Contents

Cato Shadow Self GPU Infrastructure

This document describes the GPU infrastructure requirements for running the Shadow Self component of Cato's consciousness verification system.

Overview

The Shadow Self is a local neural network that provides mechanistic verification of introspective claims. In production, this uses **Llama-3-8B** running on GPU infrastructure. Currently, Cato simulates this via LLM API calls, but true local inference provides:

- **Lower latency** (no network round-trip)
- **Activation access** (can probe hidden states)
- **Structural correspondence** (verify patterns exist in model)
- **Privacy** (no data leaves infrastructure)

Architecture Options

Option 1: AWS SageMaker (Recommended)

Best for: Production deployments with variable load

```
Cato Lambda        SageMaker Endpoint
(Node.js)              (Llama-3-8B)
                  g5.xlarge
```

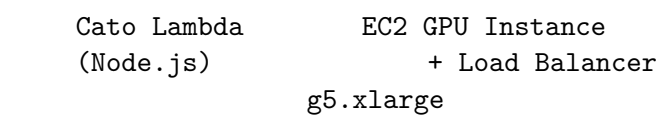
Infrastructure (CDK):

```
// packages/infrastructure/lib/stacks/cato-gpu-stack.ts

import * as sagemaker from 'aws-cdk-lib/aws-sagemaker';

const shadowSelfEndpoint = new sagemaker.CfnEndpoint(this, 'ShadowSelfEndpoint', {
  endpointName: 'cato-shadow-self',
  endpointConfigName: shadowSelfConfig.attrEndpointConfigName,
});

const shadowSelfConfig = new sagemaker.CfnEndpointConfig(this, 'ShadowSelfConfig', {
  endpointConfigName: 'cato-shadow-self-config',
  productionVariants: [{
    variantName: 'AllTraffic',
    modelName: shadowSelfModel.attrModelName,
    instanceType: 'ml.g5.xlarge',  // 24GB VRAM
    initialInstanceCount: 1,
  }],
});
```

Costs: | Instance | GPU | VRAM | Cost/hr | Cost/mo (24/7) | |———-|—|—-|——|———-| | g5.xlarge | A10G | 24GB | $1.006 | ~$724 | | g5.2xlarge | A10G | 24GB | $1.212 | ~$873 | | g5.4xlarge | A10G | 24GB | $1.624 | ~$1,169 |

Option 2: EC2 with Auto-Scaling

Best for: Cost optimization with predictable load



With Spot Instances (up to 90% savings):

```
const spotFleet = new ec2.CfnSpotFleet(this, 'ShadowSelfSpotFleet', {
  spotFleetRequestConfigData: {
    iamFleetRole: spotFleetRole.roleArn,
    targetCapacity: 1,
    launchSpecifications: [{
      instanceType: 'g5.xlarge',
      spotPrice: '0.50', // Max bid
      imageId: deepLearningAmi.imageId,
    }],
  },
});
```

Option 3: AWS Inferentia (Most Cost-Effective)

Best for: High-throughput, cost-sensitive deployments

```
Cato Lambda        inf2.xlarge
(Node.js)              (Neuron-compiled)
```

Costs: ~$0.76/hr (~$547/mo) - but requires model compilation

Model Requirements

Llama-3-8B Specifications

Requirement	Value
Model Size	~16GB (FP16) / ~8GB (INT8)
Min VRAM	16GB
Recommended VRAM	24GB
Inference Latency	50-200ms
Context Length	8,192 tokens

Probing Classifier Requirements

The Shadow Self uses **probing classifiers** trained on model activations:

- **Layer to probe**: Usually layers 16-24 (mid-to-late)
- **Activation dimensions**: 4096 (Llama-3-8B hidden size)
- **Classifier**: Linear probe or small MLP
- **Training data**: 100-1000 labeled examples per claim type

Implementation Guide

Step 1: Deploy Model to SageMaker

```
# Create model artifact
cd /path/to/llama-3-8b
tar -czvf model.tar.gz --exclude='*.bin' .

# Upload to S3
aws s3 cp model.tar.gz s3://radiant-models/shadow-self/model.tar.gz
```

Step 2: Create SageMaker Model

```
const shadowSelfModel = new sagemaker.CfnModel(this, 'ShadowSelfModel', {
  modelName: 'cato-shadow-self-llama3',
  executionRoleArn: sagemakerRole.roleArn,
  primaryContainer: {
    image: '763104351884.dkr.ecr.us-east-1.amazonaws.com/huggingface-pytorch-tgi-inference:2.1
    modelDataUrl: 's3://radiant-models/shadow-self/model.tar.gz',
```

```
    environment: {
      'HF_MODEL_ID': 'meta-llama/Meta-Llama-3-8B',
      'SM_NUM_GPUS': '1',
      'MAX_INPUT_LENGTH': '4096',
      'MAX_TOTAL_TOKENS': '8192',
    },
  },
});
```

