NeuroVisor Architecture Document

Version: Current State **Date:** February 2026 **Author:** Generated from codebase analysis

1. Executive Summary

NeuroVisor is a Firecracker-based microVM orchestrator with integrated LLM agent capabilities. It provides sandboxed code execution environments for AI-driven tasks, combining:

- **Firecracker microVMs** for lightweight, secure isolation
- **VM Pool** for instant VM availability with pre-warming
- **gRPC over vsock** for efficient host-guest communication
- **Ollama integration** for LLM inference with native tool calling
- **Agent loop** for autonomous code execution tasks

2. High-Level Architecture

```
┌──────────────────────────────────────────────────────────────────────────────┐
│                              NEUROVISOR HOST                                 │
├──────────────────────────────────────────────────────────────────────────────┤
│                                                                              │
│  ┌──────────────┐     ┌──────────────┐     ┌──────────────────────────────┐  │
│  │   CLI/Main   │────►│ Agent Mode   │────►│     Ollama LLM (/api/chat)   │  │
│  │              │     │ Controller   │◄────│     with Tool Calling        │  │
│  └──────────────┘     └──────────────┘     └──────────────────────────────┘  │
│         │                    │                                               │
│         │                    │ execute_code()                                │
│         ▼                    ▼                                               │
│  ┌─────────────────────────────────────────────────────────────────────────┐ │
│  │                           VM POOL                                       │ │
│  │  ┌─────────┐  ┌─────────┐  ┌─────────┐       ┌─────────────────────┐    │ │
│  │  │ VM      │  │ VM      │  │ VM      │ ◄───► │ Pool Replenisher    │    │ │
│  │  │ (warm)  │  │ (warm)  │  │ (warm)  │       │ (background task)   │    │ │
│  │  └────┬────┘  └─────────┘  └─────────┘       └─────────────────────┘    │ │
│  │       │acquire()                                                        │ │
│  │       ▼                                                                 │ │
│  │  ┌──────────────────────────────────────────┐                           │ │
│  │  │ Firecracker Process (active VM)          │                           │ │
│  │  │   - KVM-based microVM                    │                           │ │
│  │  │   - vsock device                         │                           │ │
│  │  │   - Alpine Linux rootfs                  │                           │ │
│  │  └────────────────────┬─────────────────────┘                           │ │
│  └───────────────────────┼─────────────────────────────────────────────────┘ │
│                          │                                                   │
│  ┌───────────────────────┼─────────────────────────────────────────────────┐ │
│  │ ExecutionClient       │ vsock handshake (CONNECT/OK protocol)           │ │
│  │ (host-side gRPC)      │                                                 │ │
│  └───────────────────────┼─────────────────────────────────────────────────┘ │
│                          │                                                   │
│ ═════════════════════════╪═══════════════════════════════════════════════════ │
│                          │ vsock (AF_VSOCK, CID-based)                       │
│ ═════════════════════════╪═══════════════════════════════════════════════════ │
│                          │                                                   │
├──────────────────────────┼───────────────────────────────────────────────────┤
│                          ▼                   FIRECRACKER GUEST VM            │
│  ┌─────────────────────────────────────────────────────────────────────────┐ │
│  │                      Guest Agent (guest_agent binary)                   │ │
│  │  ┌────────────────────────────────────────────────────────────────────┐ │ │
│  │  │ ExecutionServer (gRPC over vsock port 6000)                        │ │ │
│  │  │   - Raw libc vsock (musl compatible)                               │ │ │
│  │  │   - Execute() / ExecuteStream() RPCs                               │ │ │
│  │  └────────────────────────────────────────────────────────────────────┘ │ │
│  │                                  │                                      │ │
│  │                                  ▼                                      │ │
│  │  ┌────────────────────────────────────────────────────────────────────┐ │ │
│  │  │ Code Execution                                                     │ │ │
│  │  │   - /bin/sh (bash/shell)                                           │ │ │
│  │  │   - python3 (if installed)                                         │ │ │
│  │  │   - node (if installed)                                            │  ││
│  │  │   - Timeout enforcement                                            │  ││
│  │  │   - stdout/stderr capture                                          │  ││
│  │  └────────────────────────────────────────────────────────────────────┘  ││
│  └──────────────────────────────────────────────────────────────────────────┘│
│                                                                              │
└─────────────────────────────────────────────────────────────────────────────┘
```

