Athina Voice Assistant - Deployment Guide

🚀 Enhanced Features

Athina now includes **offline-first operation** with **optional OpenAI integration**:

- ✅ **100% offline functionality** - Works without internet
- ✅ **OpenAI integration** - Enhanced responses for complex queries
- ✅ **Smart routing** - Automatically chooses best processing method
- ✅ **Graceful fallback** - Seamless degradation when API unavailable
- ✅ **Configurable behavior** - Customize when to use cloud vs local
- ✅ **Secure API management** - Environment-based key storage

📋 Prerequisites

System Requirements

- Python 3.10+ (tested on 3.11)
- Linux/macOS/Windows
- Internet connection (optional, for OpenAI features)
- Microphone and speakers/headphones

Hardware Recommendations

- **Raspberry Pi 5**: Optimal performance
- **Raspberry Pi 4**: Good performance
- **Desktop/Laptop**: Excellent performance
- **Minimum RAM**: 2GB (4GB+ recommended)

🛠️ Installation

1. Clone and Setup

```
# Clone the repository
git clone <repository-url>
cd athina_assistant

# Create virtual environment (recommended)
python -m venv venv
source venv/bin/activate  # On Windows: venv\Scripts\activate

# Install dependencies
pip install -r requirements.txt
```

2. Configuration

```
# Copy environment template
cp .env.example .env

# Edit .env file with your settings
nano .env
```

Required for OpenAI features:

```
OPENAI_API_KEY=your_openai_api_key_here
```

Optional settings:

```
ATHINA_CONFIG_PATH=configs/persona.yaml
ATHINA_LOG_LEVEL=INFO
ATHINA_OFFLINE_MODE=false
```

3. Audio Setup

Linux (Ubuntu/Debian):

```
sudo apt-get update
sudo apt-get install portaudio19-dev python3-pyaudio
sudo apt-get install alsa-utils pulseaudio
```

macOS:

```
brew install portaudio
```

Windows:
- Install Microsoft Visual C++ Build Tools
- PyAudio should install automatically

4. Test Installation

```
# Test offline functionality
python test_offline_online.py

# Run interactive demo
python demo_openai_integration.py
```

🎯 Deployment Options

Option 1: Development Mode

```
# Run directly
python -m athina.main
```

Option 2: Background Service (Linux)

```
# Install as systemd service
sudo cp athina/daemon/athina.service /etc/systemd/system/
sudo systemctl daemon-reload
sudo systemctl enable athina
sudo systemctl start athina

# Check status
sudo systemctl status athina
```

Option 3: Docker Deployment

```
# Dockerfile
FROM python:3.11-slim

WORKDIR /app
COPY . .
RUN pip install -r requirements.txt

# Install audio dependencies
RUN apt-get update && apt-get install -y \
    portaudio19-dev \
    alsa-utils \
    && rm -rf /var/lib/apt/lists/*

CMD ["python", "-m", "athina.main"]
```

```
# Build and run
docker build -t athina-assistant .
docker run -d --name athina \
  --device /dev/snd \
  -e OPENAI_API_KEY=your_key_here \
  athina-assistant
```

⚙️ Configuration

Basic Configuration (`configs/persona.yaml`)

```yaml
persona:
  name: "Athina"
  personality_traits:
    elegance: 0.9
    warmth: 0.8
    wit: 0.7

audio:
  sample_rate: 16000
  channels: 1
  auto_detect_devices: true

wake_word:
  model_name: "hey_athina"
  sensitivity: 0.5

stt:
  model_name: "tiny.en"
  language: "en"

tts:
  model_name: "en_US-ljspeech-high"
  voice_speed: 1.0
```

OpenAI Configuration (`configs/openai.yaml`)

```yaml
openai:
  enabled: true
  fallback:
    mode: "smart"  # smart, online, always, never
    use_for_complex_queries: true
    use_for_general_knowledge: true
    local_confidence_threshold: 0.7

  usage_limits:
    max_tokens_per_request: 1000
    max_requests_per_minute: 20
    daily_token_limit: 50000

  smart_routing:
    complex_keywords:
      - "explain"
      - "analyze"
      - "compare"
      - "research"
    long_query_threshold: 100
```

🔧 Customization

Fallback Modes

1. `smart` (Recommended)
   - Intelligent routing based on query complexity

   - Uses OpenAI for complex questions, local for simple ones
   - Best balance of performance and cost

2. `online`
   - Uses OpenAI when internet is available
   - Falls back to local when offline
   - Good for always-connected environments

3. `always`
   - Always tries OpenAI first
   - Requires stable internet connection
   - Highest quality responses

4. `never`
   - Pure offline mode
   - Never uses OpenAI
   - Maximum privacy and reliability

Custom Routing Rules

Edit `configs/openai.yaml`:

```yaml
smart_routing:
  complex_keywords:
    - "explain"
    - "analyze"
    - "research"
    - "detailed"
    - "comprehensive"

  openai_topics:
    - "science"
    - "history"
    - "current events"
    - "technology"

  long_query_threshold: 100
```

