Oregon State University

College of Engineering
Electrical Engineering and Computer Science

Software Innovation Lab

Master's Project Report

Chalida (Anita) Ruangrotsakun

LabelFlicks

Defended September 1st, 2023

Commencement June 2023

Abstract

Using supervised machine learning (ML) to train a computer vision model typically requires human annotators to label objects in images and video. Given a large training dataset, this can be labor intensive, presenting a significant bottleneck in the model-development process. LabelFlicks is an open-source desktop application that aims to address this pain point with three helpful ML-assisted features: (1) a streamlined preprocessing pipeline to convert videos into a series of frames, (2) pre-labeling of video frames using an object detection model pre-trained on the COCO dataset that ships with LabelFlicks, (3) an ML-assisted human-in-the-loop workflow for correcting bounding box labels. For each frame of the video(s) provided by the user, LabelFlicks produces a text file containing labeled bounding boxes in the COCO annotation format. These datasets can be analyzed (e.g. for finding biases in data slices) or used to train or finetune an object detection model of your choice using model training tools such as PyTorch or TensorFlow.

Acknowledgments

I would like to thank my advisor, Dr. Will Braynen, for his valuable software engineering advice and helping to improve my development process throughout the project.

I would like to thank my committee members, Dr. Margaret Burnett and Dr. Stefan Lee, for teaching excellent classes in human-computer interaction and deep learning and for accommodating my last-minute requests.

I would like to thank Dr. Minsuk Kahng for taking a chance on me when I was an undergrad and nurturing my interest in HCI + AI research.

Finally, I would like to thank my friends and family for all their unconditional love and support. I would never have made it this far without them.

Table of Contents

1 Introduction **1**

2 Existing Solutions **3**

3 Proposed Solution **5**

4 Implementation **7**

4.1 Desktop Application and User Interface 8

4.2 PostgreSQL Database 26

4.3 FastAPI Server 27

4.4 Scope and Current Limitations 38

5 Future Work **40**

5.1 Custom Dataset Creation 40

5.2 Use IML to Train While Labeling 43

5.3 Testing Computer Vision Models 44

6 Conclusion **46**

References **47**

List of Figures

2.1 Example Bounding Boxes 3

3.1 Human-in-the-loop 5

4.1 LabelFlicks Dependency Diagram 7
4.2 Home screen with project creation modal 9
4.3 Home screen with listed projects 9
4.4 Home screen class diagram 10
4.5 Video upload screen with upload modal 11
4.6 Video upload screen with listed videos 12
4.7 Video upload screen class diagram 13
4.8 Preprocessing screen 14
4.9 Preprocessing screen class diagram 15
4.10 Labeling screen initially loaded 16
4.11 Labeling screen portion with individual box selection 17
4.12 Labeling screen with some human-reviewed frames 18
4.13 Labeling screen after navigating with labeling timeline 18
4.14 Labeling screen with some labeling timelines marked "hidden" 19
4.15 Labeling screen with create label modal 20
4.16 Labeling screen with label dropdown 21

4.17 Labeling screen with some corrected labels 21
4.18 Labeling screen AI-assistance button 22
4.19 Labeling screen class diagram 23
4.20 Export labels screen showing local save path 24
4.21 Export labels screen class diagram 25
4.22 Database entity-relationship diagram 26
4.23 Backend class diagram 27
4.24 FastAPI endpoints 28
4.25 Sequence diagram: home screen 29
4.26 Sequence diagram: video upload screen 30
4.27 Sequence diagram: preprocessing screen 32
4.28 Sequence diagram: initial loading of the labeling screen 33
4.29 Sequence diagram: navigating between frames on the labeling screen 34
4.30 Sequence diagram: AI-assistance on the labeling screen 35
4.31 Sequence diagram: labeling screen create and delete interactions 36
4.32 Sequence diagram: export labels 37

1. Introduction

Artificial intelligence (AI) has become a much-hyped research area in the last several years, driven mainly by the impressive results from machine learning (ML) and deep learning (DL) methods. Nowadays, people can create digital art using generative adversarial networks, write reports with the help of large language models, and even ride in self-driving cars powered by computer vision models.

