Mesh Multicontext Protocol (MMP)

Decentralized, privacy-aligned AI execution mesh enabling controllable LLM function orchestration across heterogeneous nodes with minimal infrastructure.

Repository | Research Basis | Case Study

AI Infra | Edge / On-Prem | Function Calling | LLM Tooling | Security-first | Offline Models | Scalable Mesh

Role: Lead Architect & Full-Stack / AI Systems Engineer (Self-Directed Project)

Snapshot Outcomes

DEPLOYMENT FOOTPRINT	LLM (FUNC CALL)	SETUP TIME	EXTENSIBILITY
≤ 7B parameter models run locally	LLM empowered function call	< 15 min first-node bootstrap	Python functions hot-added

Professional Case Study

1. Problem Context

Agencies and regulated organizations often maintain fragmented data silos and face constraints preventing central data aggregation or unrestricted cloud use. Traditional centralised model serving platforms introduce risk, bandwidth cost, or governance concerns. Simultaneously, emerging LLM use-cases demand controlled function invocation (tool use), contextual augmentation, and incremental adoption without major hardware refresh.

MMP targets this tension: provide a mesh-oriented, low-friction orchestration layer where lightweight LLMs execute controlled function calls locally, while enabling later federation of knowledge across nodes without raw data egress.

2. Solution Overview

Mesh Multicontext Protocol (MMP) is a decentralized runtime & configuration-driven orchestration layer that couples:

Function Call Governance
LLMs restricted to declared synchronous, string-returning functions defined in diagram.json.

Multi-Context Execution
Server, embedded, and desktop contexts share a unified tool interface.

Offline Model Support
Runs with local models (e.g., qwen2.5:7b) via Ollama—no external API dependency.

Declarative Topology
diagram.json expresses nodes, roles, allowable tools, enabling reproducibility.

3. Key Contributions

- Architected end-to-end system: Python backend + Next.js interface + desktop client integration.
- Designed controlled function calling pattern with strict output + synchronous contract enforcement.
- Implemented configuration compiler (diagram.json) enabling rapid network/function mutation without redeploy.
- Integrated multi-model pipeline (core, vision, analysis) with fallbacks and context steering.
- Established secure port exposure pattern (public UI vs. local orchestration ports).
- Authored UX for function visibility, file contextual augmentation (RAG/CAG placeholder), and comparison tooling.
- Conducted iterative performance tuning (prompt minimization, cold-load handling, memory reuse).

4. High-Level Architecture

```
+---------------------+       +-----------------+
|  Desktop / Web UI    | <---> |  Next.js Server  |  (Port 3002 public)
+----------+----------+       +---------+--------+
                               |      |
         | REST / WebSocket-----|      |
         v                            v
 +-------+---------------+   +--------+-----------------+
 |  Python Orchestrator   |   |  File Server (diag,shape) | (Port 3003 local)
 |  (Function Router)     |   +--------------------------+
 |   - Validates calls    |
 |   - Model invocation   |
 |   - Function sandbox   |
 +-----------+-----------+
             |
             | Local Invocations / Tool Calls (Port 3001 loopback)
             v
      +----------------+
      |  LLM + Functions  |  (User defined: sync, string I/O)
      +-----------------+
```

Security boundary: Only UI port (3002) is exposed; execution + config layers remain local. Function registry prevents arbitrary code execution.

Technology Stack

Python | Ollama | Next.js | macOS / Linux / Windows | Tailwind | JSON Config

5. Key Technical Decisions & Rationale

Strict Synchronous Functions: Simplifies determinism, removes async complexity for early-stage mesh reliability.

String-only Return Contract: Enforces serialization uniformity; reduces injection risk from complex object graphs.

Declarative Function Registry (diagram.json): Enables reproducible governance + transparent inspectability for audits.

Model Modularity: Allows layering (core reasoning, optional vision, optional analysis) without coupling runtime logic.

Minimal External Dependencies: Offline-first design to satisfy air-gapped or compliance-driven environments.

6. Security & Privacy Considerations

- Isolation of execution and configuration ports prevents lateral configuration tampering.
- No raw data export by default; only derived insights exchanged (conceptual knowledge sharing model).
- Function whitelist eliminates arbitrary prompt-injected code execution.
- Local inference removes risk of cloud model data retention/logging.

