

Video Interpolation

Students: Kevin Puig, Tomas Ambrosini, Nathaniel Rowe, Tahjae Shamari Chamberlain & Nathan Castillo

Instructor/Faculty: Dr. Seyed Masoud Sadjadi, Florida International University

PROBLEM

Low Frame Rate in Online Video Content:

Modern video content often suffers from low frame rates, leading to choppy playback and a subpar viewing experience. While high frame rate (HFR) video offers smoother motion, it is costly and time-intensive to produce. Our goal was to create a lightweight and user-friendly web application that allows users to upload a low-frame-rate video and automatically generate a smoother, interpolated version using neural networks — all without specialized hardware or software.

CURRENT SYSTEM

Current video interpolation tools typically require:

- High-end computing resources (GPUs),
- Installation of complex software packages,
- Technical knowledge to configure neural networks or pipelines.

Most solutions are desktop-based and require manual steps, limiting accessibility to everyday users. Cloud-based options often come with expensive subscription fees and do not offer transparency or control over how the video is processed. There is no accessible, open-source platform offering neural network-powered interpolation through a simple browser interface — until now.

REQUIREMENTS

Functional Requirements:

- Users must be able to upload video files through a web interface.
- The system must assign a unique task ID to each video.
- Videos must be processed via a trained neural network to generate interpolated outputs.
- Status updates (queued, processing, processed, or errored) must be available.
- Users must be able to download the final processed video.

Non-Functional Requirements:

- The system must respond to user requests within a reasonable time frame (<2s for uploads/status checks).
- Video processing must happen asynchronously.
- The interface must be accessible and intuitive for non-technical users.
- The architecture must support modular codebases across front-end (React), back-end (Java REST API), and processing engine (Python PyTorch).

SYSTEM DESIGN

- Frontend (React): Provides a simple web interface for video uploads and downloads.
- Backend (Java REST API): Handles file operations and exposes RESTful endpoints.
- Processor (Python + PyTorch): Runs the neural network to generate interpolated videos.
- File-Based Queue System: Organizes videos into queue/, processing/, processed/, and error/ folders, allowing loose coupling between Java and Python layers.

This modular architecture allows each component to function independently and ensures scalability, maintainability, and easier debugging.

OBJECT DESIGN

Object-Oriented Architecture & Component Interaction

Our software architecture leverages object-oriented principles on the backend through Java and modular design on the Python side.

- The **Java backend** is divided into:
 - A **Controller layer** that handles all incoming HTTP requests.
 - A **Service layer** responsible for business logic and file operations.
 - A file-based system acting as a pseudo-repository, organizing videos into folders by task status (/queue, /processing, /processed, /error).
- Each video task is treated as a self-contained object with a unique ID, file path, and current status. These attributes are used to manage the processing pipeline.
- The **Python neural network module** treats each video as a batch of frames, applies interpolation using trained model parameters, and outputs a new interpolated video object that replaces the original in the system.
- All REST endpoints interact with these objects through defined interfaces, ensuring a clear and testable separation of concerns.

IMPLEMENTATION

- Built using a full-stack architecture: React (frontend), Java SpringBoot (REST API), and Python with PyTorch (neural processor).
- Videos are uploaded via the UI, routed through the REST API, queued, and processed asynchronously by the neural network.
- Final outputs are returned to the user via a download link after processing.

All system layers communicate through well-defined endpoints and a shared file structure to keep processing modular and efficient.

VERIFICATION

- Endpoints tested: Upload, status, download, and error handling for various video formats.
- Interpolation quality: Verified through visual inspection and real-world testing with multiple video types.
- Edge cases handled: Corrupted files, unsupported formats, and frame inconsistencies.
- Neural Network performance: Tuned over several iterations; tested for smoothness, accuracy, and consistency.

SUMMARY

We developed a modular web app that allows users to upload videos and receive smoother, high-frame-rate versions using neural network interpolation. Through React, Java REST APIs, and PyTorch, we created a scalable solution with an intuitive interface. This project deepened our understanding of full-stack development, AI integration, and teamwork.

REFERENCES

- PyTorch – <https://pytorch.org>
- React – <https://reactjs.org>
- Spring Boot – <https://spring.io>
- FFmpeg – <https://ffmpeg.org>

ACKNOWLEDGEMENT

The material presented in this poster is based upon the work supported by Seyed Masoud Sadjadi. We thank Seyed Masoud Sadjadi for his assistance and mentorship that I/we received throughout the senior design project.