A lot of labeled data is typically needed to train these highly capable models using supervised DL methods. Data labeling in some domains can be done automatically – for example, text data can be scraped from all over the Internet and chopped into phrases to train language models to predict the next most likely word in a sentence. Labeling images and videos, however, is a more complex task and requires human labor. The data labeling step necessary for most advanced DL models that consume complex visual data has become a bottleneck in the overall DL development process.

LabelFlicks is an open-source desktop application that aims to address this pain point with three helpful ML-assisted features: (1) a streamlined preprocessing pipeline to convert videos into a series of frames, (2) pre-labeling of video frames using an object detection model pre-trained on the COCO dataset that ships with LabelFlicks, and (3) an ML-assisted human-in-the-loop workflow for correcting bounding box labels. For each frame of the video(s) provided by the user, LabelFlicks produces a text file containing labeled bounding boxes in the COCO annotation format. These datasets can be analyzed (e.g. for finding biases in data slices) or used to train or finetune an object detection model of your choice using model training tools such as PyTorch or TensorFlow. The "Future Work" section of this report will discuss how LabelFlicks could be extended to support creating computer vision datasets without assuming a baseline detection model, as well as iteratively train or test a trained model using human-in-the-loop workflows.

This report will first provide a brief survey of the computer vision landscape and existing video labeling tools (Section 2). I will then give an exhaustive overview of the technical

implementation details of the LabelFlicks application (Sections 3 and 4). The "Future Work" section will discuss how LabelFlicks could be extended to support creating computer vision datasets without assuming a baseline detection model, as well as how LabelFlicks could support iteratively train or test a trained model using human-in-the-loop workflows (Section 5).

2. Existing Solutions

Computer vision is a field of AI that focuses on creating models that can process visual data, such as images and videos. Object detection is one of the core tasks that computer vision models are trained to do. It's used in various applications every day–for example, helping your phone find people and objects in your photo library or helping self-driving cars identify road signs and pedestrians. Object detection models are trained to identify regions of images, represented with bounding boxes, and classify each as a specific type of object, such as a tree, chair, or table as in the example provided in Figure 2.1.

Figure 2.1: Example image containing labeled bounding boxes for detected trees, tables, and chairs from the TensorFlow Object Detection Tutorial. [1]

The process for developing a supervised ML model generally follows this pattern: collect the training data, prepare the data, train the model, evaluate and deploy the model. ML models learn from examples, and the more high-quality training data they can get, the better they

will learn. Training object detection models typically requires many thousands of example images that have been manually annotated with bounding boxes and classification labels, which is a hugely time-consuming and expensive process [2].

The AI community has recognized that data labeling can be a huge bottleneck in the overall process and many data labeling tools have developed over the last several years to address this pain point.

Many companies offer advanced video annotation features to automate parts of the video annotation process, but they are restricted behind paywalls. Companies, such as SuperAnnotate, V7, Labelbox, Supervisely, CVAT, and Dataloop, offer video annotation tools that can be used for all video file formats, can annotate videos while playing them at their native frame rate, can interpolate annotations between two points in time, and can support real-time collaboration. Some of these platforms even offer ML-assisted labeling workflows and a way to hire professional labeling teams. These are highly advanced tools but they are inaccessible to most non-enterprise customers, such as tech hobbyists, researchers, and students.

Several open-source video annotation platforms exist as alternatives, such as Universal Data Tool, UltimateLabeling, and Label Studio, and they generally offer similar features as the paid solutions: there are ways to label online or offline, with or without collaborators, and for a variety of computer vision tasks. Label Studio even offers a way to attach an "ML backend" to train a model while labeling, but it requires some coding to set up. None of these open-source tools offer an ML-assisted human-in-the-loop labeling workflow out of the box.

3. Proposed Solution

LabelFlicks aims to address the data labeling issue in computer vision pipelines by providing an open-source implementation of a video annotation tool with an AI-assisted, human-in-the-loop labeling workflow with no code setup required.

