

# **Amir Aeiny**

**AI/ML Developer | Python - NLP - Audio Transcription**

GitHub: <https://github.com/DarkOracle10>

LinkedIn: <https://www.linkedin.com/in/amir-aeiny-dev>

## **Project: Persian Audio Transcriber**

AI/ML Production Tool - Audio Transcription and NLP System

## Technology Stack

| Backend         | Python 3.11+, AsyncIO                                  |
|-----------------|--|
| AI/ML           | OpenAI Whisper, Faster-Whisper, Google Speech, PyTorch |
| NLP             | Hazm (Persian), NLTK                                   |
| GPU             | CUDA, torch GPU acceleration                           |
| Audio           | FFmpeg, librosa, pydub                                 |
| Data            | pandas, numpy  |
| CLI             | argparse, tqdm, colorama                               |
| Formats         | MP3, WAV, M4A, FLAC, MP4, OGG, WEBM, AAC, WMA, AIFF    |
| Testing         | pytest, unittest                                       |
| Version Control | Git, GitHub  |

## Project Statistics

|                         |                        |
|-------------------------|------------------------|
| Lines of Code           | 2,500+                 |
| Transcription Engines   | 3                      |
| Supported Audio Formats | 10+                    |
| Output Formats          | 3                      |
| Supported Languages     | 90+                    |
| GPU Acceleration        | Yes, with CPU fallback |
| Batch Processing        | Parallel execution     |
| Status                  | Production Ready       |

## Project Overview

A production-ready AI-powered audio transcription system combining Faster-Whisper, OpenAI Whisper, and Google Speech with intelligent fallback. Features GPU acceleration with automatic CPU fallback, Persian text normalization via Hazm, batch processing with live progress, and multiple outputs including timestamped JSON and SRT. Built for large-scale transcription with robust error handling, logging, and scalable architecture.

## Key Features

1. Multi-engine transcription with automatic fallback
2. CUDA acceleration with CPU fallback

3. Persian NLP via Hazm for normalization
4. Batch processing with real-time progress
5. Multiple outputs: TXT, JSON with timestamps, SRT
6. Universal format support with auto conversion
7. Comprehensive error handling and logging
8. Live progress indicators and status updates
9. Quality tuning: language detection and model selection
10. Production-ready architecture and tooling

## Skills Demonstrated

### AI and Machine Learning

- OpenAI Whisper integration and optimization
- Faster-Whisper acceleration and tuning
- Google Speech Recognition API integration
- Multi-engine fallback orchestration
- GPU acceleration with PyTorch and CUDA
- Model performance benchmarking

### Natural Language Processing

- Persian text normalization with Hazm
- Tokenization and lemmatization
- Text cleaning and standardization
- Multi-language support (90+ languages)
- Character encoding handling for Persian script

### Backend Development

- Python 3.11+ with AsyncIO
- Multi-threaded and parallel pipelines
- REST-ready architecture and patterns

- Robust error handling and logging

## Audio Processing

- FFmpeg format conversion and extraction
- Audio quality analysis and optimization
- Waveform analysis and metadata extraction
- Real-time audio streaming support

## Performance Optimization

- CUDA-accelerated execution with CPU fallback
- Batch processing for throughput
- Memory-aware handling for large files
- Resource usage monitoring and tuning

## DevOps and Production

- Modular architecture and clean code
- Production-grade logging and monitoring
- Cross-platform compatibility (Win, Mac, Linux)
- Dependency and virtual environment management

## Similar Projects I Can Build

### Audio and Video Processing

- Podcast transcription with diarization
- Video subtitle generation and sync
- Audio content indexing and search
- Multi-language transcription platforms
- Real-time transcription APIs

## NLP and Language Processing

- Persian text analysis and summarization
- Multi-language translation systems
- Sentiment analysis for Persian and Arabic
- Named entity recognition for Persian
- Text classification pipelines

## AI and ML Systems

- Custom GPT-based content tools
- AI document processing systems
- Image recognition and classification
- Chatbots with NLP capabilities
- Recommendation systems

