ENCAPURE

Context-Aware Semantic Tool Search Engine

Find the right tool from thousands in under 70ms.

Built for MCP environments where AI agents need instant, accurate tool discovery at scale. Powered by a two-stage retrieval pipeline with quantized ONNX models in Rust.

Rust | Axum | ONNX Runtime | Tokio | INT8 Quantized

Danny Avraham
Backend Engineer | dannyavrs45@gmail.com | github.com/dannyavrs

The Problem

As AI agents grow more capable, they need access to thousands of tools across cloud providers, databases, messaging platforms, DevOps systems, and more. The MCP ecosystem now hosts 1,000+ tools, and that number is growing rapidly.

> **The bottleneck:** Current tool selection approaches rely on LLM function calling, consuming 3-5 seconds per decision and burning through token budgets. At scale, this becomes the primary latency and cost driver in agent pipelines.

Existing solutions either use naive keyword matching (MCP registries), single-stage vector search (Semantic Router), or expensive LLM-based routing (LangChain, CrewAI, AutoGen). None combine sub-100ms latency with high accuracy at 1,000+ tool scale.

The Solution: Encapure

Encapure is a high-performance, context-aware semantic tool search engine purpose-built for MCP environments. It uses a two-stage retrieval pipeline with quantized transformer models running locally via ONNX Runtime, all implemented in Rust for maximum throughput.

70ms	30+	98%	1,000
Avg. Latency (Single Mode)	Requests/sec (Concurrent)	Accuracy (50 Test Cases)	MCP Tools Indexed

Architecture: Two-Stage Retrieval Pipeline

Encapure combines the speed of bi-encoder retrieval with the precision of cross-encoder reranking, achieving both fast response times and high-quality results.

INPUT		STAGE 1		STAGE 2		OUTPUT
Query + Agent Context	>> >	**Bi-Encoder** all-MiniLM-L6-v2 INT8 Quantized	>> >	**Cross-Encoder** BGE-Reranker-v2-M3 INT8 Quantized	>> >	Ranked Results
		~5ms		**~60ms**		**Top K**
		Cosine similarity over 1,000 embeddings Returns top 20		Full attention on 20 (query, tool) pairs Precise relevance scoring		Best tools for the agent's context

Context-Aware Search

The core differentiator: the same query returns different tools depending on the agent's context. When an agent_description is provided, the query is augmented to bias both retrieval stages toward domain-relevant tools.

Query	Agent Context	Top Results
"send message"	None	send_message, send_sms, send_notification
"send message"	Slack bot	send_slack_message, send_slack_dm
"send message"	Email agent	send_email, send_email_notification
"create server"	AWS engineer	create_ec2_instance, launch_aws_vm
"create server"	Azure engineer	create_azure_vm, deploy_azure_instance

Competitive Advantage

No existing solution combines sub-100ms latency, high concurrency, semantic two-stage retrieval, quantized local models, and 1,000+ tool scale optimization. Here is how Encapure compares:

Capability	Encapure	LLM Agents (LangChain, CrewAI)	Semantic Router (Aurelio)	MCP Registry (Official)
Latency	**~70ms**	3,000-5,000ms	~100ms	Web-scale
Throughput	**30+ req/s**	< 5 req/s	~15 req/s	N/A
Retrieval Method	**Two-stage (bi + cross-encoder)**	LLM function calling	Single-stage vector search	Keyword filtering
Context Awareness	**Full semantic**	Via prompt	Route-based	None
Tool Scale	**1,000+ optimized**	~50-200 (context limit)	Hundreds	Registry-wide
Local Deployment	**Yes (Rust + ONNX)**	Cloud API required	Partial	Cloud only
Cost per Query	**~$0 (local)**	$0.01-0.10 (tokens)	~$0 (local)	Free

50x faster than LLM-based tool selection. Where LangChain agents spend 3-5 seconds choosing the right tool (and consume expensive tokens doing so), Encapure delivers the same decision in 70ms with 98% accuracy — entirely on local hardware with zero API costs.

