🎬 PreenCut - AI-Powered Video Intelligence Platform

PreenCut is an enterprise-grade AI-powered video intelligence platform that automatically analyzes and extracts meaningful segments from audio/video content using advanced speech recognition and large language models. Transform your media content into viral-ready clips optimized for social media platforms with professional-grade architecture and deployment capabilities.

🎉 What's New - Enterprise Production Ready!

PreenCut has been **completely refactored** for enterprise production use while maintaining all original functionality:

🏗️ Production Architecture

- **Clean Service-Oriented Design**: Dependency injection with testable interfaces
- **Environment-Based Configuration**: Professional `.env` configuration system
- **Structured JSON Logging**: Production-grade logging with performance metrics
- **Custom Exception System**: Standardized error handling with clear error codes
- **Health Monitoring**: Built-in health checks and performance monitoring
- **Docker Support**: Containerized deployment with Docker Compose
- **Backwards Compatibility**: Gradual migration path for existing code

🚀 Enhanced AI Features

- **Social Media Optimization**: Platform-specific content for TikTok, Instagram, YouTube Shorts
- **Viral Content Analysis**: AI-powered viral potential scoring and engagement prediction
- **Advanced Relevancy Ranking**: 1-10 scoring system for content relevance and quality
- **Smart Topic Extraction**: Context-aware segment extraction with precise timestamps
- **Enhanced Summaries**: Detailed 20-80 word descriptions with word count metrics
- **Hook Generation**: Attention-grabbing opening lines for viral content

✨ Core Features

🧠 AI-Powered Content Analysis

- **Advanced Speech Recognition**: WhisperX and Faster-Whisper integration
- **Large Language Model Processing**: Multiple LLM support (Ollama, DeepSeek, DouBao)
- **Natural Language Queries**: Find segments using descriptive commands
- **Intelligent Segmentation**: AI-driven content analysis and categorization
- **Batch Processing**: Process multiple files with unified analysis
- **Re-analysis Capability**: Experiment with different prompts without reprocessing

📱 Social Media Optimization

- **Platform-Specific Optimization**:
  - **TikTok**: 15-180s clips with viral hooks and trending focus
  - **Instagram Reels**: 15-90s clips emphasizing aesthetic appeal
  - **YouTube Shorts**: 15-60s clips optimized for retention and value
  - **Universal**: Balanced optimization for all platforms
- **Viral Content Features**:
  - Engagement scoring (1-10 scale)
  - Viral potential assessment (Low/Medium/High)
  - Hook text generation (10-15 words)
  - Platform-specific hashtag suggestions
  - Content style selection (Educational/Entertainment/Inspirational)

Advanced Content Discovery

- **Relevancy Ranking**: Intelligent scoring system for content importance
- **Topic Extraction**: Find all instances of specific topics with context
- **Smart Segmentation**: Automatic segment extension for complete narratives
- **Word Count Analytics**: Content density assessment for engagement optimization
- **Composite Scoring**: Weighted algorithms combining relevance and engagement

🔧 Professional Features

- **RESTful API**: Complete API for integration and automation
- **Health Monitoring**: Built-in health checks and performance metrics
- **Scalable Architecture**: Service-oriented design for horizontal scaling
- **Security**: Input validation, file upload security, environment-based secrets
- **Error Handling**: Comprehensive error handling with user-friendly messages
- **Performance Optimization**: GPU acceleration and configurable batch processing

