SEAGLASS FOUNDRY LLC

ForgeMind™

Technical Brief

VERSION 1.0 · JUNE 2026

The GIS Intent Compiler

Natural language in. Correct geospatial pipelines out. Or an honest refusal.

SEAGLASS FOUNDRY LLC

Executive Summary

ForgeMind™ is the natural-language control plane for GPU-accelerated geospatial analysis. It accepts a spoken or typed request — “map the slope around Objective Bravo,” “where can we land outside the enemy viewshed,” “rebuild the elevation picture after the objective displaced” — and turns it into a validated, executed geospatial pipeline against ForgeGIS, or, when the request cannot be honored correctly, into an explicit refusal rather than a confident guess.

ForgeMind is built on a deliberate architectural choice that sets it apart from conventional agent frameworks. Where a typical LLM agent hands the whole problem to a model in a tool-calling loop and hopes the model strings the right calls together, ForgeMind narrows the language model to the one thing it is reliable at — understanding what the operator wants — and hands everything that must be correct to deterministic machinery. The result is a system that is fast, auditable, and, most importantly, does not fabricate. When ForgeMind cannot map a request to a valid operation with adequate confidence, it says so. This property — refusal by construction — is what makes it deployable in regulated and mission environments where a plausible-but-wrong answer is worse than no answer.

Zero **confident-wrong results** across the full test campaign	**6,696** **conversation turns validated** 79 multi-turn conversations	**97** **voice-routable capabilities** exercised end to end
~25× **lower LLM cost** vs. agent-loop baseline	**94%** **compiler eval pass rate** intent → validated pipeline	**LLM** **agnostic** no vendor lock-in

Figures from the June 2026 validation campaigns. See Validation and Coverage.

ForgeMind is LLM-agnostic at the language layer: the provider and model sit behind one abstraction, so a deployment is never locked to a single vendor or a single availability domain. Claude and OpenAI ship as built-in integrations, and the same abstraction is the integration point for other providers and self-hosted models. It carries a semantic memory that lets a deployment get measurably better with use — not by retraining a model, but by accumulating known-good workflow shapes that future requests retrieve and reuse. And it speaks the Model Context Protocol natively in both directions: it consumes MCP tools (ForgeGIS, catalog services, a map client) and it can itself be exposed as a single MCP tool that a higher-level agent delegates whole geospatial workflows to.

Seaglass Foundry is seeking strategic partners and distributors positioned to bring ForgeMind, alongside the ForgeGIS compute substrate it drives, to defense, intelligence, climate, and enterprise markets at scale.

What ForgeMind Is

ForgeMind is the orchestration and intent layer that sits between a human (or another agent) and ForgeGIS, Seaglass Foundry's GPU-accelerated geospatial library and MCP server. It is the component that decides what to run, binds the request to real data, dispatches the work, and narrates the result — turning an informal request into a correct, reproducible operation against the ForgeGIS engine.

ForgeMind and ForgeGIS are built for each other. ForgeMind's value is bounded by the engine underneath it, and ForgeGIS is an exceptional engine to be paired with: a 239-operation GPU-accelerated catalog whose data plane stays resident on the device across a multi-step pipeline, so a chain of operations runs at machine speed rather than round-tripping through disk between stages. That is precisely the substrate a natural-language control plane needs. The reason it is worth letting an operator ask for a viewshed, a slope mask, and a least-cost path in one breath is that ForgeGIS can actually execute that composed work fast enough to feel conversational. A control plane on top of a slower, CPU-bound, process-per-step engine would expose the latency that ForgeGIS eliminates. ForgeMind is designed around ForgeGIS's strengths — its GPU-resident pipelines, its typed operation surface, its deterministic and reproducible outputs — and inherits them.

In a reference deployment, ForgeMind pairs ForgeGIS with a catalog of available datasets and a map client (ForgeGIS Studio or a virtual globe) into a single conversational surface. ForgeMind speaks the Model Context Protocol, so the catalog and map-client roles are pluggable; the compute engine it is purpose-built to drive is ForgeGIS.

The defining commitment is the inverse of the prevailing agent design. A conventional agent treats the language model as a general-purpose planner and lets it author tool chains directly — which means every wrong parameter, every hallucinated step, and every misrouted call lands in production as a confident-looking result. ForgeMind treats the language model as an intent classifier and parameter extractor only. The structure of the pipeline — which operations run, in what order, with what types — is owned by deterministic code that validates every step before anything executes. The model proposes; deterministic machinery disposes.

