Here’s a ready-to-use, minimal pipeline to go from “prompted frames” → “audited frames” → “postprocessed sprites” → “Phaser 3 atlas (JSON + PNG)” using **LangGraph** (durable/stateful flow), **LangChain** (model + tool wrappers), and **LangFlow** (visual dev/test). It’s designed for quick copy-paste, then expand.

TL;DR

- **Stack**: LangGraph for orchestration, LangChain for tools/LLMs, LangFlow to visualize/test.
- **Flow**: `draft_frames → auto_audit → postprocess → pack_atlas → phaser_import`.
- **Artifacts**: deterministic frame metadata (seed, pose tag), audit flags/scores, and Phaser-compatible atlas JSON.

Why these pieces?

- **LangGraph**: makes your agent **stateful** and **resumable** (great for long image jobs & retries).
- **LangChain**: consistent wrappers for LLMs/tools (ImageMagick/pngquant/your packer).
- **LangFlow**: drag-and-drop to observe runs, tweak nodes, and replay failures quickly.

State model (compact)

```
stateDiagram-v2
    [*] --> draft_frames
    draft_frames --> auto_audit
    auto_audit --> postprocess : pass
    auto_audit --> draft_frames : fail (re-draft or fix params)
    postprocess --> pack_atlas
    pack_atlas --> phaser_import
    phaser_import --> [*]
```

Artifact contracts (save as JSON alongside images)

1) Frame metadata (one per frame)

```
{
  "id": "runner_idle_0001",
  "sequence": "runner_idle",
  "index": 1,
  "seed": 133742,
  "prompt": "runner idle, GB pixel style",
  "pose_tag": "idle_A",
  "width": 64,
  "height": 64,
  "alpha": true,
  "sha256": "…",
  "created_at": "2026-01-14T00:00:00Z"
}
```

2) Audit record (paired to frame)

```
{
  "frame_id": "runner_idle_0001",
  "coherence_score": 0.92,
  "anchor_match": 0.90,
  "silhouette_ok": true,
  "palette_ok": true,
  "flags": ["OK"],
  "notes": "Outline thickness OK; pose matches anchor."
}
```

3) Atlas manifest (input to packer)

```
{
  "atlas_name": "runner_atlas",
  "padding": 2,
  "max_size": 2048,
  "images": [
    {"path": "out/frames/runner_idle_0001.png", "key":
"runner_idle_0001"},
    {"path": "out/frames/runner_idle_0002.png", "key":
"runner_idle_0002"}
  ]
}
```

Phaser 3 atlas format (output)

- **Type**: `phaser-texture-atlas` (JSON Array or Hash).
- **Frames** minimal example:

```
{
  "frames": {
    "runner_idle_0001": { "frame": {"x":0,"y":0,"w":64,"h":64} },
    "runner_idle_0002": { "frame": {"x":64,"y":0,"w":64,"h":64} }
  },
  "meta": { "image": "runner_atlas.png", "scale": "1" }
}
```

Python: LangGraph + LangChain minimal pipeline (copy-paste)