Human-in-the-loop means that humans are involved in the AI development process, including the data collection, preprocessing, model training, and evaluation phases [3]. Human-in-the-loop systems can be thought of as a way to selectively include meaningful, human feedback into automated systems such that the overall system is more efficient and takes advantage of the strengths of both human capabilities and computer processors [4], as shown in Figure 3.1. Human-in-the-loop includes concepts such as Interactive Machine Learning (IML), which is "an interaction paradigm in which a user or user group iteratively builds and refines a mathematical model to describe a concept through iterative cycles of input and review" [5].

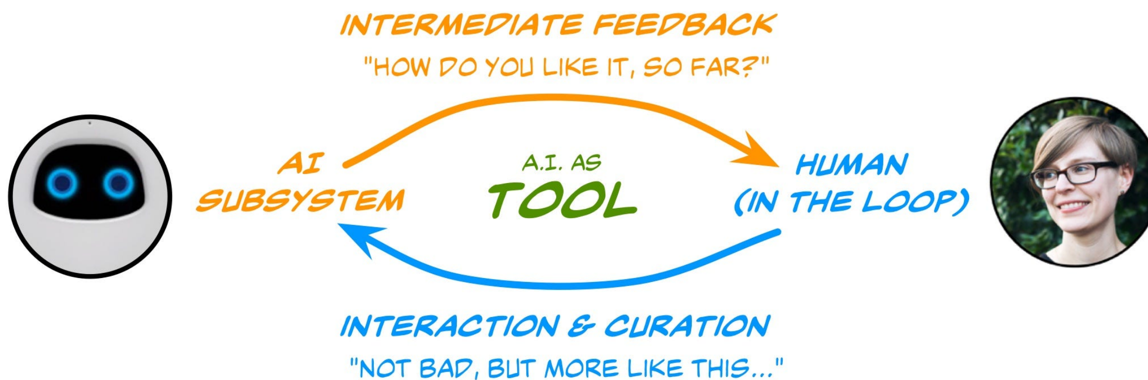

Figure 3.1: General outline of a human-in-the-loop system

LabelFlicks applies an iterative review approach similar to IML, but it applies it to the data labeling process rather than the model training process. Instead of iteratively correcting the model's predictions during the training process in order to improve the final model, LabelFlicks allows the user to iteratively correct the label predictions made by a small, assistive ML model in order to make later label predictions more accurate and accelerate

the labeling process. The end goal is still the same for both systems: to reduce the effort required by the human user as the system is iteratively trained to better understand the target concepts. Desmond et al. from IBM Research conducted user studies to investigate the impact of AI-assisted data labeling and found that "the accuracy of human labeling can be improved with relatively weak AI support" [6]. LabelFlicks aims to improve labeling accuracy through the use of a relatively weak AI-supported interface as well, but for visual data rather than text data as was used in the study.

The Implementation section will dive into the technical details regarding how the LabelFlicks proof of concept (PoC) was constructed. The Future Work section will describe how LabelFlicks could expand its capabilities to implement model-training IML in the future, along with other technical improvements that would make it suitable for other use cases as well.

4. Implementation

The bird's eye view of the LabelFlicks architecture is provided in Figure 4.1. The LabelFlicks code is contained in two GitHub repositories – LabelFlicks-backend and LabelFlicks-desktop – representing the backend and frontend components of the whole system. The backend implements a REST API that is consumed by the frontend, meaning the API serves as a common "contract" for expected inputs and outputs. The REST API abstraction of the backend allows these two major components to be decoupled so that sub-components, such as the JavaScript framework or database, can be modified or replaced without affecting the other major component. Both frontend and backend components will be described in detail in the following subsections.