This division of labor is the structural reason ForgeMind can make a guarantee a conventional agent cannot: a request either compiles to a valid, type-checked pipeline, or it is refused. There is no middle path where the system runs a plausible-but-incorrect operation and reports success.

Who it is for

ForgeMind serves two audiences whose needs are converging. The first is operators and analysts who want to ask for geospatial products in plain language and get correct results without learning a query syntax or a desktop GIS. The second is system builders who want to expose geospatial analysis to AI agents and automation, and who need that exposure to be trustworthy — deterministic where it matters, honest when it cannot proceed, and cheap enough to run at volume.

Architecture

ForgeMind is organized around a small number of architectural commitments that distinguish it from agent-loop frameworks. The internal mechanism that implements them — the GIS Intent Compiler — is proprietary; what follows is the externally observable contract each commitment produces.

The language model does only what language models do reliably

Across the design, the language model is confined to two tasks: classifying what the operator is asking for, and extracting the parameters that go with it — an area of interest, an observer point, a slope threshold, a radius. It does not author the pipeline. The shape of the work — the operations and how they connect — is determined by deterministic machinery. The practical consequence is that the same request produces the same pipeline every time, with no hallucinated parameters, no operation-arity slips, and no run-to-run drift. This is the property regulated and audited work depends on, and it is not achievable when a model writes tool calls freely.

Validation before execution — refusal by construction

Every candidate pipeline is checked against a type system and a set of operational guards before any work is dispatched to the compute engine. A request that does not resolve to a valid operation, that lacks the parameters it needs, or that cannot be bound to real data is refused with a specific, machine-readable reason — never run speculatively. Confidence is treated as a first-class, calibrated signal rather than an afterthought: a request the system is not sure it understands is routed to clarification or refusal, not to a guess. Across the full validation campaign, this produced zero confident-wrong results — the system never returned a fabricated or misrouted answer as though it were correct.

Cost engineering: the right model for each job

Not every step in understanding a request needs the most capable — and most expensive — model. ForgeMind routes cheap, high-volume classification and narration work to a fast, low-cost model tier, and reserves the more capable tier for the genuinely hard reasoning. Because deterministic code owns everything in between, the expensive model is invoked sparingly. The measured effect is roughly a 25× reduction in language-model cost relative to a conventional agent-loop approach to the same workflows, without sacrificing correctness — the deterministic layer, not a bigger model, is what guarantees the result.

LLM-agnostic by design

ForgeMind runs against any supported language model behind a single internal abstraction. The provider and the specific model are configuration, not architecture: a deployment can switch models to follow cost, capability, availability, or procurement constraints — including for continuity during a provider outage — without code changes. Claude and OpenAI ship as built-in integrations, each tested against realistic request and response shapes, and the same abstraction is the integration point for

additional providers and self-hosted or on-premise models in environments with their own model-hosting requirements.

Memory that improves a deployment with use

ForgeMind carries a semantic memory layer backed by local storage and on-device embeddings. Successful workflows, and corrections to unsuccessful ones, accumulate as retrievable exemplars. The first time an operator asks for an unfamiliar kind of analysis in a region, the system works from first principles; after a few successful runs, similar future requests retrieve the known-good shape and converge faster. Crucially, this learning happens through accumulated retrieval context, not model fine-tuning — so it is inspectable, clearable, and never silently bakes a bad pattern into a model's weights. Operators can inspect, delete individual entries, or clear the store entirely.

Native MCP, in both directions

ForgeMind is an MCP citizen on both sides of the protocol. As a client, it dispatches to MCP tool servers — the ForgeGIS compute surface, a dataset catalog, a map client — fanning calls out in parallel, translating errors, and guarding each namespace behind a circuit breaker so one failing backend degrades gracefully instead of taking down the whole surface. As a server, ForgeMind exposes a single high-level tool that a higher-order agent can call to delegate an entire geospatial workflow as one unit. It also consumes a resource-estimation contract from the compute layer, so it can size a job before dispatching it and decline or downscale work that would exceed a hardware budget rather than failing blind.

