FIELD GUIDE

Mneme Architecture Field Guide

A local-first, multi-persona AI platform with MCP tool integration and modular creator systems

Author: Dave Wheeler (aka Weller Davis) · Version: 1.0 · Date: December 2025

Executive Summary
Mneme is a production AI system that favors local-first operation, specialized personas with LoRA lineages, and a tool bridge (MCP) that turns LLM “thinking” into real-world “doing.” A role-based LLM router, a unified scheduler, a disciplined filesystem, and checkpointed recovery form a reusable spine across creator modules (E-book, Audiobook, Image, Music, Code). This guide documents the architecture patterns and operational practices that made it durable.

Table of Contents

1. System Overview

2. Role-Based Multi-LLM Router

3. Personas with LoRA Lineage

4. MCP: Bridging Thinking and Doing

5. Creator Module Pattern

6. Image & Music Pipelines (ComfyUI + YuE)

7. Code Creator: Plan/Execute/Validate/Fix

8. Operations & Recovery

9. Roadmap & Hiring Note

Appendix: Commands & Checklists

1. System Overview

Mneme’s core tenets: local-first by default, role-based LLM routing, persona specialization with LoRA, and a standardized tool bridge (MCP). The same spine powers multiple creator modules with consistent observability and recovery.

High-Level Flow (ASCII)

```
[User/Task]
  → LLM Router (role → provider)
  → Persona (prompt + LoRA lineage)
  → Creator Orchestrator
  → MCP Tools (filesystem, fetch, browser, pandoc, ComfyUI)
  → Validation Gates
  → Artifacts (EPUB/PDF/MP3/PNG/FLAC)
  → Publishing (site, LinkedIn)
  → Checkpoint/Retry → Resume
```

Layer	Purpose	Notes
Router	Maps cognitive roles → providers	local_llm, research_llm, code_llm, ponder_llm, image_llm
Personas	Specialized capabilities	Versioned LoRA lineage per persona
MCP	Tool bridge	filesystem, fetch, browser, pandoc, ComfyUI
Orchestrators	Module workflows	Planning, execution, validation, artifacts
Ops	Durability & visibility	Scheduler, filesystem discipline, WebSockets, recovery

2. Role-Based Multi-LLM Router

Tasks are assigned a *role* and routed to a suitable provider. Health, cost, and streaming are monitored. LoRA adapters are loaded for local models where applicable.

Common Roles

- ✓ **local_llm**: fast planning, glue logic
- ✓ **research_llm**: deep content generation/validation
- ✓ **ponder_llm**: reflection & critique
- ✓ **image_llm**: vision analysis

Pseudo-Route

```
def route(role, messages, **opts):
  p = pick_provider(role)               # health, cost, capability
  return p.generate(messages, **opts)
```

Tip: Treat roles as cognitive contracts; avoid overloading one role with divergent tasks.

3. Personas with LoRA Lineage

Personas are skill profiles with versioned LoRA adapters. Improvements are additive and rollbackable. Examples: priya (architect), casey (coder), zed (validator), izzy/picaso (image), max (marketing).

Lineage Example

```
V1 (baseline)
  → V2 (+structure, fewer generic paragraphs)
    → V3 (+examples, pitfalls discipline)
      → V4 (+style coherence, diagram cues)
```

- ✓ Keep a manifest per version (checksum, date, purpose)
- ✓ Train on “bad vs good” pairs for targeted bias
- ✓ Gate new versions through validation content

4. MCP: Bridging Thinking and Doing

MCP offers a stable tool contract so personas can act: read/write files, fetch web content, automate browsers, convert documents, and trigger image/music workflows.

Key Tools

- ✓ filesystem_server (R/W/edit/search)
- ✓ fetch_server (HTML→markdown)
- ✓ puppeteer_server (headless browser)
- ✓ pandoc_server (doc conversions)
- ✓ comfyui_client / fooocus_server (media)

Example (Pseudo)

```
if mcp.has("pandoc_server.convert"):
  pdf = mcp.exec("pandoc_server","convert",{
    "text": md, "from_format":"markdown", "to_format":"pdf"
  })
  fs.save(project, "artifacts/book.pdf", pdf)
```

5. Creator Module Pattern

All modules share the same shape. That consistency reduces cognitive load and speeds new module creation.

Pattern

```
Topic → Project → Workflow (multi-phase)
             Plan → Generate → Validate → Fix
         → Artifacts → Publishing
```

- ✓ Unified scheduler + orchestrator per module
- ✓ WebSocket progress at key steps
- ✓ Validation gates before packaging
- ✓ Filesystem discipline per project

6. Image & Music Pipelines (ComfyUI + YuE)

Images: Fooocus (fast starts) → ComfyUI (control). Dynamic LoRA loading, quality presets, and vision-LLM validation (3 attempts) increased pass rates. Windows box with RTX 4080 (16GB) hardened (firewall, DNS pinning, SDPA over FlashAttention2).

