# Shock Agent Suite — ADK/MCP Repo Skeleton (Google AI Agent Intensive)

This repository blueprint implements four advanced agents with on-demand tool synthesis, causal/chronosynaptic research, machine-checkable reasoning, and world-simulation planning. It is aligned with ADK function tools and MCP exposure. Copy this structure into a new repo and fill in TODOs.

---

## 0) Repository Layout

```
shock-agent-suite/
├─ README.md
├─ .env.example
├─ .gitignore
├─ docker/
│  └─ Dockerfile
├─ scripts/
│  ├─ dev_up.sh
│  ├─ run_demo.sh
│  └─ seed_fixtures.sh
├─ adk_app/
│  ├─ app.py
│  ├─ tools/
│  │  ├─ web_fetch.py
│  │  ├─ html_extract.py
│  │  ├─ pdf_tables.py
│  │  ├─ notes_graph_add.py
│  │  ├─ tool_specify.py
│  │  ├─ tool_codegen.py
│  │  ├─ tool_sandbox_run.py
│  │  ├─ tool_register.py
│  │  ├─ causal_tools.py
│  │  ├─ tda_tools.py
│  │  ├─ smt_tools.py
│  │  ├─ fuzz_tools.py
│  │  ├─ world_model_tools.py
│  │  └─ guard_tools.py
│  └─ policies/
│     ├─ tool_allowlist.yaml
│     └─ safety_rules.md
├─ mcp_server/
│  ├─ server.py
```

```
│  ├─ schemas/
│  │  ├─ web_fetch.json
│  │  ├─ html_extract.json
│  │  ├─ pdf_tables.json
│  │  ├─ notes_graph_add.json
│  │  ├─ tool_specify.json
│  │  ├─ tool_codegen.json
│  │  ├─ tool_sandbox_run.json
│  │  ├─ tool_register.json
│  │  ├─ causal_tools.json
│  │  ├─ smt_tools.json
│  │  └─ world_model_tools.json
│  └─ registry/
│     └─ README.md
├─ sandbox/
│  ├─ worker.py
│  ├─ policy.py
│  └─ tests/
│     ├─ sample_oracles.json
│     └─ tool_examples/
│        └─ pdf_ocr_tables/
│           ├─ fixture.pdf
│           └─ expected.json
├─ engines/
│  ├─ chronosynaptic/
│  │  ├─ graph_store.py
│  │  ├─ episodes.py
│  │  └─ planner.py
│  ├─ program_of_thought/
│  │  ├─ ir.py
│  │  ├─ compiler.py
│  │  ├─ smt_adapter.py
│  │  └─ viz.py
│  └─ world_sim/
│     ├─ model.py
│     ├─ rollout.py
│     └─ evc.py
└─ tests/
   ├─ test_tool_foundry.py
   ├─ test_causal_planner.py
   ├─ test_pot_auditor.py
   └─ test_world_sim.py
```

---

## 1) Quickstart

```
# 1) Clone and setup
cp .env.example .env
python -m venv .venv && source .venv/bin/activate
pip install -r requirements.txt

# 2) Start ADK app + MCP server (dev)
./scripts/dev_up.sh

# 3) Seed fixtures for demos
./scripts/seed_fixtures.sh

# 4) Run the 90s Tool-Foundry demo
./scripts/run_demo.sh tool-foundry
```

`requirements.txt` (excerpt):

```
fastapi
uvicorn
httpx
lxml
beautifulsoup4
pydantic
networkx
scikit-learn
numpy
scipy
pandas
pillow
pytesseract
pdfminer.six
camelot-py[cv]
python-multipart
z3-solver
hypothesis
matplotlib
orjson
pyyaml
```