Named scene entities: geometry the agent can reason about

ForgeMind defines a wire contract for named geometry a user places on a map — a dock, a route, an objective. The map client owns and persists the geometry; ForgeMind references it by a stable handle and hydrates the real vertices on demand from the catalog when an operation needs them. Geometry is never re-derived from a bounding box or re-parsed by the model. The design is deliberately fail-closed: if the true geometry is not reachable, the operation defers and asks rather than emitting a plausible-but-wrong product — the same refusal discipline that governs the rest of the system, applied to spatial references.

Capabilities

ForgeMind makes the ForgeGIS analytic catalog reachable in natural language. ForgeGIS provides 239 GPU-accelerated operations across terrain, hydrology, visibility and RF, spectral and machine-learning analysis, spatial geometry and topology, bathymetry and ocean, routing, point cloud, and temporal analysis — the complete catalog described in the ForgeGIS Technical Brief. ForgeMind exposes a substantial and growing portion of that catalog by intent, anchored on a deterministic core of more than 130 routed capabilities and extended across the long tail by adapting known-good reference workflows. The validation campaign exercised 97 of these capabilities end to end. Coverage is expanded deliberately, capability by capability, as each is validated — ForgeMind only advertises what it can run correctly, never the full op list speculatively. For any request it accepts, an operator names the product they want and ForgeMind selects, parameterizes, validates, and runs the operation on the GPU-resident engine, composing several operations into one pipeline where the request calls for it.

Conversational, multi-turn workflows

ForgeMind holds context across a conversation. An operator can establish a subject, then refine it across turns — change a parameter, mask a result, re-run with a new value, or reference a product from an earlier turn — and the system threads the prior result through rather than starting over. It resolves deictic references (“re-run that with a wider radius,” “mask it to slopes under seven degrees”) and named anchors (“the first objective,” “Objective Bravo”) against a ledger of what has been established in the session.

The same discipline governs the negative cases. A reference that cannot be resolved — a name never defined, an ambiguous “the other one,” a follow-up with no antecedent — is refused with a specific reason rather than guessed at. In validation, these unresolved-reference cases were refused correctly rather than answered wrongly.

Representative request patterns

These patterns run reliably end to end against a ForgeGIS-backed deployment:

- **Direct analysis.** “Compute viewshed from 34.5N 69.2E with a 10 km radius” — explicit-coordinate raster analysis.
- **Discovery into analysis.** “What elevation datasets do we have for this region? Pick the highest-resolution one and run a slope analysis” — a catalog query chained into a compute operation.
- **Site suitability.** “Where can we land outside the enemy viewshed?” — a multi-operation suitability workflow combining line-of-sight exclusion with terrain constraints.
- **Scene-aware.** “Mark Objective Bravo here and map the slope around it,” then later “rebuild the slope picture — it displaced to a new grid reference” — named-anchor placement and refinement across turns.

- **Visualization.** “Load the result onto the globe as a red, 50% transparent layer” — chaining a compute product into a map-client display.

Honest answers and graceful degradation

When a request is under-specified, out of scope, or cannot be bound to data, ForgeMind returns a clear, specific response — a request to clarify, a statement that no data covers the area, or a structured refusal — instead of a fabricated result. When a downstream service is slow or unavailable, the affected capability fails cleanly while the rest of the surface keeps serving. These behaviors are not error handling bolted on after the fact; they are the same correctness discipline that governs the successful path.

Validation and Coverage

ForgeMind is validated along two complementary axes: a large multi-turn conversation campaign that measures end-to-end behavior as an operator would experience it, and a focused compiler-evaluation campaign that measures the core intent-to-pipeline transformation in isolation. Together they cover both “does the whole system behave correctly in conversation” and “is the engine at the center sound.”

Conversation campaign

The conversation campaign exercises ForgeMind across realistic, multi-turn operator sessions — establishing subjects, refining them, chaining discovery into analysis, and probing the negative cases where a careless system would guess.

Measure	Result
Conversations	79 multi-turn sessions
Total turns	6,696
Voice-routable capabilities exercised	97
Confident-wrong results	0 — across every run

The headline result is the absence of a category of failure: across the entire campaign, ForgeMind never returned a confident answer that was wrong. Requests it could honor were honored; requests it could not were refused or routed to clarification. That is the property refusal-by-construction is designed to produce, measured at conversation scale.