> Assumes you have: `langchain`, `langgraph`, `pydantic`, `Pillow`, `numpy`, `pngquant` (CLI), `imagemagick` (CLI), and a function `gen_frame()` that returns a PNG (swap with your model call).

```
# pip install langchain langgraph pydantic pillow numpy
# and ensure `magick` (ImageMagick) + `pngquant` are on PATH.
```

```
from typing import List, Dict, Any, Literal
from dataclasses import dataclass, field
import json, subprocess, os, hashlib
from PIL import Image
from langchain_core.runnables import RunnableLambda
from langgraph.graph import StateGraph, END

# ---------- Config ----------
OUT_FRAMES = "out/frames"
OUT_POST = "out/post"
OUT_ATLAS = "out/atlas"
os.makedirs(OUT_FRAMES, exist_ok=True)
os.makedirs(OUT_POST, exist_ok=True)
os.makedirs(OUT_ATLAS, exist_ok=True)

# ---------- Helpers ----------
def sha256_file(path:str)->str:
    h=hashlib.sha256()
    with open(path,"rb") as f:
        for chunk in iter(lambda: f.read(1<<16), b""):
            h.update(chunk)
    return h.hexdigest()

def run(cmd:list[str]):
    subprocess.run(cmd, check=True)

# Placeholder image generator; replace with your model/tool call.
def gen_frame(prompt:str, seed:int, w:int=64, h:int=64,
path:str="out.png")->str:
    # TODO: integrate your generator (e.g., ComfyUI API or local
script)
    # For now, make a dummy transparent image as a stand-in.
    img = Image.new("RGBA", (w,h), (0,0,0,0))
    img.save(path, "PNG")
    return path

def audit_png(path:str)->Dict[str,Any]:
    # Example: simple heuristics; replace with your audit logic/LLM
    with Image.open(path) as im:
        alpha_ok = im.mode in ("RGBA","LA")
        w,h = im.size
    score = 0.9 if alpha_ok and w==h and w in (32,48,64,72,96,128)
else 0.6
    return {
        "coherence_score": score,
        "anchor_match": score - 0.02,
        "silhouette_ok": True,
        "palette_ok": True,
        "flags": ["OK"] if score>=0.8 else ["RETRY"],
        "notes": "Auto-audit heuristic"
    }

def postprocess_png(src:str, dst:str):
    # Trim, then pad to even canvas if needed; quantize for game perf.
    tmp = dst.replace(".png","_trim.png")
    run(["magick", src, "-trim", "+repage", tmp])
    # Optional: ensure power-of-two or grid alignment here.
    run(["pngquant", "--force", "--output", dst, "256", tmp])
    os.remove(tmp)

def pack_atlas(images:List[Dict[str,str]], atlas_png:str,
atlas_json:str, padding:int=2):
    """
    Naive packer for demo (left-to-right). Replace with a proper
packer when ready.
    """
    # Load images
    frames = []
    bitmaps = []
    width = padding
    row_h = 0
    for item in images:
        p = item["path"]
        k = item["key"]
        im = Image.open(p).convert("RGBA")
        bitmaps.append((k, im))
    # Simple row pack
    atlas_w = sum(im.size[0] for _,im in bitmaps) +
padding*(len(bitmaps)+1)
    atlas_h = max(im.size[1] for _,im in bitmaps) + padding*2
    atlas = Image.new("RGBA", (atlas_w, atlas_h), (0,0,0,0))
    x = padding
    frames_dict = {}
    for key, im in bitmaps:
        atlas.paste(im, (x, padding))
        w,h = im.size
        frames_dict[key] = {"frame":{"x":x,"y":padding,"w":w,"h":h}}
        x += w + padding
    atlas.save(atlas_png, "PNG")
    with open(atlas_json, "w") as f:
        json.dump({"frames": frames_dict, "meta":{"image":
os.path.basename(atlas_png), "scale":"1"}}, f, indent=2)

# ---------- LangGraph State ----------
@dataclass
class PipelineState:
    goal: str
    sequence: str                          # e.g., runner_idle
    seeds: List[int]                       # one per frame
    width: int = 64
    height: int = 64
    frames: List[Dict[str,Any]] = field(default_factory=list)     #
metadata
    audits: Dict[str,Dict[str,Any]] = field(default_factory=dict) #
frame_id -> audit
    approved_ids: List[str] = field(default_factory=list)
    atlas_name: str = "runner_atlas"
    padding: int = 2

# ---------- Nodes ----------
def node_draft(state:PipelineState)->PipelineState:
    new_frames=[]
    for i,seed in enumerate(state.seeds, start=1):
        frame_id = f"{state.sequence}_{i:04d}"
        out_path = f"{OUT_FRAMES}/{frame_id}.png"
        gen_frame(prompt=state.goal, seed=seed, w=state.width,
h=state.height, path=out_path)
        meta = {
            "id": frame_id,
            "sequence": state.sequence,
            "index": i,
            "seed": seed,
            "prompt": state.goal,
            "pose_tag": "auto",
            "width": state.width,
            "height": state.height,
            "alpha": True,
            "sha256": sha256_file(out_path)
        }
        new_frames.append(meta)
        with open(f"{OUT_FRAMES}/{frame_id}.json","w") as f:
            json.dump(meta, f, indent=2)
    state.frames.extend(new_frames)
    return state

def node_audit(state:PipelineState)->PipelineState:
    approved=[]
    for meta in state.frames:
        fid = meta["id"]
        png = f"{OUT_FRAMES}/{fid}.png"
        if fid in state.audits:
            continue
        audit = audit_png(png)
        state.audits[fid]=audit
        with open(f"{OUT_FRAMES}/{fid}.audit.json","w") as f:
            json.dump({"frame_id": fid, **audit}, f, indent=2)
        if "OK" in audit["flags"]:
            approved.append(fid)
    state.approved_ids = approved
    return state
```

```
def node_postprocess(state:PipelineState)->PipelineState:
    for fid in state.approved_ids:
        src = f"{OUT_FRAMES}/{fid}.png"
        dst = f"{OUT_POST}/{fid}.png"
        if not os.path.exists(dst):
            postprocess_png(src, dst)
    return state

def node_pack(state:PipelineState)->PipelineState:
    images = [{"path": f"{OUT_POST}/{fid}.png", "key": fid} for fid in
state.approved_ids]
    atlas_png  = f"{OUT_ATLAS}/{state.atlas_name}.png"
    atlas_json = f"{OUT_ATLAS}/{state.atlas_name}.json"
    pack_atlas(images, atlas_png, atlas_json, padding=state.padding)
    # Save manifest used
    with open(f"{OUT_ATLAS}/{state.atlas_name}.manifest.json","w") as
f:

json.dump({"atlas_name":state.atlas_name,"padding":state.padding,"imag
es":images}, f, indent=2)
    return state

def node_import(state:PipelineState)->PipelineState:
    # In a real project: copy/move to your Phaser public assets dir.
    # Here we just “finish”.
    return state

# ---------- Build the graph ----------
graph = StateGraph(PipelineState)
graph.add_node("draft_frames", RunnableLambda(node_draft))
graph.add_node("auto_audit",   RunnableLambda(node_audit))
graph.add_node("postprocess",  RunnableLambda(node_postprocess))
graph.add_node("pack_atlas",   RunnableLambda(node_pack))
graph.add_node("phaser_import",RunnableLambda(node_import))

graph.set_entry_point("draft_frames")
graph.add_edge("draft_frames","auto_audit")
graph.add_edge("auto_audit","postprocess")
graph.add_edge("postprocess","pack_atlas")
graph.add_edge("pack_atlas","phaser_import")
graph.add_edge("phaser_import", END)

app = graph.compile()

if __name__ == "__main__":
    init = PipelineState(
        goal="16-bit idle loop, anchor match, DMG-style palette",
        sequence="runner_idle",
        seeds=[101,102,103,104,105],
        width=64, height=64,
        atlas_name="runner_atlas",
        padding=2
    )
    final = app.invoke(init)
    print("Done. Atlas in:", OUT_ATLAS)
```