3. Module Breakdown

3.1 Core Modules

Module	Path	Description
`vm`	`src/vm/`	Firecracker VM lifecycle management
`agent`	`src/agent/`	LLM-driven code execution loop
`grpc`	`src/grpc/`	gRPC server/client for host-guest communication
`ollama`	`src/ollama/`	Ollama LLM client with tool calling
`security`	`src/security/`	Seccomp filters, capabilities, rate limiting
`cgroups`	`src/cgroups/`	Resource isolation (CPU/memory limits)
`metrics`	`src/metrics/`	Prometheus metrics for observability

3.2 Guest Components

Component	Path	Description
`guest_agent`	`guest/agent/main.rs`	In-VM execution server

4. VM Management Subsystem

4.1 VMManager (`src/vm/manager.rs`)

Handles VM creation and destruction:

- Creates Firecracker processes with API sockets

- Configures boot source (kernel, rootfs)
- Sets up vsock devices for host-guest communication
- Assigns unique CIDs (Context IDs) per VM
- Supports snapshot-based boot for faster startup

Configuration:

```
VMManagerConfig {
    kernel_path: "./vmlinuz",
    rootfs_path: "./rootfs.ext4",
    snapshot_path: Option<"./snapshot_file">,
    mem_path: Option<"./mem_file">,
    resource_limits: ResourceLimits,
    vsock_port: 6000,
}
```

4.2 VMPool (`src/vm/pool.rs`)

Thread-safe pool of pre-warmed VMs:

```
┌─────────────────────────────────────────────────────────────────┐
│  VM Pool                                                        │
│                                                                 │
│  ┌───────────────┐    ┌───────────────┐    ┌───────────────┐  │
│  │   VM Ready    │    │   VM Ready    │    │   VM Ready    │  │
│  │   (warm)      │    │   (warm)      │    │   (warm)      │  │
│  └───────────────┘    └───────────────┘    └───────────────┘  │
│          │                                                      │
│          ▼                                                      │
│    acquire() ──► VM assigned to request                        │
│          │                                                      │
│          ▼                                                      │
│    release() ──► VM destroyed, replenish triggered             │
└─────────────────────────────────────────────────────────────────┘
```

Key Features: - **Pre-warming:** Creates VMs at startup for instant availability - **Acquire/Release:** Thread-safe VM checkout/return - **Background Replenisher:** Maintains target pool size automatically - **Destroy-on-release:** VMs are destroyed after each use for isolation - **Configurable limits:** `warm_size` (default 3), `max_size` (default 10)

4.3 VMHandle (`src/vm/handle.rs`)

Represents a running VM with: - Unique `vm_id` (UUID v7) - Firecracker child process - API socket path - Vsock socket path - Context ID (CID) - Status tracking (Warm/Active)

---

5. Agent Loop Subsystem

5.1 AgentController (`src/agent/controller.rs`)

Orchestrates LLM-driven code execution:

```
User Task
    │
    ▼
┌──────────────────────────────────────┐
│         AgentController               │
│  ┌──────────────────────────────┐    │
│  │ 1. Send task to Ollama       │     │
│  │    (with execute_code tool)  │    │
│  └──────────────────────────────┘    │
│               │                       │
│               ▼                       │
│  ┌──────────────────────────────┐    │
│  │ 2. Parse tool calls          │    │
│  │    (native or text-based)    │    │
│  └──────────────────────────────┘    │
│               │                       │
│               ▼                       │
│  ┌──────────────────────────────┐    │
│  │ 3. Execute in sandboxed VM   │    │
│  │    - Acquire VM from pool    │    │
│  │    - Connect via vsock       │    │
│  │    - Run code                │    │
│  │    - Capture output          │    │
│  │    - Release VM              │    │
│  └──────────────────────────────┘    │
│               │                       │
│               ▼                       │
│  ┌──────────────────────────────┐    │
│  │ 4. Feed result back to LLM   │    │
│  │    Loop until complete       │    │
│  └──────────────────────────────┘    │
└──────────────────────────────────────┘
```