7. Sample Configuration Pattern (diagram.json Excerpt)

```
{
"id": "node_0",
"type": "main",
"name": "main",
"description": "Intelligent assistant with image, data, web, math capabilities. Enforce: do not alter prompt when calling. Use <funct
"importPath": {
  "module": "handlerApp.ollama_handlers.handler",
  "function": "OllamaHandler",
  "aliases": "OllamaHandler"
},
"parameters": {
  "query": { "type": "string", "description": "Query to send to Handler" }
},
"model": "qwen2.5:7b",
"system_prompt": "Format: <function_call> {...} </function_call>"
}
```

Shows declarative binding of model, function entrypoint, and enforced call schema.

8. Challenges & Resolutions

Prompt Drift in Function Calls: Implemented explicit wrapper + negative constraint instructions. Result: improved call precision consistency.

Model Resource Constraints: Selected balanced 7B variant for baseline; optional heavier models tagged for post-processing only.

Contextual Augmentation Without Overfetch: Deferred heavy embedding pipeline; used staged file preview + user-controlled inclusion.

Cross-Platform Complexity: Positioned desktop client for macOS/iOS + generic web server fallback for other environments.

9. Impact & Outcomes

- Reduced barrier to internal AI prototyping for constrained environments.
- Provided a blueprint for layered function governance usable across agencies.
- Established repeatable configuration enabling future policy-driven expansion.
- Improved interpretability of AI actions via explicit function call trace surfaces.

10. Differentiators

Offline-first Orchestration
No dependency on proprietary remote APIs.

Strict Tool Governance
Predictable LLM behavior via enforced schemas.

Modular Growth Path
Add functions without redeploying architecture.

Multi-Context Parity
Same core semantics across desktop / embedded / web.

11. Future Roadmap

- Add cryptographic signing for function registry integrity.
- Introduce async task queue for long-running analytical functions.
- Pluggable embedding & retrieval layer for structured RAG pipeline.
- Role-based access & policy-driven function visibility.
- Automated performance profiling (latency & memory watermark logging).

12. Lessons Learned

- Simplicity in function contract (sync + string) accelerates adoption before optimizing complexity.
- Declarative topologies reduce drift vs. imperative instantiation scripts.
- Early demarcation of public vs. internal ports prevents retrofitted security patches.
- LLM function-call reliability improves markedly with minimal but strict system prompts.