---

## 2) ADK Agent Host (adk_app/app.py)

```
from google.adk.agents import LlmAgent
from google.adk.tools import Tool
from google.genai.types import Tool as GeminiTool, UrlContext

from tools.web_fetch import web_fetch
from tools.html_extract import html_extract
from tools.pdf_tables import pdf_tables
from tools.notes_graph_add import notes_graph_add
from tools.tool_specify import tool_specify
from tools.tool_codegen import tool_codegen
from tools.tool_sandbox_run import tool_sandbox_run
from tools.tool_register import tool_register
from tools.causal_tools import corpus_ingest, causal_propose, causal_intervene,
evidence_verify
from tools.tda_tools import tda_summarize
from tools.smt_tools import pot_compile, smt_check, proof_export
from tools.fuzz_tools import fuzz_casegen
from tools.world_model_tools import sim_train, sim_rollout, act_execute
from tools.guard_tools import guard_estimate_risk, rollback_recover

agent = LlmAgent(
    model="gemini-2.5-flash",
    name="shock_agent_suite",
    description="Self-extending tool foundry + causal planner + PoT auditor +
world-sim planner.",
    instruction=(
        "Prefer url_context for retrieval. For new capabilities, synthesize a
tool via"
        " tool_specify→tool_codegen→tool_sandbox_run→tool_register before use."
        " For research, maintain claims in notes_graph_add."
        " For reasoning, compile to PoT IR and check with smt_check; export
proof."
        " For projects, simulate with sim_rollout under risk/cost guardrails."
    ),
    tools=[
        Tool(fn=web_fetch), Tool(fn=html_extract), Tool(fn=pdf_tables),
Tool(fn=notes_graph_add),
        Tool(fn=tool_specify), Tool(fn=tool_codegen), Tool(fn=tool_sandbox_run),
Tool(fn=tool_register),
        Tool(fn=corpus_ingest), Tool(fn=tda_summarize), Tool(fn=causal_propose),
Tool(fn=causal_intervene), Tool(fn=evidence_verify),
        Tool(fn=pot_compile), Tool(fn=smt_check), Tool(fn=proof_export),
Tool(fn=fuzz_casegen),
        Tool(fn=sim_train), Tool(fn=sim_rollout), Tool(fn=act_execute),
```

```
Tool(fn=guard_estimate_risk), Tool(fn=rollback_recover),
        GeminiTool(url_context=UrlContext),
    ],
)
```

---

## 3) Core Drop-in Tools (summaries + TODOs)

### 3.1 web_fetch.py

```
import httpx, time
from bs4 import BeautifulSoup

def web_fetch(url: str, timeout_sec: int = 20) -> dict:
    try:
        with httpx.Client(follow_redirects=True, timeout=timeout_sec) as s:
            r = s.get(url)
        ct = (r.headers.get("content-type") or "").lower()
        text = r.text
        title, links, md = None, [], text
        if "html" in ct or text.strip().startswith("<"):
            soup = BeautifulSoup(text, "lxml")
            title = soup.title.string if soup.title else None
            for t in soup(["script","style","noscript"]): t.decompose()
            links = [a.get("href") for a in soup.find_all("a") if a.get("href")]
            md = "\n".join(h.get_text(" ", strip=True) for h in
soup.find_all(["h1","h2","h3","p","li"]))
        return {"title": title, "markdown": md, "links": links[:200],
"content_type": ct}
    except Exception as e:
        return {"isError": True, "message": str(e)}
```

### 3.2 html_extract.py

```
from bs4 import BeautifulSoup
import re

def html_extract(markup: str, selector: str, attrs: list[str] | None = None) ->
dict:
    soup = BeautifulSoup(markup, "lxml")
    if selector.startswith("table"):
        m = re.match(r"table(?:\[(\d+)\])?$", selector)
        idx = int(m.group(1)) if m and m.group(1) else 0
        tables = soup.find_all("table")
```

```
        if idx >= len(tables):
            return {"rows": []}
        rows = []
        for tr in tables[idx].find_all("tr"):
            cells = [c.get_text(" ", strip=True) for c in
tr.find_all(["th","td"])]
            if cells: rows.append(cells)
        return {"rows": rows}
    nodes = soup.select(selector)
    out = []
    for n in nodes:
        item = {"text": n.get_text(" ", strip=True)}
        if attrs: item["attrs"] = {a: n.get(a) for a in attrs}
        out.append(item)
    return {"matches": out}
```

### 3.3 pdf_tables.py (skeleton)

```
import subprocess, tempfile, json, os
# TODO: implement with camelot/tabula fallback + OCR (pytesseract) for scanned
PDFs

def pdf_tables(url: str, pages: str | None = None) -> dict:
    # Download to temp; try camelot; fallback to OCR grid extraction
    return {"tables": [], "note": "TODO: implement PDF table extraction
pipeline"}
```

### 3.4 notes_graph_add.py

```
import time, uuid, networkx as nx
G = nx.MultiDiGraph()

def notes_graph_add(nodes: list[dict], edges: list[dict] | None = None) -> dict:
    ids = []
    for n in nodes:
        nid = n.get("id") or str(uuid.uuid4())
        G.add_node(nid, **{k:v for k,v in n.items() if k != 'id'})
        ids.append(nid)
    for e in edges or []:
        G.add_edge(e["src"], e["dst"], **{k:v for k,v in e.items() if k not in
('src','dst')})
    return {"node_ids": ids, "node_count": G.number_of_nodes(), "edge_count":
G.number_of_edges()}
```