Step 3: Update Shadow Self Service

```
// shadow-self.service.ts

private async invokeLocalModel(context: string): Promise<{
  activations: number[];
  response: string;
}> {
  const sagemakerRuntime = new SageMakerRuntimeClient({});

  const response = await sagemakerRuntime.send(new InvokeEndpointCommand({
    EndpointName: 'cato-shadow-self',
    ContentType: 'application/json',
    Body: JSON.stringify({
      inputs: context,
      parameters: {
        max_new_tokens: 256,
        return_hidden_states: true,  // Get activations
        hidden_states_layer: 20,     // Layer to probe
      },
    }),
  }));

  const result = JSON.parse(new TextDecoder().decode(response.Body));

  return {
    activations: result.hidden_states,
    response: result.generated_text,
  };
}
```

Step 4: Train Probing Classifiers

```
# probing/train_probe.py

import torch
import torch.nn as nn
from sklearn.model_selection import train_test_split
```

```python
class LinearProbe(nn.Module):
    def __init__(self, input_dim=4096, num_classes=5):
        super().__init__()
        self.classifier = nn.Linear(input_dim, num_classes)

    def forward(self, x):
        return self.classifier(x)

def train_probe(activations, labels, claim_type):
    """Train a probe for a specific claim type."""
    X_train, X_test, y_train, y_test = train_test_split(
        activations, labels, test_size=0.2
    )

    probe = LinearProbe(num_classes=len(set(labels)))
    optimizer = torch.optim.Adam(probe.parameters(), lr=1e-3)
    criterion = nn.CrossEntropyLoss()

    for epoch in range(100):
        optimizer.zero_grad()
        outputs = probe(X_train)
        loss = criterion(outputs, y_train)
        loss.backward()
        optimizer.step()

    # Evaluate
    with torch.no_grad():
        test_outputs = probe(X_test)
        accuracy = (test_outputs.argmax(1) == y_test).float().mean()

    return probe, accuracy.item()
```

Environment Variables

```bash
# Required for GPU inference
SHADOW_SELF_ENDPOINT=cato-shadow-self
SHADOW_SELF_REGION=us-east-1
SHADOW_SELF_USE_GPU=true

# Optional: Local inference (for development)
SHADOW_SELF_LOCAL_MODEL_PATH=/models/llama-3-8b
SHADOW_SELF_DEVICE=cuda:0
```

Licensing Notes

Important: Llama-3 requires acceptance of Meta's license agreement: - Visit: https://llama.meta.com/llama-downloads/ - Accept license for commercial use - Download from HuggingFace: `meta-llama/Meta-Llama-3-8B`

The license allows commercial use but requires: 1. Attribution to Meta 2. Compliance with

acceptable use policy 3. Monthly active users < 700M (otherwise contact Meta)

Monitoring

CloudWatch Metrics

```
// Monitor GPU utilization
new cloudwatch.Alarm(this, 'GPUUtilizationAlarm', {
  metric: shadowSelfEndpoint.metricGPUUtilization(),
  threshold: 90,
  evaluationPeriods: 3,
  alarmDescription: 'Shadow Self GPU utilization > 90%',
});

// Monitor inference latency
new cloudwatch.Alarm(this, 'InferenceLatencyAlarm', {
  metric: shadowSelfEndpoint.metricModelLatency(),
  threshold: 500, // 500ms
  evaluationPeriods: 5,
  alarmDescription: 'Shadow Self inference latency > 500ms',
});
```

Cost Optimization

1. **Auto-scaling**: Scale to 0 during low-usage periods
2. **Spot Instances**: Use for non-critical probing
3. **Inferentia**: Compile model for AWS Neuron (~40% cheaper)
4. **Quantization**: Use INT8 to reduce VRAM (slight accuracy trade-off)
5. **Caching**: Cache probing results for repeated contexts

Fallback Behavior

When GPU infrastructure is unavailable, Cato falls back to:

1. **LLM API Simulation**: Uses Claude/GPT to simulate Shadow Self responses
2. **Pattern Matching**: Uses regex/embedding similarity instead of probing
3. **Degraded Mode**: Skips Shadow Self phase, relies on other 3 phases

This ensures Cato remains functional even without dedicated GPU resources.