Configuration:

```
AgentConfig {
    model: "qwen3",           // LLM model
    max_iterations: 10,       // Max LLM calls
    execution_timeout_secs: 30,
    vsock_port: 6000,
    connection_retries: 10,
    connection_retry_delay_ms: 500,
}
```

5.2 Tool Definition

The agent provides one tool to the LLM:

```
{
  "type": "function",
  "function": {
    "name": "execute_code",
    "description": "Execute code in a sandboxed environment",
    "parameters": {
      "type": "object",
      "properties": {
        "language": {
          "type": "string",
          "enum": ["python", "bash", "javascript"]
        },
        "code": {
          "type": "string",
          "description": "The code to execute"
        }
      },
      "required": ["language", "code"]
    }
  }
}
```

---

6. Communication Layer

6.1 Vsock Protocol

Firecracker uses vsock for efficient VM communication:

```
Host                                  Guest
  │                                     │
  │ 1. Connect to UDS                   │
  │    (./neurovisor-{id}.vsock)        │
  │────────────────────────────────────►│
  │                                     │
  │ 2. Send "CONNECT 6000\n"            │
  │────────────────────────────────────►│
  │                                     │
  │ 3. Receive "OK {port}\n"            │
  │◄────────────────────────────────────│
  │                                     │
  │ 4. gRPC messages                    │
  │◄───────────────────────────────────►│
  │                                     │
```

6.2 ExecutionClient (`src/grpc/execution.rs`)

Host-side gRPC client: - Performs vsock handshake protocol - Creates Tonic gRPC channel over vsock - Provides `execute()` and `execute_with_env()` methods - Retry logic for guest readiness

6.3 ExecutionServer (Guest)

In-VM gRPC server (`guest/agent/main.rs`): - Raw libc vsock listener (musl compatible) - Supports Python, Bash/Shell, JavaScript - Timeout enforcement - Streaming output support

Protobuf Definition:

```
service ExecutionService {
    rpc Execute(ExecuteRequest) returns (ExecuteResponse);
    rpc ExecuteStream(ExecuteRequest) returns (stream ExecuteChunk);
}

message ExecuteRequest {
    string language = 1;
    string code = 2;
    uint32 timeout_secs = 3;
    map<string, string> env = 4;
}

message ExecuteResponse {
    string stdout = 1;
    string stderr = 2;
    int32 exit_code = 3;
    double duration_ms = 4;
    bool timed_out = 5;
}
```

---

7. Ollama Integration

7.1 ChatClient (`src/ollama/tool_use.rs`)

Communicates with Ollama's `/api/chat` endpoint:

- Native tool calling support (model-dependent)
- Fallback text-based tool call parsing
- Temperature=0 for deterministic behavior
- Conversation history management

System Prompt:

```
You are an AI assistant with the ability to execute code in a
sandboxed environment. When you need to run code to accomplish
a task, use the execute_code tool. Always prefer using bash/shell
for simple file operations and system commands.
```

7.2 Tool Call Parsing

Two parsing modes: 1. **Native:** Model returns `tool_calls` array (qwen3, llama3.2)
2. **Text-based:** Extract JSON tool calls from response text