---

## 4) Tool-Foundry (Create/Test/Register Tools On-the-Fly)

### 4.1 tool_specify.py

```
from typing import Dict

def tool_specify(task_text: str) -> Dict:
    # Heuristic sketch: derive name/description/input_schema and 2–3 oracle
tests
    # TODO: replace with LLM-backed spec generator constrained by JSON schema
    return {
        "name": "pdf_ocr_tables",
        "description": "Extract tables from scanned PDFs using OCR then
structure rows.",
        "input_schema": {
            "type": "object",
            "properties": {"url": {"type": "string"}, "pages": {"type":
["string","null"]}},
            "required": ["url"]
        },
        "oracle_tests": [
            {"input": {"url": "fixtures/fixture.pdf"}, "expect": {"tables_min":
1}}
        ]
    }
```

### 4.2 tool_codegen.py

```
from typing import Dict

def tool_codegen(spec: Dict) -> Dict:
    # Emit a Python tool file from the spec (simple template)
    name = spec["name"]
    code = f"""
import json
from typing import Dict

def {name}(url: str, pages: str | None = None) -> Dict:
    # TODO: real OCR + table extractor; this is a stub
    return {{"tables": [], "note": "stub"}}
"""
    return {"files": [{"path": f"generated/{name}.py", "content": code}],
"entrypoint": f"generated/{name}.py", "build_cmd": ""}
```

### 4.3 tool_sandbox_run.py

```
import time, resource, subprocess, tempfile, os, json

def tool_sandbox_run(files: list[dict], cmd: str = "python -c 'print(0)'",
tests: list[dict] | None = None, limits: dict | None = None) -> dict:
    limits = limits or {"cpu_sec": 5, "mem_mb": 512, "net": False}
    with tempfile.TemporaryDirectory() as td:
        for f in files:
            p = os.path.join(td, f["path"])
            os.makedirs(os.path.dirname(p), exist_ok=True)
            with open(p, "w", encoding="utf-8") as h: h.write(f["content"])
        start = time.time()
        try:
            proc = subprocess.run(cmd, cwd=td, shell=True, capture_output=True,
text=True, timeout=limits["cpu_sec"])
            wall_ms = int((time.time()-start)*1000)
        except subprocess.TimeoutExpired:
            return {"pass": False, "failures": [{"message": "timeout"}],
"metrics": {"wall_ms": limits["cpu_sec"]*1000}}
        failures = []
        # TODO: import generated tool and run tests as callables
        return {"pass": len(failures)==0, "failures": failures, "metrics":
{"wall_ms": wall_ms}}
```

### 4.4 tool_register.py

```
import json, os
REGISTRY = "mcp_server/registry"

def tool_register(schema: dict) -> dict:
    os.makedirs(REGISTRY, exist_ok=True)
    name = schema.get("name")
    path = os.path.join(REGISTRY, f"{name}.json")
    with open(path, "w", encoding="utf-8") as h:
        json.dump(schema, h, indent=2)
    return {"tool_id": name, "version": schema.get("version", "0.1.0"), "path":
path}
```

---

## 5) Causal / Chronosynaptic Research Tools

### engines/chronosynaptic/graph_store.py

```
import networkx as nx
class GraphStore:
    def __init__(self):
        self.G = nx.MultiDiGraph()
    def add_claim(self, text, source_id, confidence: float):
        nid = f"claim:{hash(text)}"
        self.G.add_node(nid, type="claim", text=text, confidence=confidence,
source_id=source_id)
        return nid
    def link(self, a, b, etype, **props):
        self.G.add_edge(a, b, type=etype, **props)
```

### adk_app/tools/causal_tools.py (stubs)

```
def corpus_ingest(uri: str) -> dict:
    return {"chunks": [], "embeddings_ref": None}

def causal_propose(variables: list[str], priors: dict | None = None) -> dict:
    return {"dags": [], "scores": []}

def causal_intervene(dag: dict, do: dict) -> dict:
    return {"expected_delta": 0.0, "plan": []}

def evidence_verify(claim_id: str, source: str) -> dict:
    return {"verdict": "unknown", "score": 0.0}
```

---

## 6) Program-of-Thought Auditor

### engines/program_of_thought/ir.py

```
from dataclasses import dataclass
@dataclass
class Step:
    name: str; inputs: dict; outputs: dict | None = None
@dataclass
class PoTIR:
    steps: list[Step]
```

### adk_app/tools/smt_tools.py

```
from engines.program_of_thought.ir import PoTIR

def pot_compile(plan_text: str) -> dict:
    # TODO: parse to IR + invariants
    return {"ir": {"steps": []}, "invariants": []}

def smt_check(ir: dict) -> dict:
    # TODO: translate to SMT, run z3
    return {"status": "unknown", "model": None}

def proof_export(format: str = "lean") -> dict:
    return {"artifact": None}
```

### adk_app/tools/fuzz_tools.py

```
from hypothesis import given, strategies as st

def fuzz_casegen(ir: dict, targets: list[str] | None = None) -> dict:
    # TODO: generate counterexamples for targets
    return {"counterexamples": []}
```