## Automation and Data Processing

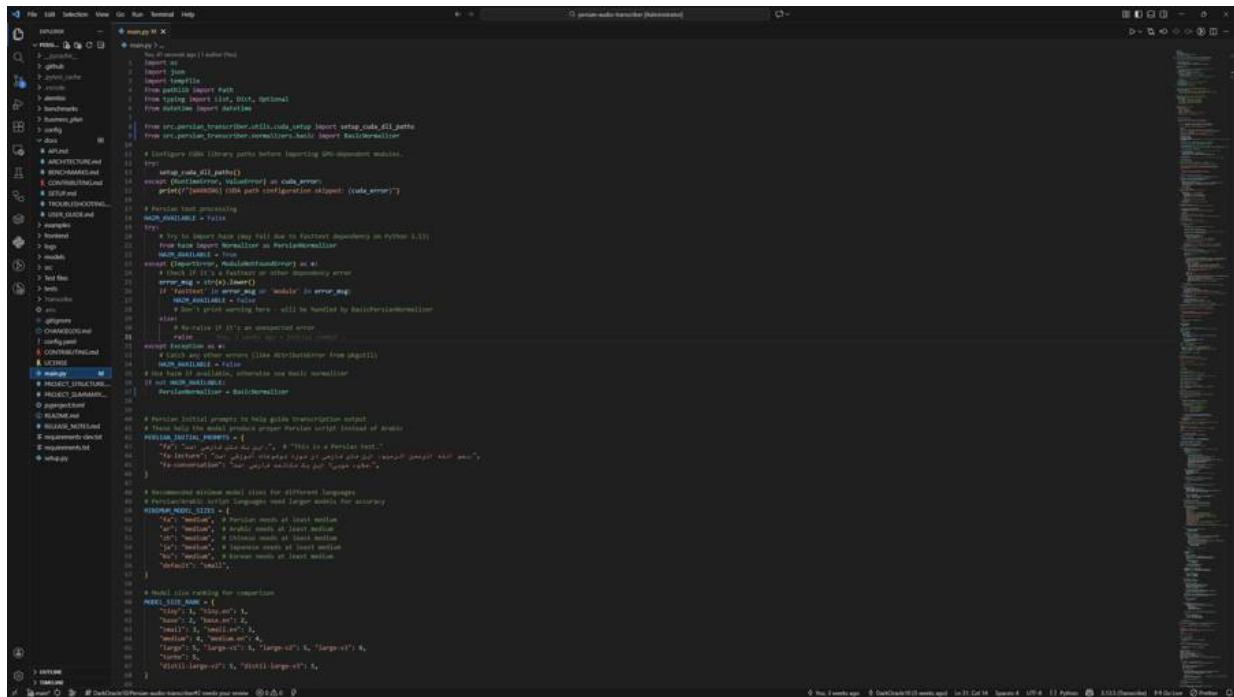
- Batch file processing systems
- Data pipeline automation (ETL)
- Report generation automation
- Workflow automation tools
- API-driven data collection

## API Development

- RESTful wrappers for AI services
- Webhook integration systems
- Third-party API integration
- Microservices-ready designs
- Rate limiting and caching strategies

