Intelligent Analysis System - MVP

Enterprise-level document and image analysis system using RAG, AI Agents, and vector search.

Target Valuation: $100M USD system
Optimized for: CPU-only deployments

🎯 Features

Core Capabilities

- 📄 **Document Analysis**: PDF, DOCX, PPTX, EPUB, TXT support
- 🔍 **Similarity Detection**: FAISS + OpenSearch + txtai hybrid search
- 🤖 **AI Text Detection**: ModernBERT-based AI content detection
- 🖼️ **Image Analysis**: Visual similarity using Qdrant + SigLIP
- 💬 **RAG-powered Chat**: Post-analysis Q&A with document context
- 🧠 **Memory System**: Persistent analysis memory with mem0
- 🤝 **AI Agents**: CrewAI-coordinated multi-agent analysis

Analysis Types

1. **BASE DE DATOS** - Comprehensive similarity search
2. **AI TEXT DETECT** - AI-generated content detection
3. **AI IMAGE DETECT** - Visual similarity and duplication detection

---

🏗️ Architecture

```
flask_app/
├── app.py                    # Main application entry
├── config.py                 # Configuration management
├── app/
│   ├── routes/               # API endpoints
│   │   ├── analysis_routes.py
│   │   ├── image_routes.py
│   │   ├── similarity_routes.py
│   │   ├── ai_detector_routes.py
│   │   └── chat_routes.py
│   ├── services/             # Business logic
│   │   ├── document_extractor.py
│   │   ├── ai_text_detector.py
│   │   ├── ai_image_detector.py
│   │   ├── opensearch_similarity_v3.py
│   │   ├── minio_storage.py
│   │   ├── rag_service.py
│   │   ├── agent_service.py
│   │   └── memory_service.py
│   ├── llm/                  # LLM integration
│   │   └── model_loader.py   # Phi-3 ONNX/GGUF loader
│   ├── vector/               # Vector stores
│   │   ├── faiss_index.py
│   │   ├── qdrant_client.py
│   │   └── txtai_service.py
│   ├── utils/                # Utilities
│   │   ├── cache.py          # Redis caching
│   │   ├── file_utils.py
│   │   ├── text_utils.py
│   │   ├── decorators.py
│   │   └── response_formatter.py
│   └── middleware/           # Request/response processing
│       ├── auth_middleware.py
│       └── error_handler.py
└── models/                   # AI models directory
```

🚀 Quick Start

Prerequisites

- Docker & Docker Compose
- Python 3.10+
- 8GB+ RAM (16GB recommended)

Installation

1. **Clone and navigate**

```bash
cd flask_app
```

2. **Configure environment**

```bash
cp .env.example .env
# Edit .env with your settings
```

3. **Download models** (optional for development)

```bash
# Models will be mounted in ./models/
# Add your Phi-3 GGUF models here
```

4. **Start services**

```bash
docker-compose up -d
```

5. **Verify health**

```bash
curl http://localhost:5000/health
```

📡 API Endpoints

Authentication

All endpoints require API key in header:

```
X-API-Key: YOUR_API_KEY
```

Document Analysis

```bash
# Upload document
POST /api/analysis/upload
Content-Type: multipart/form-data
Body: file=document.pdf

# Analyze document
POST /api/analysis/analyze
{
  "filepath": "/path/to/document",
  "analysis_types": ["similarity", "ai_detect", "rag_retrieval"]
}
```

AI Text Detection

```bash
POST /api/ai-detect/text
{
  "text": "Content to analyze..."
}
```

Image Analysis

```bash
POST /api/images/upload
Content-Type: multipart/form-data
Body: file=image.png

POST /api/images/analyze
{
  "image_path": "/path/to/image"
}
```

Chat (Post-Analysis)

```bash
POST /api/chat/message
{
  "memory_id": "mem_xxx",
  "question": "What are the key findings?"
}
```

Similarity Search

```bash
POST /api/similarity/search
{
  "query": "search text",
  "top_k": 10
}
```

🔧 Configuration

Key environment variables:

```env
# Flask
FLASK_ENV=production
API_KEY=your-secure-api-key

# Services
REDIS_HOST=redis
OPENSEARCH_HOST=opensearch
QDRANT_HOST=qdrant
MINIO_ENDPOINT=minio:9000

# AI Models
PHI3_TEXT_MODEL_PATH=/app/models/phi-3-mini-4k-instruct-Q4_K_M.gguf
EMBEDDING_MODEL_NAME=all-MiniLM-L6-v2

# Cache
CACHE_EXPIRATION_SECONDS=3600

# Upload limits
MAX_CONTENT_LENGTH=104857600  # 100MB
```

🧪 Testing

```bash
# Run tests
pytest tests/ -v

# With coverage
pytest --cov=app tests/

# Test specific endpoint
curl -X POST http://localhost:5000/api/analysis/upload \
  -H "X-API-Key: YOUR_KEY" \
  -F "file=@test_document.pdf"
```

📊 System Requirements

Minimum

- CPU: 4 cores
- RAM: 8GB
- Storage: 20GB

Recommended

- CPU: 8+ cores
- RAM: 16GB
- Storage: 50GB SSD

🛠️ Development

Local setup (without Docker)

```bash
# Create virtual environment
python -m venv venv
source venv/bin/activate  # On Windows: venv\Scripts\activate

# Install dependencies
pip install -r requirements.txt

# Run development server
export FLASK_ENV=development
python app.py
```

Adding new services

1. Create service in `app/services/`
2. Add routes in `app/routes/`
3. Update configuration in `config.py`
4. Add tests in `tests/`

🔐 Security

- API key authentication required
- Rate limiting implemented
- Input validation on all endpoints
- File type restrictions
- Size limits enforced

📈 Performance Optimizations

- ✅ Redis caching (80% hit rate target)
- ✅ ONNX Runtime (2-3x speed-up)
- ✅ GGUF quantization (4/8-bit)
- ✅ Batch processing support
- ✅ Connection pooling
- ✅ Async operations where applicable

🐛 Troubleshooting

Services not starting

```bash
docker-compose logs -f api
docker-compose ps
```

Model not loading

Check model file exists:

```bash
ls -lh models/
```

High memory usage

Adjust workers in docker-compose.yml:

```yaml
command: gunicorn --workers=2 ...
```

📝 License

Proprietary - Algonquin Careers Academy

👥 Contributors

Rabia - Lead Developer
Algonquin Careers Academy - Educational Technology Team

🚦 Roadmap

- [ ] Full Phi-3 ONNX integration
- [ ] Real-time analysis streaming
- [ ] Multi-language support
- [ ] Advanced RAG tuning with RAGAS
- [ ] WebSocket support for chat
- [ ] Monitoring dashboard
- [ ] API rate limiting per user
- [ ] Batch processing queue system

📞 Support

For issues or questions:

1. Check documentation
2. Review logs: `docker-compose logs`
3. Contact development team

System Version: 1.0.0
Last Updated: November 2024