Deployment Guide: Seamless-M4T API for Amharic Speech-to-Text

This guide explains how to deploy the Seamless-M4T API for high-performance, low-latency Amharic speech-to-text translation. The API is designed to handle a high number of concurrent requests efficiently.

Hardware Recommendations

For optimal performance processing Amharic speech-to-text translations with Seamless-M4T-v2-Large, I recommend the following hardware specifications:

Optimal Configuration

- **GPU**: NVIDIA A10G or higher (24GB VRAM)
- **CPU**: 16+ cores (32+ vCPUs)
- **RAM**: 64GB+
- **Storage**: 200GB+ SSD
- **Network**: High bandwidth connection (1Gbps+)

Minimum Viable Configuration

- **GPU**: NVIDIA T4 (16GB VRAM)
- **CPU**: 8+ cores (16+ vCPUs)
- **RAM**: 32GB
- **Storage**: 100GB SSD

Scaling Considerations

- Each GPU can handle approximately 10-15 concurrent requests
- For higher throughput, deploy multiple instances across multiple GPUs
- Vertical scaling (bigger GPU) helps with larger audio files
- Horizontal scaling (more GPUs) helps with more concurrent users

Deployment Options

Option 1: Cloud Providers (Recommended)

Google Cloud Platform (GCP)

- **Instance type**: A2 standard (with NVIDIA A100) or N1 with T4
- **Deployment**: Google Kubernetes Engine (GKE) or Compute Engine
- **Benefits**: Excellent GPU options, autoscaling, global reach

Amazon Web Services (AWS)

- **Instance type**: g4dn.2xlarge (T4 GPU) or g5.xlarge (A10G GPU)
- **Deployment**: ECS/EKS or EC2 with Docker
- **Benefits**: Wide geographic distribution, integration with other AWS services

Azure

- **Instance type**: NC-series (V4) with NVIDIA T4 or A10
- **Deployment**: AKS or Azure Container Instances
- **Benefits**: Good enterprise integration, solid performance

Option 2: Self-Hosted

For organizations with existing infrastructure or specific regulatory requirements:

- Enterprise-grade NVIDIA GPUs (RTX A6000, A100, etc.)
- Docker and Docker Compose for deployment
- Optional: Kubernetes for orchestration

Deployment Steps

1. Prepare the Deployment Environment

Clone the repository and prepare the deployment files:

```bash
# Clone repository (if using Git)
git clone https://github.com/your-org/seamless-m4t-api
cd seamless-m4t-api

# Or create directories and files manually
mkdir -p seamless-m4t-api/{nginx,scripts}
cd seamless-m4t-api
# Copy all the provided files to their respective locations
```

2. Configure the Application

Review and adjust configuration in `docker-compose.yml`:

```yaml
environment:
  - PORT=8000
  - WORKERS=4  # Adjust based on CPU cores
  - BATCH_SIZE=8  # Adjust based on GPU memory
  - MAX_AUDIO_LENGTH=300  # Maximum audio length in seconds
```

3. Building and Deploying

Using Docker Compose (Simpler)

```bash
# Build the Docker image
docker-compose build

# Start the services
docker-compose up -d

# Check logs
docker-compose logs -f
```

Using Kubernetes (For Production)

```bash
# Apply Kubernetes manifests
kubectl apply -f k8s/namespace.yaml
kubectl apply -f k8s/deployment.yaml
kubectl apply -f k8s/service.yaml
kubectl apply -f k8s/ingress.yaml

# Check deployment status
kubectl get pods -n seamless-m4t
```

4. Verify Deployment

Check that the API is running correctly:

```bash
# Test the health endpoint
curl http://your-server-ip/health

# Test transcription with a sample file
curl -X POST http://your-server-ip/transcribe \
  -F "file=@sample.wav" \
  -F "target_language=amh"
```

Performance Tuning

GPU Optimization

1. **Enable TensorRT acceleration**:

```python
# In app.py, modify model loading
if DEVICE == "cuda":
    model = model.half()  # Use FP16
    # Optionally add TensorRT optimization
```

2. **Adjust batch size** based on your GPU memory:
   - Larger batch sizes improve throughput but require more memory
   - Test with different values (4, 8, 16) to find the optimal setting