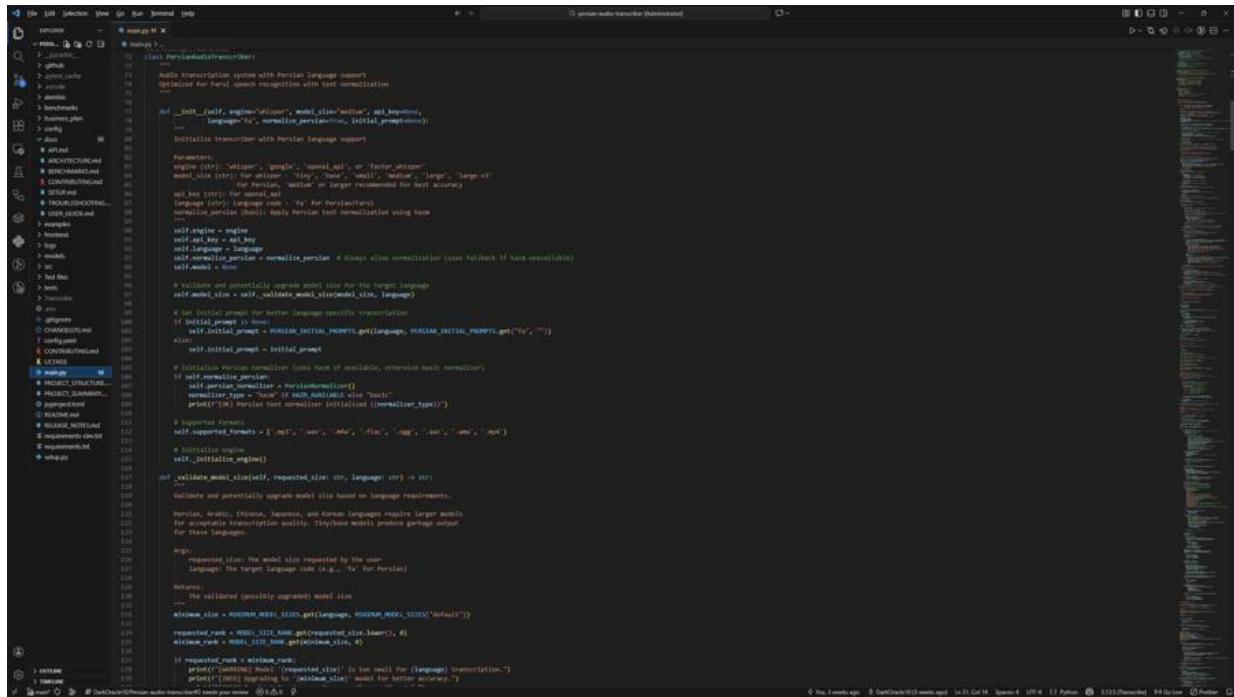
# Screenshots

## Code Structure and Engine Logic



```
#!/usr/bin/env python3
# coding: utf-8
# Copyright 2023 Amir Aein. All Rights Reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
# ======================================================================
# This file contains the main logic for the Persian AI engine.
# It handles text normalization and transcription for Persian and English languages.
# The engine supports three model sizes: small, medium, and large.
# It includes a fallback mechanism for Persian text normalization using English models.
# The code uses argparse for command-line arguments and os.path for file paths.
# It also imports various modules from the engine package and its submodules.
# The main function initializes the engine and performs a transcription task.
# The PersianTextNormalizer class is the core component for text normalization.
# It loads models based on the selected size and performs normalization using them.
# The engine class handles the transcription process, validating model sizes and
# using fallback mechanisms if necessary.
# The code is well-documented with comments explaining the logic and variable names.
```

## GPU Acceleration and Fallback



```
#!/usr/bin/env python3
# coding: utf-8
# Copyright 2023 Amir Aein. All Rights Reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
# ======================================================================
# This file contains the main logic for the Persian AI engine.
# It handles text normalization and transcription for Persian and English languages.
# The engine supports three model sizes: small, medium, and large.
# It includes a fallback mechanism for Persian text normalization using English models.
# The code uses argparse for command-line arguments and os.path for file paths.
# It also imports various modules from the engine package and its submodules.
# The main function initializes the engine and performs a transcription task.
# The PersianTextNormalizer class is the core component for text normalization.
# It loads models based on the selected size and performs normalization using them.
# The engine class handles the transcription process, validating model sizes and
# using fallback mechanisms if necessary.
# The code is well-documented with comments explaining the logic and variable names.
# GPU Acceleration and Fallback
# Check for CUDA availability
# If CUDA is available, use PyTorch's cuda_* functions
# Otherwise, fall back to CPU operations
```

## Generated Outputs (TXT, JSON, SRT)

The screenshot shows a Microsoft Word document with the title "Project Plan" at the top. The document contains a table with two columns: "Task" and "Start Date". The tasks listed include:

- Project Kick-off Meeting
- Market Research Phase
- Competitor Analysis
- Product Concept Development
- Design Phase
- Prototyping
- Beta Testing Phase
- User Feedback Analysis
- Final Product Launch

Each task has a corresponding start date in the second column. The document also includes several sections and tables related to project management, such as "Project Structure", "Project Summary", "Risks and Issues", and "Release Notes". A footer at the bottom right indicates the document was created on 10/10/2018.

# Sample Persian Transcription Output