at `/agui/v1/<product_id>`.

Google ADK:
`adk run` / `adk web` for dev. Production: deploy app (e.g. FastAPI + SSE) or use **Vertex AI Agent Engine** (hosted). Live API and WebSocket for real-time.

CrewAI:
You deploy the app (e.g. FastAPI). **CrewAI** offers enterprise deployment and Visual Agent Builder. No single canonical "platform" like AgentOS.

LangChain (Agents):
Library; you build the server (agent is a LangGraph graph). Same deployment as LangGraph: **LangGraph Cloud** for hosted. No built-in chat UI; use LangChain/LangGraph SDKs for front-ends.

Decision Guide for Production

- **Need a full deployment runtime and UI (APIs + chat + traces) in one stack?**
  → **Agno** (AgentOS + AG-UI). OpenAI and Google ADK require you to build and host the API and UI.
- **Need real-time voice/video or streaming tools (e.g. live video analysis)?**
  → **Google ADK** (Gemini Live, streaming tools). Others are text/streaming-text focused.
- **Need handoffs and manager-style multi-agent with strong guardrails and tool control?**
  → **OpenAI** (handoffs, agents-as-tools, RunConfig guardrails, tool_use_behavior).
- **Need built-in RAG and long-term memory in the same framework?**
  → **Agno** (Knowledge + VectorDb, memory layer). With OpenAI/ADK you wire RAG/memory yourself.
- **Want to avoid vendor lock-in for models and infra?**
  → **Agno** or **OpenAI** (model-agnostic / provider-agnostic). Google ADK is Gemini/Vertex-optimized even if it supports other models.
- **Want a ready-made ReAct agent (graph under the hood) with dynamic model/tools and middleware?**
  → **LangChain (Agents)** (`create_agent`, docs); same ecosystem as LangGraph and LangSmith.
- **Need graph-based workflows, checkpointing, and human-in-the-loop with time-travel debugging?**
  → **LangGraph** (StateGraph, checkpointer, LangSmith/Studio, LangGraph Cloud).
- **Prefer role-based crews, tasks, and sequential/hierarchical processes with built-in memory?**
  → **CrewAI** (Agents + Tasks + Crews, Flows, knowledge, optional Visual Agent Builder).
- **Need conversational multi-agent with code execution (local/Docker) and flexible topology?**
  → **AutoGen** (AgentChat, UserProxyAgent, AutoGen Studio for prototyping).

Thank You

For more details: reach.pramodsingh@gmail.com