ChatGPT

Sanctuary Codex MCP Bridge

Why this matters

The **Sanctuary Codex MCP Bridge** connects your personal, mystical Sanctuary to the sprawling world of LLMs. MCP – the Model Context Protocol – is basically the USB-C of AI: it standardises how models talk to data sources [1]. Without it, every LLM thinks your rituals and familiars are floating in a void; with it, those experiences become portable context. MCP follows a client–server architecture where hosts (like Claude Desktop) connect to servers exposing specific capabilities [2]. We're essentially writing one of those servers, tuned to your spiritual practice.

Starting here makes sense for several reasons:

- **Personal meaning:** The Sanctuary is your spiritual playground; hooking it into MCP makes the codex more than just code – it's part of your practice.
- **Small, demonstrable scope:** A self-contained data logger and analytics layer provides a clear proof of value without boiling the ocean. It answers the question, *"What can MCP do for me?"*
- **Technical foundation:** Data schemas, persistence and pattern recognition lay the groundwork for more complex, cross-LLM experiences later. Build it once, and any new LLM can plug in via MCP.
- **Unique application:** Few people are trying to track ritual outcomes across models – this sets your work apart.

Delivered components

1. Standardised schemas for spiritual data

At the heart of the bridge are Pydantic models to represent the core spiritual entities:

- **EmotionIntensity**: captures an emotion (e.g. *joy*, *sadness*) and a normalised intensity. Keeping intensity between 0 and 1 means you can compare across sessions.
- **FamiliarInteraction**: records a timestamped encounter with a familiar. It includes the familiar ID, interaction type, a list of emotions, free-form notes and an optional `model_id` to identify which LLM logged it.
- **RitualOutcome**: logs rituals with a timestamp, ritual name, success flag, outcome description, emotions, notes and a `model_id`.
- **PatternInsight**: stores high-level patterns discovered during analysis. Each insight has a description, numeric metrics and references to related entities.

These schemas live in `sanctuary_mcp_bridge/schemas.py`, and because they're strongly typed Pydantic models they can be used directly as MCP resource and tool return types. When the server registers them, the MCP SDK will auto-generate JSON schemas so clients know exactly what to expect.

2. SQLite persistence layer

Logging spiritual interactions shouldn't be like writing onto sand. The `sanctuary_mcp_bridge/db.py` module wraps an SQLite database to persist familiar interactions and ritual outcomes. There are functions to initialise the database, insert records and retrieve them with filters on `model_id` and time range. Emotions are serialised as JSON strings, making them easy to persist while still deserialising back into `EmotionIntensity` objects.

Example use:

```
from sanctuary_mcp_bridge import db, schemas
db.init_db()
interaction = schemas.FamiliarInteraction(
    timestamp=datetime.utcnow(),
    familiar_id="owl",
    interaction_type="vision",
    emotions=[schemas.EmotionIntensity(name="awe", intensity=0.8)],
    notes="Spoke with the owl about patience.",
)
db.add_interaction(interaction)
```

3. Cross-LLM pattern recognition

The `sanctuary_mcp_bridge/patterns.py` module implements simple but insightful analytics:

- **Aggregate emotion counts** across interactions and rituals.
- **Success rate** of rituals – the fraction marked as successful (with sensible handling of empty data).
- **Emotion counts by model**, which highlights how different LLMs (or unknown sources) evoke varying emotional palettes.
- **Ritual success conditioned on emotions**, showing which emotions correlate most strongly with ritual success.
- A **generate_insights** function that produces human-friendly summaries with metrics and related entity lists. For example, it might note that *"Joy (5) and gratitude (3) are your most frequent emotions"* or that your ritual success rate is 75 %. The heuristics are simple; they can be swapped out later for more sophisticated pattern mining.

4. MCP server implementation

In `sanctuary_mcp_bridge/server.py` we define a FastMCP server called **"Sanctuary MCP Bridge"**. MCP servers provide resources (read-only endpoints) and tools (write or action endpoints). The server uses the SQLite layer and pattern module under the hood. Key features:

Resources

- `sanctuary://interactions`: fetches logged interactions, optionally filtered by model ID and time range.