---

## 7) World-Simulation Planner

### engines/world_sim/model.py

```
class WorldModel:
    def __init__(self):
        self.params = {}
    def predict(self, state, action):
        # TODO: learned transition + cost/latency predictor
        return state, {"cost": 0.0, "risk": 0.0}
```

### adk_app/tools/world_model_tools.py

```
from engines.world_sim.model import WorldModel
WM = WorldModel()

def sim_train(history: list[dict]) -> dict:
    return {"ref": "wm:0"}
```

```python
def sim_rollout(goals: dict, tools: list[str], budget: dict) -> dict:
    # TODO: beam search plans with EVC
    return {"ranked_plans": []}

def act_execute(plan: dict) -> dict:
    return {"events": [], "artifacts": [], "costs": {}}
```

**adk_app/tools/guard_tools.py**

```python
def guard_estimate_risk(plan: dict) -> dict:
    return {"risk_score": 0.0, "red_flags": []}

def rollback_recover(snapshot_id: str) -> dict:
    return {"status": "restored"}
```

---

## 8) MCP Server (mcp_server/server.py)

```python
import json, sys
from pathlib import Path
from fastapi import FastAPI
from pydantic import BaseModel

app = FastAPI()

# Load schemas dynamically
SCHEMA_DIR = Path(__file__).parent / "schemas"
REGISTRY_DIR = Path(__file__).parent / "registry"
REGISTRY_DIR.mkdir(parents=True, exist_ok=True)

@app.get("/tools")
def list_tools():
    tools = []
    for p in SCHEMA_DIR.glob("*.json"):
        tools.append(json.loads(p.read_text()))
    for p in REGISTRY_DIR.glob("*.json"):
        tools.append(json.loads(p.read_text()))
    return {"tools": tools}

@app.post("/call/{name}")
def call_tool(name: str, body: dict):
    # TODO: route to actual implementations (import modules, validate schemas)
    return {"name": name, "echo": body}
```

## 9) Security & Policy

- **Allowlist**: `adk_app/policies/tool_allowlist.yaml` defines callable tools; block shadow/dup names.
- **Quotas**: CPU/mem/time/net limits enforced in `sandbox/policy.py` for generated tools.
- **HIL gates**: Any egress/spend/write tool returns `requiresApproval: true` for the agent to request confirmation.
- **Taint tags**: Mark externally fetched content; sanitize before reuse.
- **Audit**: Log all tool calls with `{user, ts, tool, args_hash, outcome}`.

## 10) Demos

### 10.1 90-second Tool-Foundry Demo

```
$ ./scripts/run_demo.sh tool-foundry
# Prompts agent: "Extract tables from this scanned PDF and join with a CSV…"
# Expected: tool_specify → tool_codegen → tool_sandbox_run → tool_register →
call tool → output table JSON
```

### 10.2 Causal Card Demo

```
$ ./scripts/run_demo.sh causal
# Ingest 3 papers; output claims, DAG hypotheses, suggested interventions,
confidence
```

### 10.3 PoT Auditor Demo

```
$ ./scripts/run_demo.sh pot
# Compile reasoning to IR; z3 check; export proof; fuzz for counterexamples
```

### 10.4 World-Sim Planner Demo

```
$ ./scripts/run_demo.sh world-sim
# Train tiny world model from synthetic tool histories; simulate plans; execute
top plan with guardrails
```

## 11) Testing

```
pytest -q
pytest tests/test_tool_foundry.py::test_codegen_register
pytest tests/test_pot_auditor.py::test_smt_contracts
```

---

## 12) Next Steps (Fill-In Priority)

1. Implement `pdf_tables` OCR+table extraction with fallback chain.
2. Replace spec/codegen heuristics with LLM-guided spec planner constrained by JSON schema.
3. Add `tda_summarize` persistence diagrams (Giotto-tda or Ripser) and cluster mapping.
4. Flesh out `pot_compile` translation templates and Z3 contracts for DP/graph problems.
5. Add EVC in world-sim with cost/risk prediction and rollback snapshots.
6. Harden sandbox: seccomp profile, file IO deny-list, absolute wall clock cap.

---

## 13) README.md (starter copy)

> Shock Agent Suite delivers four novel agents: 1) Tool-Foundry (self-extending tools) 2) Chronosynaptic causal research 3) Program-of-Thought auditor 4) World-simulation planner.
>
> Built for Google AI Agent Intensive. ADK function tools + MCP server. Sandbox tests, proofs, and cost/risk ledgers included.

```
make dev   # or scripts/dev_up.sh
make test
make demo TOOL=tool-foundry
```

**License:** Choose permissive (Apache-2.0) or keep private during the course.