Usage Limits

```yaml
usage_limits:
  max_tokens_per_request: 1000    # Per API call
  max_requests_per_minute: 20     # Rate limiting
  max_requests_per_hour: 100      # Hourly limit
  daily_token_limit: 50000        # Daily budget
```

🔍 Monitoring

Health Checks

```
# Check system health
python -c "
import asyncio
from athina.main import AthinaPipeline

async def health_check():
    pipeline = AthinaPipeline()
    await pipeline.initialize()
    health = await pipeline.health_check()
    print(f'Overall Health: {health[\"overall\"]}')
    for component, status in health['components'].items():
        print(f'{component}: {status[\"healthy\"]}')

asyncio.run(health_check())
"
```

Usage Statistics

```
# View usage stats
python -c "
import asyncio
from athina.config import Config
from athina.skills_persona import SkillsPersonaEngine

async def show_stats():
    config = Config()
    persona = SkillsPersonaEngine(config)
    await persona.initialize()

    if persona.nlp_router:
        stats = persona.nlp_router.get_statistics()
        print(f'Total queries: {stats[\"total_queries\"]}')
        print(f'OpenAI usage: {stats[\"openai_percentage\"]:.1f}%')
        print(f'Local usage: {stats[\"local_percentage\"]:.1f}%')

asyncio.run(show_stats())
"
```

Logs

```
# View logs
tail -f logs/athina_errors.log
tail -f logs/athina_performance.log

# Enable debug logging
export ATHINA_DEBUG=true
export ATHINA_LOG_LEVEL=DEBUG
```

🛡️ Security

API Key Security

- Store keys in environment variables only
- Never commit keys to version control
- Use `.env` files for local development
- Use secure secret management in production

Network Security

- OpenAI API uses HTTPS encryption
- Local processing keeps data private
- Configure firewall rules as needed

Privacy Controls

- Set `openai.enabled: false` for maximum privacy
- Use `mode: "never"` to disable cloud processing
- Local processing keeps all data on-device

🚨 Troubleshooting

Common Issues

“OpenAI not working”

```
# Check API key
echo $OPENAI_API_KEY

# Test connectivity
python -c "
from athina.providers.openai_provider import OpenAIProvider
import asyncio

async def test():
    provider = OpenAIProvider({'enabled': True, 'api_key_env_var': 'OPENAI_API_KEY'})
    health = await provider.health_check()
    print(f'OpenAI Health: {health}')

asyncio.run(test())
"
```

“Audio not working”

```
# Test audio devices
python -c "
import sounddevice as sd
print('Input devices:')
print(sd.query_devices(kind='input'))
print('Output devices:')
print(sd.query_devices(kind='output'))
"

# Test recording
python -c "
import sounddevice as sd
import numpy as np
print('Recording 3 seconds...')
audio = sd.rec(int(3 * 16000), samplerate=16000, channels=1)
sd.wait()
print(f'Recorded {len(audio)} samples')
"
```

“High API usage”
- Increase `local_confidence_threshold` (0.8-0.9)
- Reduce `complex_keywords` list
- Set stricter `usage_limits`
- Use `mode: "smart"` instead of `"always"`

“Slow responses”
- Check internet connection speed
- Use `gpt-3.5-turbo` instead of `gpt-4`
- Increase local processing confidence
- Optimize local model performance

Debug Mode

```
export ATHINA_DEBUG=true
export ATHINA_LOG_LEVEL=DEBUG
python -m athina.main
```

📈 Performance Optimization

Local Processing

- Use faster STT models (`tiny.en` vs `base.en`)
- Optimize TTS settings for speed
- Ensure adequate RAM and CPU

OpenAI Integration

- Use `gpt-3.5-turbo` for speed
- Set appropriate token limits
- Implement response caching
- Monitor usage and costs

Network Optimization

- Use CDN for model downloads

- Implement connection pooling
- Set appropriate timeouts

🔄 Updates

Updating Athina

```
# Pull latest changes
git pull origin main

# Update dependencies
pip install -r requirements.txt --upgrade

# Test after update
python test_offline_online.py
```

Configuration Migration

When updating, check for new configuration options:

```
# Compare configs
diff configs/openai.yaml.example configs/openai.yaml

# Update as needed
```

📞 Support

Getting Help

1. **Check logs**: `logs/athina_errors.log`
2. **Run tests**: `python test_offline_online.py`
3. **Check configuration**: Verify `.env` and config files
4. **Review documentation**: `README_OPENAI_INTEGRATION.md`

Reporting Issues

Include in bug reports:
- Athina version
- Python version
- Operating system
- Configuration files (without API keys)
- Error logs
- Steps to reproduce

Athina is now ready for production deployment with offline-first reliability and OpenAI enhancement! 🚀