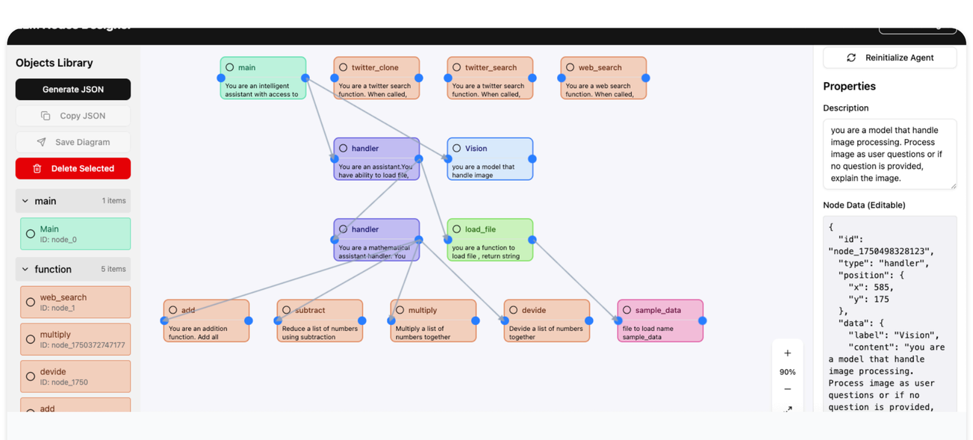
13. 30-Second Pitch

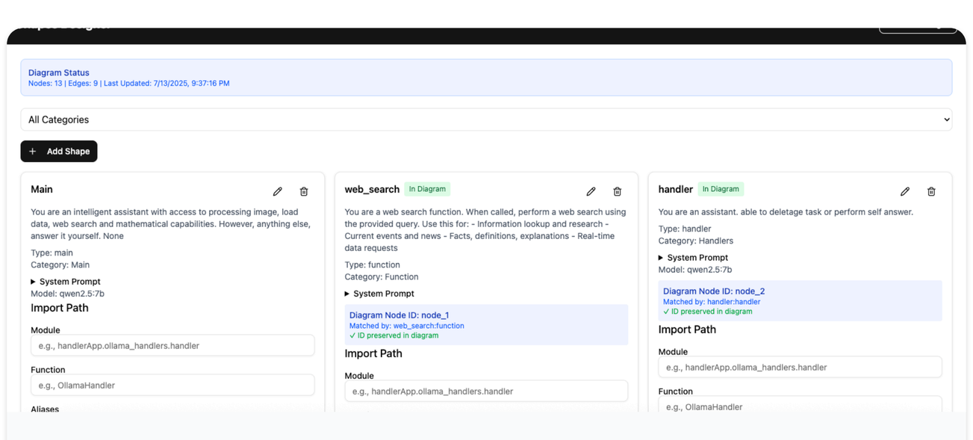
MMP is a lightweight, offline-capable AI mesh that lets organizations safely adopt LLM function calling within regulated or air-gapped environments. It offers a declarative governance layer, controlled execution, and multi-context parity—enabling teams to iterate on AI augmentation without surrendering data sovereignty or provisioning heavy centralized infrastructure.