📁 Project Architecture

```
PreenCut/
├── 🏗️ config/                    # Environment-based configuration
│   ├── settings.py               # Type-safe configuration classes
│   └── __init__.py               # Legacy compatibility exports
├── 🔧 core/                      # Core infrastructure
│   ├── logging.py                # Structured JSON logging
│   ├── exceptions.py             # Custom exception hierarchy
│   ├── dependency_injection.py   # DI container for services
│   └── __init__.py
├── 🎯 services/                  # Business logic services
│   ├── interfaces.py             # Service contracts
│   ├── video_service.py          # Video processing service
│   ├── file_service.py           # File management service
│   ├── llm_service.py            # LLM integration service
│   ├── speech_recognition_service.py # Speech recognition service
│   └── __init__.py
```

```
├── 🛠️ utils/                       # Organized utilities
│   ├── file_utils.py               # File operations
│   ├── time_utils.py               # Time formatting utilities
│   ├── media_utils.py              # Media processing utilities
│   └── __init__.py
├── 🌐 web/                         # Web interface & API
│   ├── gradio_ui.py                # Enhanced Gradio web interface
│   └── api.py                      # RESTful API endpoints
├── 📦 modules/                     # Processing modules
│   ├── video_processor.py          # Legacy video processor
│   ├── video_processor_refactored.py # Refactored video processor
│   ├── llm_processor.py            # Legacy LLM processor
│   ├── llm_processor_refactored.py   # Refactored LLM processor
│   ├── text_aligner.py             # Text alignment with WhisperX
│   ├── processing_queue.py         # Task queue management
│   └── speech_recognizers/         # Speech recognition implementations
├── 🧪 tests/                       # Comprehensive test suite
│   ├── run_all_tests.py            # Main test runner
│   ├── test_enhanced_features.py   # Enhanced features tests
│   ├── test_social_media_download.py # Social media tests
│   ├── final_validation.py         # Architecture validation
│   └── ...                         # Additional test files
├── 📚 docs/                        # Documentation
│   ├── PRODUCTION_DEPLOYMENT.md    # Production deployment guide
│   ├── REFACTORING_SUMMARY.md      # Architecture documentation
│   └── ...                         # Additional documentation
├── 📊 logs/                        # Application logs
├── 🗂️ temp/                        # Temporary processing files
├── 📤 output/                      # Generated output files
├── .env.example                    # Environment configuration template
├── main.py                         # Application entry point
└── requirements.txt                # Python dependencies
```

🚀 Quick Start

1. Environment Setup

```
# Clone the repository
git clone https://github.com/roothch/PreenCut.git
cd PreenCut

# Copy environment template
cp .env.example .env

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

2. Install Dependencies

```
# Create virtual environment (recommended)
python -m venv venv
source venv/bin/activate  # Linux/Mac
# or
venv\Scripts\activate  # Windows

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

3. Install System Dependencies

```
# Ubuntu/Debian
sudo apt install ffmpeg

# CentOS/RHEL
sudo yum install ffmpeg

# macOS (using Homebrew)
brew install ffmpeg

# Windows: Download from https://ffmpeg.org/
```

4. Configure LLM Services

Set up your LLM API keys in the `.env` file:

```
# Example for DeepSeek and DouBao
DEEPSEEK_V3_API_KEY=your_deepseek_api_key
DOUBAO_1_5_PRO_API_KEY=your_doubao_api_key

# Ollama configuration (if using Ollama)
OLLAMA_HOST=localhost
OLLAMA_PORT=11434
```

5. Run Application

```
# Start the application
python main.py

# Access the web interface at:
# http://localhost:8860/web
```

🔧 Configuration

Environment Variables (`.env`)

```
# Application Settings
APP_ENV=development
DEBUG=true
PORT=8860
HOST=0.0.0.0

# GPU Configuration
WHISPER_DEVICE=cuda
WHISPER_BATCH_SIZE=16
WHISPER_MODEL_SIZE=large-v3
WHISPER_GPU_IDS=0,1

# File Processing
MAX_FILE_SIZE=10737418240  # 10GB
MAX_FILE_NUMBERS=10
TEMP_FOLDER=./temp
OUTPUT_FOLDER=./output

# Ollama Configuration
OLLAMA_HOST=localhost
OLLAMA_PORT=11434
OLLAMA_TIMEOUT=300
OLLAMA_KEEP_ALIVE=5m

# Logging
LOG_LEVEL=INFO
LOG_FORMAT=json
LOG_FILE=logs/app.log

# API Keys
DEEPSEEK_V3_API_KEY=your_api_key_here
DOUBAO_1_5_PRO_API_KEY=your_api_key_here
```

