

SESHA SAI VAMSI M

vamsi.marella027@gmail.com | +91 75500 27299 | [LinkedIn](#) | [GitHub](#) | [Portfolio](#)

TECHNICAL SKILLS

Languages: Python, TypeScript, JavaScript, C++, SQL, Solidity

Backend: FastAPI, Django, Flask, Node.js, Celery, WebSockets, SQLAlchemy (ORM)

Data/Infra: PostgreSQL, Redis, DynamoDB, Timestream, Parquet, AWS, GCP, Docker, Nginx, Caddy, Prometheus, Linux

EDUCATION

B.Tech, Computer Science (Cyber-Physical Systems), VIT Chennai

Sep 2021 – Jul 2025

EXPERIENCE

Founding Engineer, OneCent Labs

May 2026 – Sept 2026

- Built and run a real-time trade routing and settlement engine that has processed \$1.6M+ in live volume, averaging 386 successful settlements/day.
- Built the core bidding pipeline around a 4.85s hard deadline, maintaining a 99.3% valid-response rate for requests coming every 6 seconds.
- Cut worst-case response time 7× from 20.3s to 2.9s by reducing 780 candidate routes to 17 before network calls, running survivors across 32 workers, and fixing async and connection-pool bottlenecks.
- Built route search across 20K+ assets and 38K+ liquidity pools using bounded multi-hop search, liquidity-aware ranking, fanout limits, and request-size-aware pruning to stay within the bidding budget.
- Built pricing and indexing systems with path simulation, outlier filtering, confidence scoring, batched RPC discovery, and zero-downtime repricing of 7,900+ records.
- Integrated 74 protocol adapters across 91 chain-specific setups; reverse-engineered undocumented pricing logic and validated results to 1e-4 tolerance against forked-chain execution.
- Hardened production with async connection pooling, caching, request deduplication, circuit breakers, rate limits, retries, and backoff; also reduced 240MB/hr of duplicate logs and added request tracing, failure capture, and replay tooling.
- Root-caused silent production failures across APIs, contracts, DNS, and networking, including 200 OK responses that dropped submissions and TCP connects that consumed the full execution deadline; owned deployments and monitoring across AWS/GCP.

Backend Engineer, GlueX Protocol

Jan 2025 – Feb 2026

- Built and owned an async Python/FastAPI backend handling thousands of concurrent requests with non-blocking I/O, parallel execution, background jobs, and sub-10ms cached response times.
- Built a yield aggregation service across 55+ DeFi protocols, standardizing reward schedules, lending rates, liquidity incentives, and pool data into consistent APY, borrow-rate, and TVL metrics.
- Designed 8 specialized TTL/LRU caches, reducing codebase complexity by 37.5% and lock contention by 40%; built fault-tolerant batch pipelines processing 2,000+ pools and yield sources per run.
- Engineered async RPC orchestration for hundreds of EVM calls per cycle using batching, retries, timeout isolation, and rate-limit-aware scheduling; designed low-latency data paths with DynamoDB, Timestream, async writes, and cache-first reads.
- Deployed and operated workloads on AWS Lambda and EC2 with horizontal scaling, event-driven jobs, zero-downtime releases, structured logs, dashboards, alerts, health checks, and p50–p99 latency monitoring.

Software Development Engineer Intern, Olive Technologies

Nov 2023 – Jan 2024

- Built Node.js inventory REST APIs and request validation, plus a React Native OCR capture flow that converted scanned bills into structured JSON for backend ingestion.

PUBLICATION

IEEE CICT 2025: Co-authored “A Physics-Informed Computer Vision Framework for Firefighting Assistance” [\[IEEE Xplore\]](#)

PROJECTS

VibeDeploy | Self-Hosted Deployment & AI Security Platform

[Live Demo](#) · [Code](#)

Python, FastAPI, PostgreSQL, asyncpg, Docker, AWS EC2, Caddy, MCP

- Built and deployed a self-hosted deployment control plane running 6 services on a 2-vCPU, 1.9GB EC2 instance, with HMAC-verified GitHub webhooks, PostgreSQL SKIP LOCKED job queues, renewable leases, fencing tokens, and dynamic Caddy routing for 2 live applications.
- Built health-gated blue-green deployments with automated rollback and graceful draining, achieving 0 failed requests across ~300 requests at 10 RPS; added a 10-second reconciliation loop that recovers from crashes at 5 deployment stages, adopts orphaned containers, and prevents stale workers from publishing routes.
- Built a schema-driven PostgreSQL security engine using 5 RLS policy templates, running 29 verification check types across 4 personas and 6 attack scenarios per table before deployment; validated fail-closed behavior with 19 planted-vulnerability tests covering tenant isolation and privilege escalation.
- Built an MCP-based agent gateway with 6 permission-scoped tools, user-inherited database access, risk-based approvals, diff-hash validation, and tamper-evident audit logs; enforced at-most-once execution through a 13-state workflow, backed by 119 gateway tests and 660 passing tests overall.

Distributed SQL Query Engine over Object Storage

[Live Demo](#) · [Code](#)

Python, PyArrow, Polars, SQLGlot, Django, Celery, Redis, PostgreSQL, Docker

- Built a SQL query engine executing analytical workloads directly against partitioned Parquet files in S3/MinIO without data ingestion, implementing AST parsing, logical/physical query planning, a 3-rule optimizer, process-based parallel scans, and partial aggregation with coordinator-side merging.
- Cut storage I/O by up to 430× (99.8%) through partition pruning, predicate pushdown, row-group statistics, and column projection; skipped 119/120 row groups, scanning 14KB of 5.9MB on a 300K-row dataset, and separately reduced COUNT(*) I/O 19× (1.59MB to 85KB).
- Processed 300K rows in 68.5ms (~4.4M rows/sec) in a single-node aggregation benchmark; built adaptive multiprocessing with row-group-level tasks, worker-local aggregation, and parallel result merging, preserving SQL NULL semantics and identical results across 1 and 4 workers.
- Built an asynchronous query platform using Django REST Framework, Celery, Redis metadata caching, and PostgreSQL, with a 9-page observability dashboard exposing execution plans, scanned bytes, pruning metrics, and query history; validated correctness against DuckDB across ~165 test cases.