Server Tuning

1. **Worker configuration** in Gunicorn:

```bash
# In Dockerfile, modify the ENTRYPOINT
ENTRYPOINT ["gunicorn", "app:app", "--bind", "0.0.0.0:8000", "--workers", "4", "--w
```

   - Set workers to (2 × CPU cores) + 1 for CPU-bound tasks
   - For GPU workloads, adjust based on GPU memory

2. **NGINX configuration**:
   - Review and adjust buffer sizes and timeouts in `nginx/nginx.conf`
   - Enable compression for text responses
   - Configure proper caching headers

Scaling Strategies

Vertical Scaling

- Upgrade to more powerful GPUs
- Increase CPU cores and RAM
- Tune batch size and worker count

Horizontal Scaling

- Deploy multiple instances behind a load balancer
- Use Kubernetes for automatic scaling based on load
- Implement a queue system (like Redis) for handling request spikes

Geographic Distribution

- Deploy instances in multiple regions
- Use a global load balancer (like Cloudflare or AWS Global Accelerator)
- Consider edge caching for repeat transcriptions

Monitoring and Maintenance

Monitoring

- Implement Prometheus metrics (add to app.py)
- Set up Grafana dashboards
- Monitor:
  - GPU utilization and memory
  - Request latency
  - Queue length
  - Error rates

Logging

- Configure structured logging
- Use a centralized logging solution (ELK stack, Loki, etc.)
- Set up alerts for error patterns

Regular Maintenance

- Update the model as new versions are released
- Apply security patches
- Perform load testing after significant changes

Cloud Provider-Specific Deployment

AWS Deployment

1. Launch an EC2 instance with GPU:

```bash
aws ec2 run-instances \
  --image-id ami-0c55b159cbfafe1f0 \
  --instance-type g4dn.2xlarge \
  --key-name your-key-pair \
  --security-group-ids sg-1234567890abcdef0
```

2. Or deploy using ECS with GPU support:

```bash
# Create ECS cluster with GPU instances
aws ecs create-cluster --cluster-name seamless-m4t-cluster

# Register task definition with GPU requirements
aws ecs register-task-definition --cli-input-json file://task-definition.json

# Deploy service
aws ecs create-service --cluster seamless-m4t-cluster --service-name seamless-m4t-s
```

GCP Deployment

1. Create a VM instance with GPU:

```bash
gcloud compute instances create seamless-m4t-instance \
  --machine-type=n1-standard-8 \
  --zone=us-central1-a \
  --accelerator=type=nvidia-tesla-t4,count=1 \
  --boot-disk-size=100GB \
  --image-family=ubuntu-2004-lts \
  --image-project=ubuntu-os-cloud
```

2. Or deploy to GKE:

```bash
# Create GKE cluster with GPU nodes
gcloud container clusters create seamless-m4t-cluster \
  --accelerator type=nvidia-tesla-t4,count=1 \
  --machine-type=n1-standard-8 \
  --num-nodes=1 \
  --zone=us-central1-a

# Deploy to GKE
kubectl apply -f k8s/
```

Load Testing

Before going to production, run load tests to verify your deployment can handle the expected load:

```bash
# Using the provided load testing script
python scripts/load_test.py \
  --url http://your-server-ip \
  --concurrency 10 \
  --requests 100 \
  --audio-dir ./test_audio
```

Analyze the results to determine if your hardware can handle the expected load or if you need to scale up/out.

iOS Client Integration

To integrate with your iOS app, use the provided Swift client:

1. Add the `SeamlessM4TClient.swift` file to your Xcode project
2. Configure the client with your API URL:

```swift
let client = SeamlessM4TClient(baseURLString: "https://your-api-url.com")
```

3. Implement the transcription functionality:

swift

```swift
// Transcribe from file
client.transcribeAudio(from: audioFileURL) { result in
    switch result {
    case .success(let text):
        print("Transcribed text: \(text)")
    case .failure(let error):
        print("Error: \(error)")
    }
}

// Or record and transcribe
client.recordAndTranscribe { result in
    // Handle result
}
```

Conclusion

This deployment guide provides comprehensive instructions for setting up a high-performance Seamless-M4T API for Amharic speech-to-text translation. By following these guidelines and recommendations, you can create a scalable, reliable service that meets your performance requirements.

For specific questions or issues, refer to the troubleshooting section in the API documentation or open an issue on the project repository.