8. Security Architecture

8.1 Security Layers

```
┌─────────────────────────────────────────────────────────────────┐
│  Layer 1: CAPABILITIES                                          │
│  Drop dangerous root powers BEFORE starting Firecracker         │
│                                                                 │
│  ┌───────────────────────────────────────────────────────────┐  │
│  │ DROP: CAP_SYS_ADMIN, CAP_PTRACE, CAP_NET_RAW, etc.        │  │
│  │ KEEP: CAP_DAC_OVERRIDE (for /dev/kvm access)              │  │
│  └───────────────────────────────────────────────────────────┘  │
└─────────────────────────────────────────────────────────────────┘
                              │
                              ▼
┌─────────────────────────────────────────────────────────────────┐
│  Layer 2: SECCOMP BPF                                           │
│  Block dangerous syscalls at kernel level                       │
│                                                                 │
│  Process ──syscall──► Filter ──allowed?──► Kernel               │
│                          │                                      │
│                          └─blocked─► SIGKILL (process dies)     │
└─────────────────────────────────────────────────────────────────┘
                              │
                              ▼
┌─────────────────────────────────────────────────────────────────┐
│  Layer 3: FIRECRACKER ISOLATION                                 │
│  - KVM hardware virtualization                                  │
│  - Minimal device model                                         │
│  - No network by default                                        │
│  - Read-only kernel, ephemeral rootfs                           │
└─────────────────────────────────────────────────────────────────┘
                              │
                              ▼
┌─────────────────────────────────────────────────────────────────┐
│  Layer 4: CGROUPS v2                                            │
│  Resource limits (CPU, memory) enforced by kernel               │
└─────────────────────────────────────────────────────────────────┘
```

8.2 Rate Limiting (`src/security/rate_limit.rs`)

Token bucket rate limiter for API requests: - Configurable capacity and refill rate - Per-request token consumption

9. Observability

9.1 Prometheus Metrics (`src/metrics/mod.rs`)

Inference Metrics: - `neurovisor_requests_total{model}` - Total requests - `neurovisor_tokens_generated_total{model}` - Tokens generated - `neurovisor_inference_duration_seconds` - Inference latency - `neurovisor_errors_total{error_type}` - Error counts

VM Pool Metrics: - `neurovisor_pool_warm_vms` - Pre-warmed VMs - `neurovisor_pool_active_vms` - Active VMs - `neurovisor_vm_acquire_seconds` - VM acquisition time - `neurovisor_vm_boot_seconds` - VM boot time

Agent Metrics: - `neurovisor_agent_tasks_total{status}` - Agent tasks - `neurovisor_agent_iterations` - Iterations per task - `neurovisor_code_execution_seconds` - Code execution time - `neurovisor_code_executions_total{language,status}` - Executions

Endpoint: `http://0.0.0.0:9090/metrics`

10. File Structure

```
neurovisor/
├── src/
│   ├── main.rs              # Daemon entry point
│   ├── lib.rs               # Library crate root
│   ├── agent/
│   │   ├── mod.rs           # Agent module
│   │   └── controller.rs    # Agent loop implementation
│   ├── vm/
│   │   ├── mod.rs           # VM module
│   │   ├── manager.rs       # VMManager
│   │   ├── pool.rs          # VMPool
│   │   ├── handle.rs        # VMHandle
│   │   ├── firecracker.rs   # Firecracker API client
│   │   ├── lifecycle.rs     # VM spawn/wait helpers
│   │   └── config.rs        # VM configuration types
│   ├── grpc/
│   │   ├── mod.rs           # gRPC module
│   │   ├── execution.rs     # ExecutionClient (host-side)
│   │   ├── gateway.rs       # GatewayServer (multi-VM)
│   │   └── server.rs        # InferenceServer
│   ├── ollama/
│   │   ├── mod.rs           # Ollama module
│   │   ├── client.rs        # OllamaClient (generate)
│   │   └── tool_use.rs      # ChatClient (chat + tools)
│   ├── security/
│   │   ├── mod.rs           # Security module
│   │   ├── seccomp.rs       # Seccomp BPF filters
│   │   ├── capabilities.rs  # Linux capabilities
│   │   └── rate_limit.rs    # Token bucket rate limiter
│   ├── cgroups/
│   │   ├── mod.rs           # Cgroups module
│   │   └── manager.rs       # CgroupManager
│   └── metrics/
│       └── mod.rs           # Prometheus metrics
├── guest/
│   └── agent/
│       └── main.rs          # Guest execution server
├── proto/
│   ├── execution.proto      # Execution service proto
│   └── inference.proto      # Inference service proto
├── vmlinuz                  # Linux kernel
├── rootfs.ext4              # Alpine Linux rootfs
└── firecracker              # Firecracker binary (symlink)
```