Engineering Excellence

Encapure demonstrates advanced backend engineering across systems programming, ML inference optimization, and production-grade API design.

Systems-Level Performance Optimization

- **Lock-free session pooling:** Multiple ONNX Runtime sessions managed via atomic round-robin index (Vec<UnsafeCell<Session>>) — zero mutex contention under concurrent load
- **Semaphore-based admission control:** tokio::sync::Semaphore prevents CPU oversubscription with configurable permit counts, maintaining the invariant PERMITS x INTRA_THREADS <= cores
- **INT8 quantization:** Both models quantized to INT8 via ONNX, reducing model size by ~4x and inference latency by ~2x versus FP32, with negligible accuracy loss
- **Pre-computed embeddings cache:** Binary-serialized embedding vectors loaded at startup from .encapure/embeddings.bin — instant cold start after first indexing

Production-Grade API Design

- **Axum + Tokio:** Fully async HTTP server with structured routing, graceful shutdown, and configurable timeouts
- **Prometheus metrics:** Built-in /metrics endpoint for observability and monitoring
- **Health and readiness probes:** Kubernetes-compatible /health and /ready endpoints for container orchestration
- **Comprehensive error handling:** Custom error types with HTTP status code mapping for clean API responses
- **Environment-driven configuration:** All settings via environment variables with sensible defaults and preset operating modes

Rigorous Testing and Benchmarking

- **50-case accuracy suite:** Automated tests covering context switching, DevOps, databases, cloud storage, monitoring, notifications, git operations, and edge cases (98%+ pass rate)
- **k6 load testing:** Simulates hundreds of concurrent virtual users with ramp-up profiles to find the server's breaking point
- **Latency-coded demos:** Interactive accuracy demos with color-coded latency indicators (green <100ms, yellow <200ms, red >200ms)

Backend Engineering Skills Demonstrated

Encapure is a showcase of production-level backend engineering. Here are the core competencies this project demonstrates:

Skill Domain	Technologies / Concepts	Evidence in Encapure
Systems Programming	Rust, unsafe code, memory management, zero-cost abstractions	Lock-free session pool with UnsafeCell, atomic operations, efficient binary serialization
ML Infrastructure	ONNX Runtime, model quantization, embedding pipelines, tokenization	INT8 quantized bi-encoder + cross-encoder, custom tokenization layer, embedding cache system
Async / Concurrency	Tokio, async/await, semaphores, thread budgeting	Configurable concurrency model with admission control, multiple operating mode presets
API Design	REST APIs, Axum, structured routing, error handling	Clean endpoint design, Prometheus metrics, health/readiness probes, graceful shutdown
Performance Engineering	Profiling, load testing, latency optimization, k6	Sub-100ms p50, 30+ req/sec throughput, automated benchmarks with k6
Information Retrieval	Semantic search, reranking, bi-encoder/cross-encoder architecture	Two-stage pipeline with cosine similarity retrieval + full-attention reranking
DevOps / Deployment	Configuration management, containerization readiness, CI/CD	Environment-based config, K8s-compatible probes, cross-platform build scripts

Performance Benchmarks

Single Mode (Low Latency)

Metric	Target	Achieved
Average Latency	< 100ms	**~70ms**
Accuracy (11 tests)	> 90%	**90%+**
Accuracy (50 tests)	> 95%	**98%+**

Concurrent Mode (High Throughput)

Metric	Target	Achieved
Average Latency	< 500ms	**~350ms**
Throughput	> 20 req/s	**30+ req/s**
P95 Latency	< 600ms	**< 500ms**
Success Rate	> 99%	**99%+**

Let's Connect

Encapure represents the intersection of systems engineering, machine learning infrastructure, and production API design — built as a solo project from architecture to load testing. I'm excited to bring this level of engineering rigor to a team solving hard backend problems.

Danny Avraham
Backend Engineer
dannyavrs45@gmail.com | github.com/dannyavrs