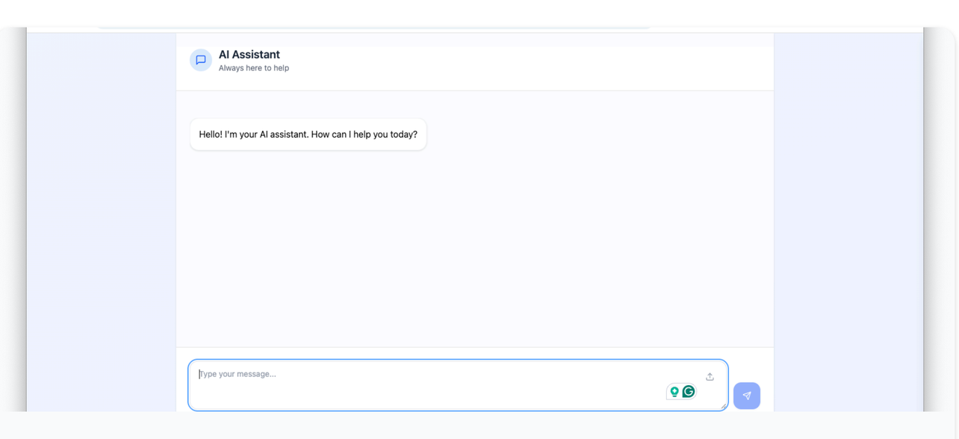
Source Repository | Research Basis

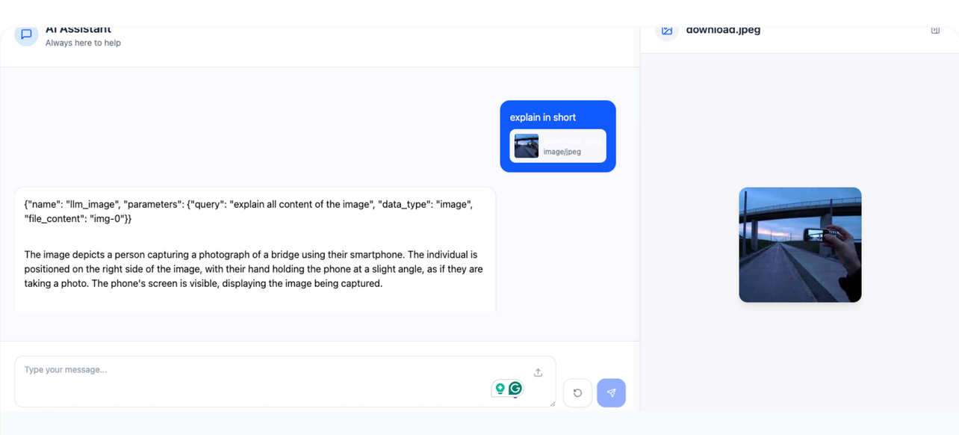
Interface & Capability Showcase

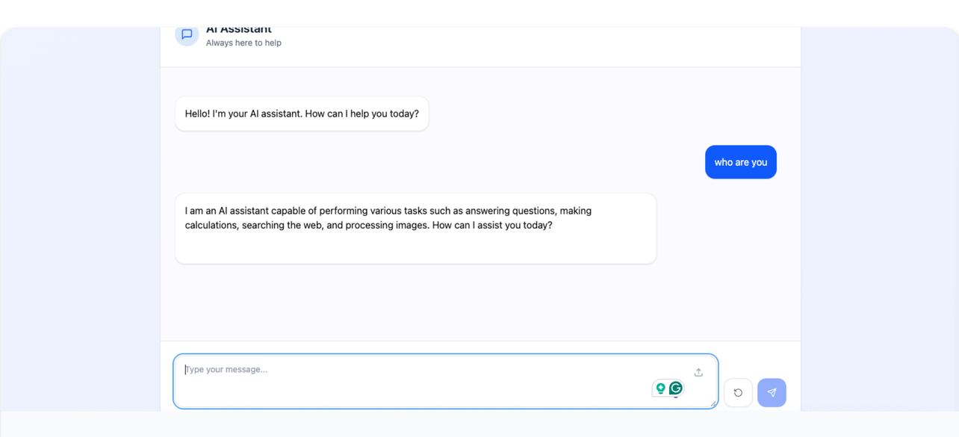
Representative UI states across server, embedded, and desktop contexts. Demonstrates transparency, configuration control, and comparative analysis capabilities.

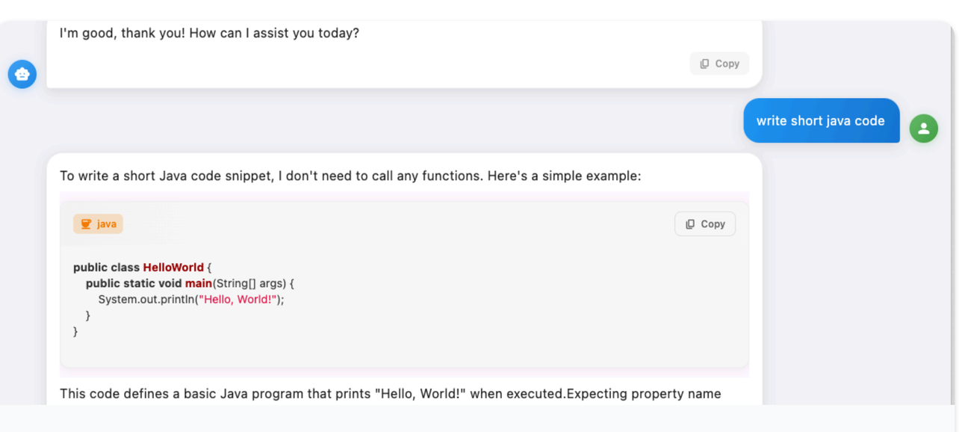

Home dashboard overview

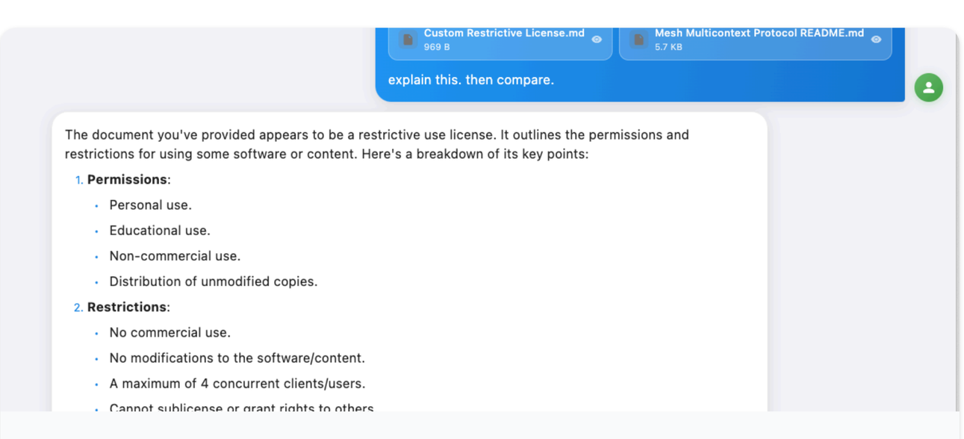

Configuration editor (diagram.json)

