Deployment Guide: Neural Forest (NF)

Target Team: Global Scaling Efficiency (Meta Compute Initiative)

Objective: Prove the reduction of Kubernetes memory overprovisioning from 16.4 GiB (monolith) to 2.7 GiB (actual need) using dynamic resource pruning and "Inference-First" routing.

1. Architectural Strategy: The "Micro-Ensemble" Pod

The Neural Forest replaces a single massive Deployment with a modular ensemble of specialized **Neural Trees**.

- **Monolithic Baseline:** A single container loading a 70B parameter model requires 16.4 GiB+ just for weights and KV cache.
- **NF Configuration:** The system uses a **Conductor** pod to orchestrate thousands of specialized, shallow sub-networks (MLPs, CNNs, RNNs) that are loaded into memory only when needed.

2. FastAPI Conductor Integration

The **Conductor** acts as the meta-cognitive orchestrator, routing incoming traffic to relevant "trees" based on the query's semantic requirements.

Conceptual Python Snippet (main.py):

Python

```python
from fastapi import FastAPI, Request
from nf_orchestrator import NeuralForestConductor

app = FastAPI()
# Conductor loads metadata for 1000+ trees, but not the full weights
conductor = NeuralForestConductor(strategy="inference-first")

@app.post("/v1/predict")
async def predict(request: Request):
```

```python
    data = await request.json()

    # Dynamic Routing: The Conductor identifies which 5% of trees are relevant
    active_trees = conductor.route(data)

    # Predictive Averaging: Aggregating results from specialized atoms
    result = await conductor.aggregate(active_trees, data)

    # Audit Shield: Log energy intensity and memory delta per request
    conductor.log_audit_metrics(request_id=data.get("id"))

    return result
```

- **Energy Intensity Logs:** Each request generates a verifiable log of energy consumed per inference, satisfying "Green AI" compliance.

3. Kubernetes YAML: Resource Pruning & OOMKilled Prevention

To achieve the 2.7 GiB footprint, the YAML configuration utilizes **horizontal pod autoscaling (HPA)** on specialized "micro-pods" rather than the monolith.

Sample Specialized Tree Deployment (nf-tree-spatial.yaml):

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nf-tree-spatial-cnn
  labels:
    tier: inference-atom
spec:
  replicas: 3
  template:
    spec:
      containers:
      - name: neural-tree
        image: meta-registry/nf-core:latest
        resources:
          # This is where we eliminate the 4x memory padding
          requests:
            memory: "1Gi"
```

```yaml
        cpu: "500m"
      limits:
        memory: "2.7Gi" # Actual peak need, no 6.1x waste
    env:
    - name: MODALITY
      value: "spatial_vision"
    - name: NF_AUDIT_LOGGING
      value: "enabled"
```

- **OOMKilled Prevention:** By utilizing "forced specialization," individual pods remain within deterministic memory boundaries, eliminating the need for standard 16.4 GiB "safety buffers".

4. Verification: Proving the 2.7 GiB Reduction

To validate the efficiency for the **Meta Compute** review board, use the following metrics dashboard:

- **Memory Wall Delta:** Compare container_memory_usage_bytes of the monolithic Llama pod (16.4 GiB) vs. the aggregate of the active NF-Tree pods (avg. 2.7 GiB).
- **Dynamic Routing Efficiency:** Track the percentage of "idle" silicon. In the NF paradigm, specific micro-cores are only powered when their assigned Neural Tree is activated by the Conductor.

- **Audit Trail:** Use the **Enterprise Audit Shield** to provide a per-request breakdown of feature importance and energy cost.

5. Silicon Roadmap Alignment

Transition these software-defined pods into hardware-accelerated **NF-Core** instances. The NF-Core will shift weights from off-chip HBM to distributed **SRAM/eDRAM**, further reducing the memory-trip energy cost that current H200/B200 clusters face.