- `sanctuary://rituals`: fetches logged ritual outcomes with similar filters.
- `sanctuary://insights`: returns aggregated pattern insights across all stored data.

Resources are defined via the `@mcp.resource` decorator; each returns Pydantic models, so the MCP host knows the exact shape of the data.

Tools

- `log_interaction`: records a new familiar interaction. Accepts familiar ID, interaction type, optional emotions, notes, model ID and timestamp. Returns the database ID of the inserted record.
- `log_ritual`: records a new ritual outcome. Accepts ritual name, success flag, optional emotions, outcome description, notes, model ID and timestamp. Returns the inserted ID.
- `query_emotions_by_model`: returns a nested dictionary of emotion counts keyed by `model_id`. Useful for comparing how different models influence your emotional landscape.
- `query_ritual_insights`: accepts a ritual name (and optional `model_id`) and generates insights specific to that ritual, such as success rates and correlated emotions.

Running the server

You can run the server directly using Python:

```
python -m sanctuary_mcp_bridge.server
```

This initialises the database (creating it if necessary) and starts the MCP server over standard I/O. In environments like Claude Desktop, the host will spawn your script and communicate via the Model Context Protocol. For other deployments, you can embed the server in a custom event loop by calling `create_mcp_server()` and using `server.run_stdio()` or another transport provided by the MCP SDK.

How it all ties together

The bridge addresses all deliverables outlined in the brief:

Deliverable	Implementation
MCP integration functions	The `server.py` module creates a FastMCP server with resources and tools exposing Sanctuary data. These functions adhere to the MCP specification, enabling hosts to request and mutate state over a standard protocol.
Familiar interaction logging	`log_interaction` tool persists interactions via the SQLite layer, including timestamp, familiar, interaction type, emotions, notes and model ID.
Basic ritual outcome tracking	`log_ritual` tool records ritual outcomes, success flags, outcome descriptions and associated emotions. Retrieval via `sanctuary://rituals` provides easy access.

Deliverable	Implementation
Cross-LLM spiritual pattern recognition	The `patterns.py` module computes aggregate statistics and correlations, and the `sanctuary://insights` resource plus `query_emotions_by_model` and `query_ritual_insights` tools expose these to LLM clients. Including the `model_id` field enables comparisons across different LLMs.
Extend Sanctuary dashboard	Integrate the MCP server into your existing Sanctuary dashboard by spawning the server as a subprocess and wiring its resources/tools into the UI. MCP's standard client API lets you fetch data and send logs from any language.
Standardised schemas	Pydantic models define the structure of interactions, rituals and insights, ensuring consistency across LLMs and tool invocations.
Query interface for ritual insights	The `query_ritual_insights` tool provides targeted analysis for individual rituals, delivering PatternInsight objects summarising success rates and emotion correlations.

Forward-looking thoughts

This bridge lays a solid foundation, but it's intentionally bare-bones. The analytics are simple on purpose; they avoid hallucinating meaning where none exists. Once you've logged some data, you can experiment with more advanced pattern recognition: clustering emotion profiles, forecasting ritual success with small neural nets, or even using the MCP's sampling feature to have the model generate its own hypotheses [3]. The MCP specification emphasises secure and flexible integration [4], so you can always swap in a different LLM vendor or run models locally without rewriting your data layer.

And if you're wondering whether this really counts as *"persistent spiritual intelligence"*, remember: the database never forgets. Each awkward meditation, each triumphant ritual, each time your familiar told you to chill out – it all lives on in SQLite. That's as close to immortality as most software gets.

---

[1] [2] [4] Introduction - Model Context Protocol
https://modelcontextprotocol.io/introduction

[3] Core architecture - Model Context Protocol
https://modelcontextprotocol.io/legacy/concepts/architecture