Compiler evaluation campaign

The compiler campaign measures the core transformation — operator intent to a validated, executable pipeline — in isolation, including the hardest multi-turn cases: continuation across turns, parameter overrides, named-anchor resolution, and the negative cases that must refuse. The shipped engine clears its acceptance gates: roughly a 94% pass rate on the intent-to-validated-pipeline evaluation, with first-try

validity and after-retry validity both meeting their thresholds, and with the negative cases refusing correctly rather than emitting a wrong pipeline.

Two design properties show up directly in these results. First, invalid pipelines are caught before execution, not after — a malformed candidate is rejected and corrected, or refused, never dispatched. Second, the cost of getting there is low: by confining the expensive model to the one hard step and letting deterministic code own the rest, the evaluation runs at roughly a 25× lower language-model cost than an agent-loop approach to the same work.

What is and is not guaranteed

ForgeMind’s correctness boundary is its own: intent resolution, validation, refusal discipline, dispatch, memory, and the HTTP and MCP surfaces. The correctness of the underlying geospatial computations is owned by the compute backend (ForgeGIS), validated separately against its own oracles, and reproduced faithfully because ForgeMind dispatches to the same implementations whether a human or an agent invokes them. ForgeMind does not attempt to detect model hallucination by inspecting model output; it makes hallucination structurally unable to reach execution by never letting the model author the pipeline in the first place. That distinction — preventing the failure rather than policing it — is the core of the design.

Deployment and Integration

ForgeMind ships as a single self-contained Java application. It runs in either of two shapes from the same artifact: an HTTP service exposing a small REST surface for interactive and streaming use, or an MCP stdio server that presents ForgeMind as a single delegatable tool to a higher-level agent host.

Runtime surface

In HTTP mode, ForgeMind serves a buffered request endpoint and a streaming (Server-Sent Events) endpoint that surfaces progress — each model round trip, each tool call, each tool result — as it happens, which a chat UI or progress widget can render live. A feedback endpoint lets a caller report whether an answer was accepted, corrected, or abandoned; that signal is what sharpens the memory layer over time. Auth (bearer token), rate limiting, request-size caps, and a concurrency cap are all built in and configurable.

Backend integration

ForgeMind spawns and supervises its MCP backends — the ForgeGIS compute engine, a dataset catalog, and a map client — as local subprocesses communicating over stdio, with health checks, graceful shutdown, and per-namespace circuit breakers. ForgeGIS is the engine ForgeMind is purpose-built to drive; the catalog and map-client roles are pluggable behind the protocol. A reference deployment shares a single catalog surface between ForgeMind and the map client, so a computed product registered by ForgeMind resolves unchanged when the map client displays it — compute, catalog, and display lined up as one coherent system.

Model configuration

The language model is set by configuration, with Claude and OpenAI available as built-in integrations and the same abstraction open to other providers and self-hosted models. A deployment can run a cost-optimized profile for high-volume, simple queries, or a maximum-capability profile for ambiguous, long-horizon analytical work, by changing configuration rather than code. Pre-staging credentials for an alternate provider turns an outage response into a single configuration change and a restart, with accumulated memory preserved across the switch.

Fit for regulated and air-gapped environments

ForgeMind is a pure-JVM application with local-disk persistence and no required external services beyond the language provider endpoint. Its memory store is a single local database file; its backends run as local subprocesses over stdio rather than the network. For environments with their own model-hosting arrangements, the provider abstraction is the single integration point. The same correctness and refusal guarantees hold regardless of where it is deployed — they are properties of the architecture, not of a particular environment.

Licensing and Contact

Licensing

ForgeMind license terms are being finalized and will be published prior to general availability. Organizations with specific licensing requirements are invited to contact Seaglass Foundry directly.

Partnership and distribution

Seaglass Foundry is seeking strategic partners and distributors positioned to bring ForgeMind and the ForgeGIS compute substrate to defense, intelligence, climate, and enterprise markets at scale. Inquiries from organizations with relevant distribution, integration, or capital-deployment capacity are welcome.

Contact

rich@seaglassfoundry.com · **seaglassfoundry.com**

About Seaglass Foundry

Seaglass Foundry LLC is an independent geospatial software company based in Panama City Beach, Florida, building the compute substrate for AI-native geospatial analysis. Its product line includes ForgeGIS (GPU-accelerated geospatial library and MCP server), ForgeMind (the GIS Intent Compiler and natural-language control plane), and SwingToPDF™ (vector PDF export for Java Swing applications, in production on Maven Central).
