ADR-004-001-sqlite-extension-vs-mcp-server

ADR-004-001: SQLite Extension vs MCP Server for Trajectory Logging

Status: Accepted **Date**: 2025-11-16 **Deciders**: Research Team **Related**: SPEC-PPP-004 (Trajectory Logging & MCP Integration)

Context

PPP framework requires multi-turn conversation tracking to calculate R_{Proact} (proactivity) and R_{Pers} (personalization) scores. We need to store: - Trajectories (one per agent execution) - Turns (user prompt + agent response pairs) - Questions asked (with effort classification) - Preference violations

Two approaches were considered: 1. **SQLite extension**: Add tables to existing `consensus.db` managed by `consensus_db.rs` 2. **MCP server**: Create new `mcp-trajectory-logger` server (similar to consensus MCP pattern)

Decision

We will use **SQLite extension** (Option 1) - extend the existing `consensus_db.rs` with trajectory logging tables.

Rationale

Performance

Metric	SQLite Extension	MCP Server
Latency (single write)	<1ms	~5ms
Latency (batched)	**0.1ms**	~2ms
Throughput	10,000/sec	2,000/sec
Overhead	**5x faster**	Baseline

Key Finding: SQLite extension is **5x faster** than MCP server due to: - No JSON-RPC serialization overhead - No IPC (inter-process communication) - Direct in-process database access

Source: SPEC-PPP-004 findings.md, benchmark estimates

Complexity

Aspect	SQLite Extension	MCP Server
New Code	200 lines (schema + API)	500+ lines (server + client)
Dependencies	tokio-rusqlite (existing)	New MCP crate
Connection Mgmt	Reuse existing	New connection pool
Deployment	Single binary	Multiple processes
Debugging	Direct stack traces	Cross-process debugging

Winner: SQLite extension requires **60% less code** and simpler deployment.

Integration

SQLite Extension:

```
// Existing: consensus_db.rs
pub fn open_consensus_db() -> Result<Connection> {
    let conn = Connection::open(db_path)?;
    init_consensus_tables(&conn)?;
    init_trajectory_tables(&conn)?;  // ← NEW
    Ok(conn)
}

// Usage: Direct function calls
let traj_id = create_trajectory(&conn, spec_id, agent_name)?;
log_turn(&conn, traj_id, prompt, response)?;
```

MCP Server:

```
// New crate: mcp-servers/mcp-trajectory-logger
#[mcp_tool]
async fn log_turn(trajectory_id: i64, prompt: String, response: String) -> Result<()> {
    // ... MCP handler
}

// Usage: Requires MCP connection
let client = mcp_connection_manager.get_client("trajectory-logger").await?;
client.call_tool("log_turn", json!({...})).await?;
```

Winner: SQLite extension integrates with **zero additional infrastructure**.

Data Linkage

Requirement: Link trajectories to consensus artifacts via `run_id`.

SQLite Extension:

```sql
-- Single database, can use JOINs
SELECT t.spec_id, t.agent_name, c.content
FROM trajectories t
JOIN consensus_artifacts c ON t.run_id = c.run_id
WHERE t.spec_id = ?;
```

MCP Server:

```rust
// Must query both DBs separately, merge in application
let trajectory = mcp_client.get_trajectory(run_id).await?;
let artifact = consensus_db.get_artifact(run_id)?;
// Manual merge logic
```

Winner: SQLite extension enables **native SQL JOINs** across trajectory and consensus data.

Cost

Category	SQLite Extension	MCP Server
Development	12 hours	24 hours
Maintenance	Low (single codebase)	Medium (2 components)
Runtime	$0 (in-process)	$0 (local)

Winner: SQLite extension is **50% faster to implement**.

Downsides of SQLite Extension

1. **Couples trajectory logging to consensus DB**
   - Mitigation: Keep trajectory logic in separate module (`trajectory_logger.rs`)
   - Acceptable: Both are part of spec-kit infrastructure
2. **Schema migrations must coordinate**
   - Mitigation: Use versioned migrations (PRAGMA user_version)
   - Acceptable: Standard practice for SQLite apps
3. **Cannot reuse trajectory logger in other tools**
   - Impact: Low - trajectory logging is tightly coupled to PPP framework
   - Future: If needed, can extract to library crate (not MCP server)

When MCP Server Would Be Better

MCP server would be preferred if: - ✅ Trajectory logging needed by **external tools** (not just codex-tui) - ✅ Multi-language clients (Python, JS) need access - ✅ Network-based access required

Current Reality: - ❌ Only codex-tui needs trajectory logging - ❌ All code is Rust (no multi-language requirement) - ❌ Local-only access (no network requirement)

Conclusion: MCP server adds unnecessary complexity for current use case.

Consequences

Positive

1. ✅ **5x performance improvement** (<1ms vs ~5ms latency)
2. ✅ **50% less development time** (12 hours vs 24 hours)
3. ✅ **Simpler architecture** (single database, direct calls)
4. ✅ **Native SQL JOINs** for cross-table queries
5. ✅ **Easier debugging** (no cross-process complexity)

Negative

1. ⚠ **Tighter coupling** between trajectory logging and consensus DB
    - Mitigation: Modular code design, separate files
2. ⚠ **Harder to extract** if needed by other tools
    - Mitigation: Can refactor to library crate later (not MCP server)
3. ⚠ **Schema migrations** must coordinate with consensus schema
    - Mitigation: Versioned migrations with rollback support

Neutral

1. 📊 **Database file size** grows (trajectories + consensus in same file)
    - Expected: +10-20 MB/month for typical usage
    - Acceptable: Total <200 MB after 1 year

Migration Path

If MCP server becomes necessary in the future:

Phase 1 (Current): SQLite extension

```
codex-tui → consensus_db.rs → SQLite
```

Phase 2 (If needed): Extract to library crate

```
codex-tui → trajectory_logger (lib) → SQLite
other-tool → trajectory_logger (lib) → SQLite
```

Phase 3 (Only if external access needed): Add MCP wrapper

```
codex-tui → trajectory_logger (lib) → SQLite
external → MCP server → trajectory_logger (lib) → SQLite
```

Key: Library crate is sufficient for code reuse - MCP server only needed for network/multi-language access.

References

1. SPEC-PPP-004 findings.md - Performance benchmarks (SQLite vs MCP)
2. SPEC-PPP-004 comparison.md - Integration strategy comparison
3. consensus_db.rs (existing) - Database connection management pattern
4. mcp_connection_manager.rs (existing) - MCP overhead

measurement

Decision Drivers

Driver	Weight	SQLite Extension	MCP Server
Performance	30%	✅ 5x faster	❌ Baseline
Complexity	25%	✅ Simpler	❌ More complex
Integration	20%	✅ Direct	⚠ Indirect
Future Flexibility	15%	⚠ Coupled	✅ Reusable
Development Speed	10%	✅ 50% faster	❌ Baseline

Total Score: SQLite Extension **85%**, MCP Server **40%**

Winner: SQLite Extension by significant margin.