Music: YuE two-stage: Stage A semantic tokens → Stage B audio synthesis. Stability via pinned CUDA/PyTorch, fp16 inference, and full checkpoint manifests (~18GB). Result: reliable short vocal clips; roadmap to longer tracks with stitching.

Image Quality Gate (3 attempts)

```
gen → vision_check → adjust → retry (≤3)
rubric: focal subject, white bg, arrows, no text artifacts
```

YuE v1 Success Criteria

- ✓ Full checkpoint presence (size/hash-verified)
- ✓ SDPA attention on Windows
- ✓ fp16 stable inference
- ✓ Stage-separated, resumable workflow

7. Code Creator: Plan/Execute/Validate/Fix

A two-tier LLM design splits fast analysis (persona on local_llm) from focused generation (code_llm). An incremental DevPlan turns big goals into atomic todos, executed one at a time with compact, token-efficient context.

Compact Context DSL (excerpt)

```
[TREE]
/src app.py
/tests test_app.py
[/TREE]
[F:src/app.py|py]
def main(): ...
[/F]
[PATCH:src/app.py:18-30]
+ def load_config(...): ...
[/PATCH]
```

Validation Gates

- ✓ Syntax parsing per file
- ✓ Undefined symbol/import checks
- ✓ Optional: tests/lint/type checks
- ✓ Auto-debug: create fix tasks (≤3 attempts)

8. Operations & Recovery

Durability was a design goal: a unified scheduler, disciplined filesystem, and checkpointed auto-resume keep the platform stable.

Filesystem Layout (per project)

```
~/mneme_data/{module}s/{project_id}/
  raw/ compiled/ artifacts/ audit_trail/ media/
```

Checkpoint (concept)

```
{
 "last_completed_step": "drafting",
 "resume_point": {"status":"validating","section_index":3},
 "retry_count": 2
}
```

- ✓ Bounded retries with backoff; due-for-retry logic
- ✓ "Write temp → atomic move" for artifact integrity
- ✓ Health checks before long jobs; graceful degradation
- ✓ WebSockets for observability (progress + metrics)

9. Roadmap & Hiring Note

Roadmap

- ✓ Deeper test-first in Code Creator; E2E browser agent
- ✓ Longer music via segment stitching + beat alignment
- ✓ Video: narration + storyboard → shot assembly + captions
- ✓ Persona "schools": shared elective LoRA improvements
- ✓ Portable workspaces: bundle models/prompts/artifacts

For AI Engineering Leaders

- ✓ Patterns > one-offs: reuse the spine across modalities
- ✓ Vendor-hedged: role routing + MCP keep options open
- ✓ Smaller teams, bigger leverage with personas + LoRA
- ✓ Measure with gates and artifacts, not just slideware

Contact — If you're building an AI platform or a local-first content factory, I'm happy to compare notes and help accelerate your roadmap. wellerdavis.com · linkedin.com/in/davewheeler-li/

Appendix: Commands & Checklists

Pandoc (HTML → PDF)

```
# Using WeasyPrint (recommended for modern CSS)
pandoc mneme-field-guide.html -o mneme-architecture-field-guide.pdf \
  --pdf-engine=weasyprint

# Or wkhtmltopdf (very compatible)
wkhtmltopdf mneme-field-guide.html mneme-architecture-field-guide.pdf
```

ComfyUI Health Check (example)

```
curl -s http://192.168.1.15:8188/system_stats | jq
```

Version Pinning (PyTorch/CUDA example)

```
pip install torch==2.6.0+cu124 torchvision==0.21.0+cu124 torchaudio==2.6.0+cu124 \
  -f https://download.pytorch.org/whl/torch_stable.html
```

Image Quality Gate (Vision LLM rubric excerpt)

- ✓ Single focal subject; white background; correct aspect
- ✓ No text artifacts; clear arrows; consistent style
- ✓ Three-attempt loop with parameter adjustments

New Module Integration Checklist

- ✓ Data layer: models, repos, settings (defaults)
- ✓ Orchestrator: phases, validation gates, artifacts
- ✓ Scheduler: ProjectHandler + dispatcher wiring
- ✓ API + UI: endpoints, WebSockets, progress views
- ✓ MCP usage: filesystem, fetch, browser, pandoc, ComfyUI
- ✓ Recovery: checkpoints, retries, sleep cycle
- ✓ Docs: add to codebase summary; examples & tests