Legacy Configuration Support

Existing code continues to work with deprecation warnings:

```
# Legacy (still works with warnings)
from config import TEMP_FOLDER, WHISPER_MODEL_SIZE

# New recommended approach
from config import get_config
config = get_config()
temp_folder = config.file.temp_folder
```

📖 Usage Guide

Web Interface

1. **Start Application**: Run `python main.py`
2. **Open Browser**: Navigate to `http://localhost:8860/web`
3. **Upload Files**: Support for mp4, avi, mov, mkv, ts, mxf, mp3, wav, flac
4. **Configure Options**:
    - Select LLM model and Whisper model size
    - Add custom analysis prompts (optional)
    - Configure GPU settings if available
5. **Process Content**: Click "Bắt đầu xử lý" to analyze content
6. **Review Results**: View enhanced results with:
    - Detailed summaries (20-80 words)
    - Word count metrics
    - Relevancy scores (1-10 scale)
    - Viral potential indicators

Enhanced Features

1. Topic Extraction with Relevancy Ranking

1. Complete basic analysis first
2. Go to "Trích xuất phân đoạn theo chủ đề" tab
3. Enter specific topic or keyword
4. Select LLM model for analysis
5. Click "Trích xuất phân đoạn theo chủ đề"
6. View ranked results sorted by relevance score

2. Social Media Optimization

1. Complete basic analysis first
2. Go to "Tối ưu hóa cho mạng xã hội" tab
3. **Select Platform**: TikTok/Instagram Reels/YouTube Shorts/Universal
4. **Enter Topic**: Trending topic or content theme
5. **Choose Style**: Educational/Entertainment/Inspirational/Tutorial/Trending
6. **Set Clip Count**: Maximum number of clips (3-10)
7. **Generate Content**: Click "🎯 Tạo nội dung viral"
8. **Review Results**: View platform-optimized content with:
    - Hook text for attention-grabbing openings
    - Platform-specific hashtags
    - Viral potential scoring
    - Engagement metrics
9. **Export**: Use "Cutting Options" tab to export as ZIP or merged video

3. Advanced Content Analysis Features

- **Enhanced Summaries**: Detailed content descriptions with context
- **Word Count Analytics**: Content density for engagement optimization
- **Composite Scoring**: Relevance × 0.6 + Engagement × 0.4

- **Smart Segment Extension**: Automatic extension for complete narratives
- **Fallback Mechanisms**: Keyword matching when LLM analysis fails

RESTful API

Upload File

```
POST /api/upload
Content-Type: multipart/form-data

# Response
{
  "file_path": "/path/to/uploaded/file.mp4"
}
```

Create Processing Task

```
POST /api/tasks
Content-Type: application/json

{
  "file_path": "/path/to/uploaded/file.mp4",
  "llm_model": "DeepSeek-V3-0324",
  "whisper_model_size": "large-v2",
  "prompt": "Extract viral content segments for TikTok"
}

# Response
{
  "task_id": "unique_task_id"
}
```

Query Task Results

```
GET /api/tasks/{task_id}

# Response
{
  "status": "completed",
  "files": ["/path/to/processed/file.wav"],
  "result": [
    {
      "filename": "file.wav",
      "segments": [
        {
          "start": 1.145,
          "end": 9.329,
```

```
            "summary": "Detailed content summary with context",
            "tags": ["tag1", "tag2"],
            "word_count": 45,
            "relevance_score": 8.5,
            "engagement_score": 7.2,
            "viral_potential": "High"
          }
        ]
      }
    ]
}
```

🏭 Production Deployment

Docker Deployment (Recommended)

```
# Quick start with Docker Compose
docker-compose up -d

# View logs
docker-compose logs -f

# Scale services for high availability
docker-compose up -d --scale app=3
```