LangFlow setup (visual harness)

- Add nodes mapping to the graph stages:
    - **Draft Frames**: your generator node (e.g., ComfyUI/Replicate/local script).
    - **Auto Audit**: scoring node (LLM rubric + heuristics).
    - **Postprocess**: shell tool nodes for `magick` and `pngquant`.
    - **Pack Atlas**: Python node invoking your packer.
    - **Emit**: file sink (to your Phaser assets folder).
- Use **Run & Inspect** to replay failed branches; pin inputs for quick A/B tests.

Phaser 3: quick import snippet

```
// preload()
this.load.atlas('runner', 'assets/runner_atlas.png',
'assets/runner_atlas.json');

// create()
this.anims.create({
  key: 'idle',
  frames:
['runner_idle_0001','runner_idle_0002','runner_idle_0003'].map(f => ({
key:'runner', frame:f })),
  frameRate: 8,
  repeat: -1
});
this.add.sprite(200, 200, 'runner').play('idle');
```

Notes for your 16BitFit pipeline

- Swap `gen_frame()` with your preferred **image-to-image + pose control** call (e.g., ComfyUI+IP-Adapter/OpenPose). Keep **seed** plumbed for determinism.
- Extend `audit_png()` to enforce **outline thickness, palette checks, silhouette stability**, and **anchor-sim** (e.g., SSIM over edges).
- Replace the naive packer with a **bin-packing** lib (e.g., `rectpack` or your existing atlas tool) but keep the **Phaser frame JSON** shape.

—---

LangChain Docs

https://docs.langchain.com/

Google Document on building a LangGraph Agent:
https://docs.cloud.google.com/agent-builder/agent-engine/develop/langgraph

Google Document on building a LangChain Agent:
https://docs.cloud.google.com/agent-builder/agent-engine/develop/langchain

Google Document on building an Agent to Agent workflow:
https://docs.cloud.google.com/agent-builder/agent-engine/develop/a2a