Different phases

here are research prompts for every Phase-1 sub-mission (copy-paste into your web agent like Atlas/Perplexity). Each block returns a JSON ARRAY so we can fold results straight into the plan.

———

1A — Message Schema & Validation Standards

Goal: Lock a durable message contract (TS + JSON Schema 2020-12) and pick a validator.

Return format (per item):

```
{
  "source_url": "",
  "source_title": "",
  "last_updated_date": "YYYY-MM-DD or 'unknown'",
  "answers": {
    "q1": "",  // Recommended JSON Schema draft + must-have keywords for our use (enum, additionalProperties, formats)
    "q2": "",  // Best validator for Node/TS (Ajv vs Valibot/Zod+zod-to-json-schema) + perf notes
    "q3": "",  // How to keep TS types and JSON Schema in sync (codegen or single-source-of-truth)
    "q4": ""   // Validation error reporting patterns suitable for CI/logs
  },
  "snippets": ["<=25 words, with date if visible"],
  "code_samples": [
    { "lang": "json", "purpose": "message.schema.json (minimal)", "body": "<=40 lines" }
  ]
}
```

Suggested queries:

- site:json-schema.org 2020-12 keywords additionalProperties enum
- ajv 2025 performance typescript error formatting
- zod to json schema 2025 best practice
- keep typescript types and json schema in sync codegen

———

1B — Debate Loop Patterns & Convergence Rules

Goal: Inform a deterministic propose→implement→critique→verify loop with clear exits.

Return format:

```
{
  "source_url": "",
  "source_title": "",
  "last_updated_date": "",
  "answers": {
    "q1": "",  // Common multi-agent debate patterns for software tasks
    "q2": "",  // Convergence/termination rules that avoid infinite loops
    "q3": "",  // Minimal rationale structs (reasons[], evidence[], risks[]) that add value
    "q4": ""   // Anti-patterns (non-determinism, hidden state) to avoid in tests/CI
  },
  "snippets": ["<=25 words"]
}
```

Suggested queries:

- multi-agent debate pattern convergence rules software engineering
- deterministic agent loop best practices testing
- critic builder verify loop research

———

1C — In-Memory Event Bus Design (Deterministic)

Goal: Confirm a simple, testable bus (publish/history) and when to add IDs/causality.

Return format:

```
{
  "source_url": "",
  "source_title": "",
  "last_updated_date": "",
```

```
  "answers": {
    "q1": "",  // Minimal event-bus API for deterministic tests (publish, history, clear)
    "q2": "",  // When to add messageId, parentId, turn, timestamp
    "q3": "",  // Ordering guarantees and how to assert them in unit tests
    "q4": ""   // Memory-safety and growth concerns (bounded logs) for local runs
  },
  "snippets": ["<=25 words"]
}
```

Suggested queries:

- typescript in-memory event bus pattern unit test deterministic
- event sourcing minimal log ordering guarantees test

1D — Mock Diff & Patch Conventions

Goal: Choose a friendly, testable diff format for Phase-1 (no real patching yet).

Return format:

```
{
  "source_url": "",
  "source_title": "",
  "last_updated_date": "",
  "answers": {
    "q1": "",  // Unified diff basics relevant to tests (headers, context lines)
    "q2": "",  // Smallest viable diff string shape for readability in CLI/logs
    "q3": "",  // How to snapshot diff in tests cleanly (stability tips)
    "q4": ""   // When to graduate to patch application (Phase 3 prep)
  },
  "snippets": ["<=25 words"]
}
```

Suggested queries:

- git unified diff format explained
- node snapshot testing diff stability vitest

1E — CLI UX & Exit Codes

Goal: Ensure CLI prints diff and then CONVERGED|FAILED, with proper exit codes.

Return format:

```
{
  "source_url": "",
  "source_title": "",
  "last_updated_date": "",
  "answers": {
    "q1": "",  // POSIX/Node conventions for exit codes (0 success, 1 failure)
    "q2": "",  // Argument parsing minimalism vs libraries (commander/yargs) for PoC
    "q3": "",  // Logging do/don't (stderr vs stdout) for CI consumption
    "q4": ""   // How to keep output stable for scripts parsing status lines
  },
  "snippets": ["<=25 words"]
}
```

Suggested queries:

- node cli exit codes best practices
- stdout vs stderr conventions ci pipeline

1F — Testing & Determinism (Vitest)

Goal: Lock patterns for stable tests and eliminate flakiness.