11. Configuration

11.1 Command Line Arguments

```
# Default daemon mode
sudo ./neurovisor

# Custom pool size
sudo ./neurovisor --warm 5 --max 20

# Use snapshots for faster boot
sudo ./neurovisor --snapshot

# Agent mode (single task)
sudo ./neurovisor --agent "Write a script that prints hello world"
```

11.2 Constants

Constant	Value	Description
`KERNEL_PATH`	`./vmlinuz`	Linux kernel path
`ROOTFS_PATH`	`./rootfs.ext4`	Root filesystem path
`METRICS_PORT`	`9090`	Prometheus metrics port
`GATEWAY_PORT`	`50051`	gRPC gateway port
`VSOCK_PORT`	`6000`	Guest execution service port
`DEFAULT_WARM_SIZE`	`3`	Default pre-warmed VMs
`DEFAULT_MAX_SIZE`	`10`	Maximum VMs

12. Data Flow Examples

12.1 Agent Task Execution

```
1. User runs: ./neurovisor --agent "List files in /tmp"

2. Main → AgentController.run("List files in /tmp")

3. AgentController → Ollama /api/chat
   - System prompt + user task + tools

4. Ollama returns tool_call:
   {
     "name": "execute_code",
     "arguments": {"language": "bash", "code": "ls -la /tmp"}
   }

5. AgentController.execute_code():
   a. pool.acquire() → VMHandle
   b. ExecutionClient.connect(vsock_path)
   c. Vsock handshake (CONNECT/OK)
   d. gRPC Execute(language="bash", code="ls -la /tmp")
   e. Guest runs /bin/sh -c "ls -la /tmp"
   f. Returns stdout/stderr/exit_code
   g. pool.release(vm) → VM destroyed

6. AgentController → Ollama /api/chat
   - Add tool result to conversation

7. Ollama returns final response (no tool calls)

8. AgentController returns AgentResult
```

12.2 VM Lifecycle

```
1. VMPool.initialize()
   └── Creates 3 warm VMs

2. VMPool.acquire()
   └── Returns pre-warmed VM, marks as active

3. [Code execution happens]

4. VMPool.release(vm)
   ├── Destroys VM (kill Firecracker, cleanup)
   └── Triggers replenisher

5. Background replenisher
   └── Creates new warm VM to maintain pool size
```

---

13. Dependencies

13.1 Rust Crates

Crate	Purpose
`tokio`	Async runtime
`tonic`	gRPC framework
`prost`	Protobuf codegen
`hyper`	HTTP server
`reqwest`	HTTP client (Ollama)
`serde_json`	JSON serialization
`prometheus`	Metrics
`uuid`	UUID generation
`lazy_static`	Static initialization

13.2 External Services

Service	Purpose	Default URL
Ollama	LLM inference	`http://localhost:11434`
Firecracker	MicroVM hypervisor	`./firecracker` binary

---

14. Future Considerations

1. **Networking:** Add VM network support for internet access
2. **Persistence:** Support persistent VM state across requests
3. **Multi-tenant:** Add user/namespace isolation
4. **Streaming:** Full streaming support for long-running tasks
5. **Languages:** Add more language runtimes to guest rootfs
6. **GPU:** GPU passthrough for ML workloads

Document generated from NeuroVisor codebase analysis - February 2026