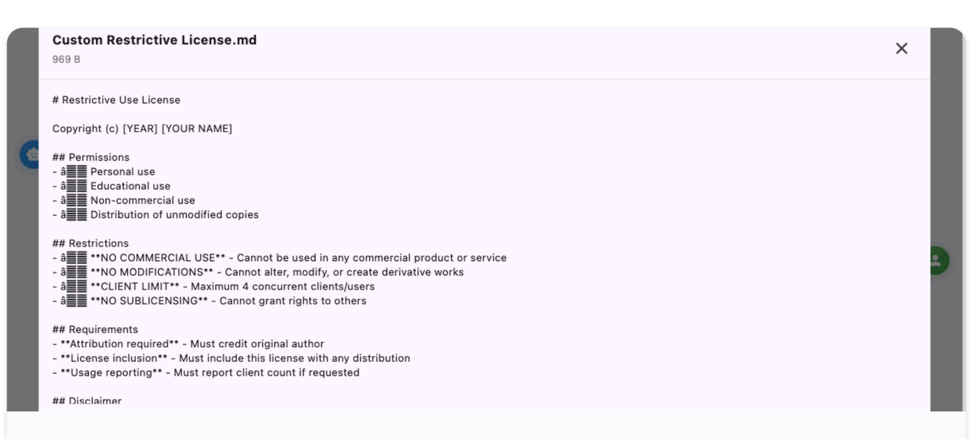

Embedded context integration

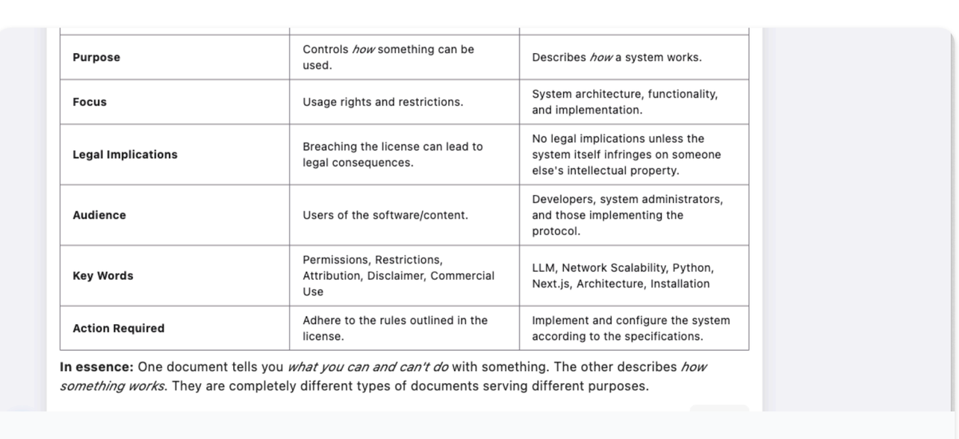

Vision / image interaction

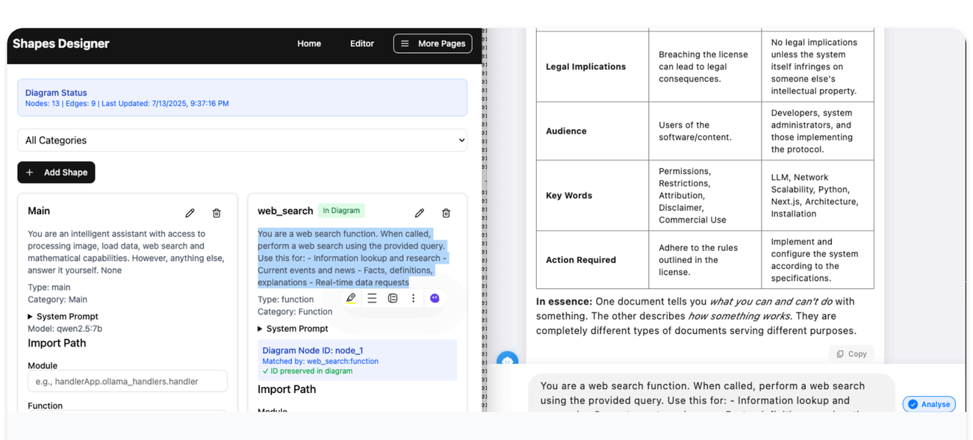

Inline reasoning output


Desktop: response + trace


Context file management


Inline document preview


Response comparison tool


Clipboard ingestion assist

Screenshots from open repository; UI subject to iteration.

Contact / Collaboration

Interested in AI infrastructure, secure on-prem orchestration, or function tooling?

mambo06@gmail.com