Return format:

```
{
  "source_url": "",
  "source_title": "",
  "last_updated_date": "",
  "answers": {
```

```
    "q1": "",  // Vitest config for Node/ESM + TypeScript (noEmit typecheck strategy)
    "q2": "",  // Techniques to avoid time/random flakiness (fake timers, fixed seeds)
    "q3": "",  // Snapshot test hygiene (when to use, when to avoid)
    "q4": ""   // Fast CI settings (coverage tradeoffs for PoC)
  },
  "snippets": ["<=25 words"],
  "code_samples": [
    { "lang": "ts", "purpose": "vitest.config minimal", "body": "<=30 lines" }
  ]
}
```

Suggested queries:

- vitest typescript esm config 2025
- vitest fake timers deterministic tests
- snapshot testing best practices vitest

1G — Static Analysis & Type Safety

Goal: Validate minimal TS/ESLint/Prettier config for a PoC that won't slow us down.

Return format:

```
{
  "source_url": "",
  "source_title": "",
  "last_updated_date": "",
  "answers": {
    "q1": "",  // TS compiler options that catch real bugs (strict, noEmit typecheck)
    "q2": "",  // ESLint rules that add signal without noise in early phases
    "q3": "",  // Prettier interplay; how to avoid formatter-vs-linter conflicts
    "q4": ""   // Recommended Node version & reasons (perf, ESM stability)
  },
  "snippets": ["<=25 words"]
}
```

Suggested queries:

- typescript strict options recommended 2025
- eslint typescript recommended config noise reduction
- node 20 vs 22 esm stability 2025

1H — CI Enhancements (Cache & Speed)

Goal: Make CI fast and reproducible for Phase-1.

Return format:

```
{
  "source_url": "",
  "source_title": "",
  "last_updated_date": "",
  "answers": {
    "q1": "",  // Node dependency caching in GitHub Actions (setup-node cache)
    "q2": "",  // When to run typecheck vs tests; order for fastest feedback
    "q3": "",  // Pinning actions by SHA and least-privilege permissions
    "q4": ""   // Matrix (Node versions) — do we need it for PoC?
  },
  "snippets": ["<=25 words"],
  "code_samples": [
    { "lang": "yaml", "purpose": "workflow snippet", "body": "<=40 lines" }
  ]
}
```

Suggested queries:

- site:docs.github.com setup-node cache npm ci
- github actions least privilege permissions pin by sha

1I — Logging & Telemetry Stubs

Goal: Decide minimal fields to log now that won't change later (to ease Phase 10).

Return format:

```
{
  "source_url": "",
  "source_title": "",
  "last_updated_date": "",
  "answers": {
    "q1": "",  // Minimal log fields for a run: task, turn, role, type, status
    "q2": "",  // JSON-lines vs pretty logs for CI/dev
    "q3": "",  // Redaction rules (no secrets) even in mocks
    "q4": ""   // Trace IDs and correlation (lightweight now)
  },
  "snippets": ["<=25 words"]
}
```

Suggested queries:

- json logging best practices node 2025
- structured logging redaction guidelines

1J — Security-in-Dev (Mocks)

Goal: Ensure mocks never leak secrets and follow secure-by-default patterns.

Return format:

```
{
  "source_url": "",
  "source_title": "",
  "last_updated_date": "",
  "answers": {
    "q1": "",  // Safe logging guidelines (no env echo; redact tokens)
    "q2": "",  // Test data handling (fixtures with fake values, no accidental secrets)
    "q3": "",  // CI secrets usage policies in a PoC repository
    "q4": ""   // License and attribution notes for code snippets/templates
  },
  "snippets": ["<=25 words"]
}
```

Suggested queries:

- security best practices logging secrets redaction
- github actions secrets misuse prevention 2025

1K — Budget Trace Semantics (Stub Alignment with 0E)

Goal: Keep budgetTrace fields meaningful now, easy to wire later.

Return format:

```
{
  "source_url": "",
  "source_title": "",
  "last_updated_date": "",
  "answers": {
    "q1": "",  // Which fields to include now (inputTokens, outputTokens) to match 0E tools
    "q2": "",  // How to record turn-level cumulative totals (schema shape)
    "q3": "",  // How CLI should behave (no provider calls; zeros acceptable)
    "q4": ""   // Migration strategy when real adapters land (Phase 2)
  },
  "snippets": ["<=25 words"]
}
```

Suggested queries:

- token usage fields api usage input_tokens output_tokens openai anthropic docs
- budget accounting per turn schema design

Global rules for all missions

- **Prefer official docs and well-known libraries' docs; avoid unverifiable blogs.**

- **Include explicit dates and versions where possible.**
- **Today’s date for reference: 2025-10-27 (Africa/Johannesburg).**

Send these back one sub-mission at a time; I’ll fold each into the Phase-1 plan and repo (schema, tests, CLI, CI) as we go.