Manual Production Setup

1. **Server Requirements**:
    - Ubuntu 20.04+ or similar Linux distribution
    - Python 3.8+, FFmpeg, CUDA drivers (for GPU acceleration)
    - Minimum 8GB RAM (16GB+ recommended)
    - 50GB+ disk space for temporary files
2. **Production Configuration**:

```
# Production .env settings
APP_ENV=production
DEBUG=false
PORT=8860
HOST=0.0.0.0

# Security
CORS_ORIGINS=https://yourdomain.com
RATE_LIMIT_ENABLED=true
RATE_LIMIT_PER_MINUTE=30

# Performance
WHISPER_BATCH_SIZE=32
```

```
WORKER_PROCESSES=4
ENABLE_CACHING=true

# Logging
LOG_LEVEL=INFO
LOG_FORMAT=json
LOG_FILE=/var/log/preencut/app.log
```

3. **System Service Setup**:

```
# Create system user
sudo useradd -r -s /bin/false preencut
sudo mkdir -p /var/lib/preencut/{temp,output}
sudo mkdir -p /var/log/preencut
sudo chown -R preencut:preencut /var/lib/preencut /var/log/preencut

# Configure systemd service
sudo systemctl enable preencut
sudo systemctl start preencut
```

Health Monitoring

```
# Health check endpoint
curl http://localhost:8860/health

# View structured logs
tail -f /var/log/preencut/app.log | jq .

# Monitor GPU usage
nvidia-smi
watch -n 1 nvidia-smi
```

⚡ Performance Optimization

GPU Configuration

```
# High-end GPUs (RTX 4090, A100)
WHISPER_BATCH_SIZE=32
WHISPER_MODEL_SIZE=large-v3
WHISPER_COMPUTE_TYPE=float16

# Mid-range GPUs (RTX 3070, 4070)
WHISPER_BATCH_SIZE=16
WHISPER_MODEL_SIZE=large-v2
WHISPER_COMPUTE_TYPE=float16

# CPU-only systems
```

```
WHISPER_DEVICE=cpu
WHISPER_BATCH_SIZE=4
WHISPER_MODEL_SIZE=base
```

Performance Tips

- **Use WhisperX** for faster processing on longer content
- **Use Faster-Whisper** for shorter segments with better accuracy
- **Adjust batch sizes** based on available VRAM
- **Enable SSD storage** for temp files when processing large videos
- **Configure worker processes** based on CPU cores for API endpoints
- **Enable caching** for repeated analysis of similar content

🧪 Testing & Validation

Run All Tests

```
# Comprehensive architecture validation
python tests/final_validation.py

# Service integration tests
python tests/test_service_integration.py

# Enhanced features tests
python tests/test_enhanced_features.py

# Social media optimization tests
python tests/test_social_media_download.py

# Run complete test suite
python tests/run_all_tests.py
```

Test Coverage

- ✅ Configuration system validation
- ✅ Logging and monitoring systems
- ✅ Service layer and dependency injection
- ✅ Enhanced AI features and social media optimization
- ✅ File handling and error management
- ✅ API endpoints and integration
- ✅ Performance and scalability

🔄 Migration from Legacy Code

Gradual Migration Strategy

The refactoring maintains backwards compatibility while providing a clear migration path:

Phase 1: Update configuration (immediate)

```
# Replace direct config imports
from config import get_config
config = get_config()
```

Phase 2: Adopt new utilities (short-term)

```
# Use organized utility modules
from utils.file_utils import generate_safe_filename
from utils.time_utils import seconds_to_hhmmss
```

Phase 3: Service architecture (long-term)

```
# Use dependency injection for services
from core.dependency_injection import get_container
container = get_container()
video_service = container.get_video_service()
```

🛡️ Security & Production Considerations

Security Features

- **Environment-based secrets**: All API keys and sensitive data in environment variables
- **File upload validation**: Strict file type, size, and content validation
- **Input sanitization**: Comprehensive input validation and sanitization
- **CORS configuration**: Configurable cross-origin request handling
- **Rate limiting**: Built-in protection against abuse and DoS attacks
- **Error sanitization**: Prevent information leakage in error messages

Best Practices

```
# Production security settings
APP_ENV=production
DEBUG=false
ALLOWED_ORIGINS=["https://yourdomain.com"]
MAX_FILE_SIZE=5368709120  # 5GB for production
ENABLE_RATE_LIMITING=true
LOG_LEVEL=INFO  # Avoid DEBUG in production
```

🔮 Future Roadmap

Planned Enhancements

1. **Database Integration**: PostgreSQL/MySQL for persistent task storage and analytics
2. **Message Queue**: Redis/RabbitMQ for horizontal scaling and background processing
3. **Advanced Analytics**: User behavior tracking, content performance metrics
4. **API Versioning**: RESTful API with proper versioning and documentation
5. **Microservices**: Split into smaller, focused services for cloud deployment
6. **Kubernetes Support**: Container orchestration for enterprise deployment
7. **Real-time Processing**: WebSocket support for live video analysis
8. **Advanced AI Models**: Integration with latest multimodal AI models

Community Contributions

- Model optimization and fine-tuning for specific use cases
- Additional language support for international content
- New export formats and platform integrations
- Performance benchmarking and optimization
- Enhanced UI/UX improvements

🤝 Contributing

Development Setup

```
# Fork and clone the repository
git clone https://github.com/yourusername/PreenCut.git
cd PreenCut

# Create development environment
python -m venv venv
source venv/bin/activate
pip install -r requirements.txt

# Create feature branch
git checkout -b feature/your-feature-name
```

Code Quality Standards

- **Type Hints**: All new code must include comprehensive type annotations
- **Error Handling**: Robust exception handling with custom error types
- **Logging**: Structured logging for all operations and business events
- **Testing**: Unit tests and integration tests for new functionality
- **Documentation**: Update documentation for new features and changes

Development Workflow

1. Create feature branch from main
2. Implement changes with comprehensive tests
3. Ensure type safety and code quality
4. Update documentation and examples
5. Submit pull request with detailed description

📞 Support & Troubleshooting

Getting Help

- **Documentation**: Check comprehensive documentation first
- **Error Messages**: Review error suggestions and context information
- **Logs**: Check `logs/app.log` for detailed structured information
- **GitHub Issues**: Create issues for bugs or feature requests

Common Issues & Solutions

Performance Issues

- **GPU Memory**: Reduce `WHISPER_BATCH_SIZE` if getting CUDA out of memory
- **CPU Usage**: Adjust `WORKER_PROCESSES` based on system capabilities
- **Disk Space**: Ensure adequate space in `TEMP_FOLDER` for large files
- **Network**: Check Ollama connectivity and API key validity

File Upload Issues

- **File Size**: Check `MAX_FILE_SIZE` configuration
- **File Format**: Ensure file format is in supported list
- **Permissions**: Verify write permissions to temp and output folders

API Integration Issues

- **Authentication**: Verify API keys are correctly set in environment
- **Network**: Check firewall settings and network connectivity
- **Rate Limits**: Monitor API usage and adjust rate limiting if needed

Performance Monitoring

```
# Monitor system resources
htop
iotop
nvidia-smi

# Check application logs
tail -f logs/app.log | grep "ERROR"
grep "Performance:" logs/app.log | jq .

# Test API endpoints
curl -X GET http://localhost:8860/health
curl -X POST http://localhost:8860/api/upload -F "file=@test.mp4"
```

📜 License

This project is licensed under the MIT License. See the LICENSE file for details.

🎉 Production Ready!

PreenCut is now enterprise-ready with:

- ✅ **Clean Architecture**: Service-oriented design with dependency injection
- ✅ **Professional Logging**: Structured JSON logs with performance metrics
- ✅ **Environment Configuration**: Production-ready configuration system
- ✅ **Docker Support**: Containerized deployment with health checks
- ✅ **Enhanced AI Features**: Social media optimization and viral content analysis
- ✅ **Comprehensive Documentation**: Complete setup and migration guides
- ✅ **Backwards Compatibility**: Gradual migration path for existing code
- ✅ **Type Safety**: Full type hints throughout the codebase
- ✅ **Error Handling**: Custom exceptions with clear error messages
- ✅ **Testing Support**: Comprehensive test suite and validation
- ✅ **Security**: Production-grade security features and best practices

The application maintains all original AI-powered video processing capabilities while providing enterprise-grade reliability, maintainability, scalability, and cutting-edge social media optimization features.

Ready for production deployment and viral content creation! 🚀