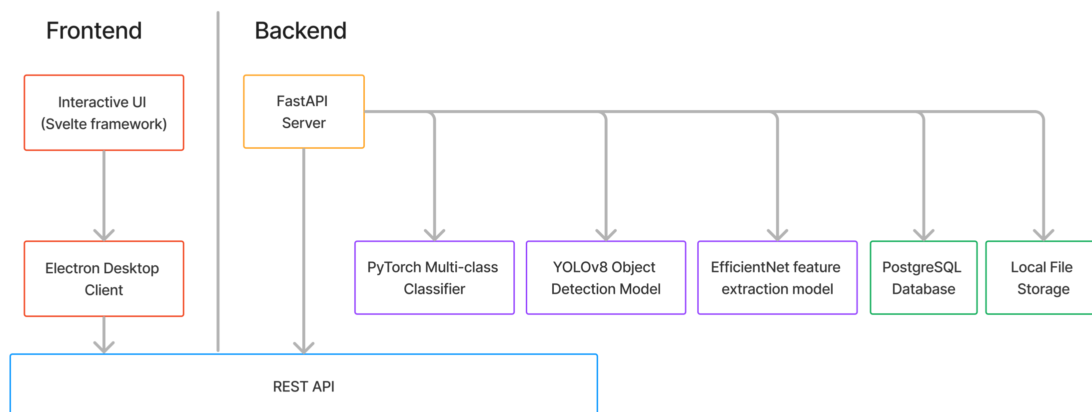

Figure 4.1: LabelFlicks Dependency Diagram outlining the major components of the frontend and backend

4.1 Desktop Application and User Interface

The frontend component of LabelFlicks is a desktop application built using Electron and Svelte. Electron is an open-source framework for building cross-platform apps using HTML and JavaScript, and Svelte was the chosen JavaScript framework for more quickly building the UI. The following subsections will provide a walkthrough of the interactions available on each main screen as well as a class diagram of their dependencies.

4.1.1 Home Screen

The LabelFlicks home screen is simple and streamlined, as seen in Figure 4.3, containing only the name of the application, a list of current projects, and a big green button for creating a new project. When a user clicks on the button, a modal appears (shown in Figure 4.2), allowing them to provide a name and create a new project. Future improvements could enhance the design of this landing page, but for the PoC, the main focus of this page is on the creation of a new project or selection of an existing project.

The home screen is also the first place where the user sees the application's navigation bar along the top of the screen. After the home screen, there are four steps that the user will follow to produce their object detection dataset and they are labeled accordingly on the navigation bar: (1) Upload Videos, (2) Preprocess Videos, (3) Label Videos, and (4) Export Labels.

The class diagram for the home screen is provided in Figure 4.4 and it shows the key dependencies required to render the page.

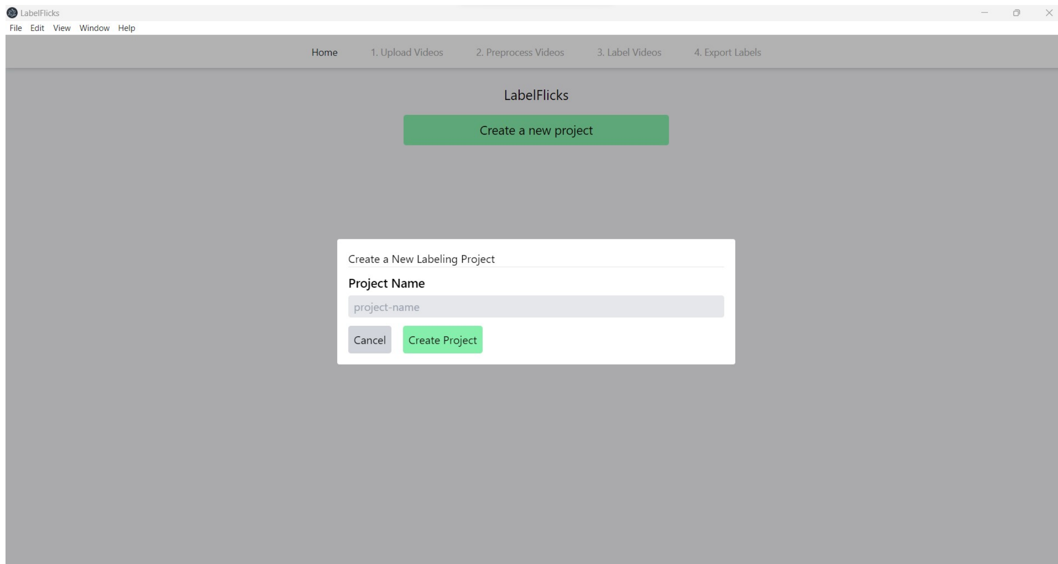

Figure 4.2: Home screen with modal for creating a new project

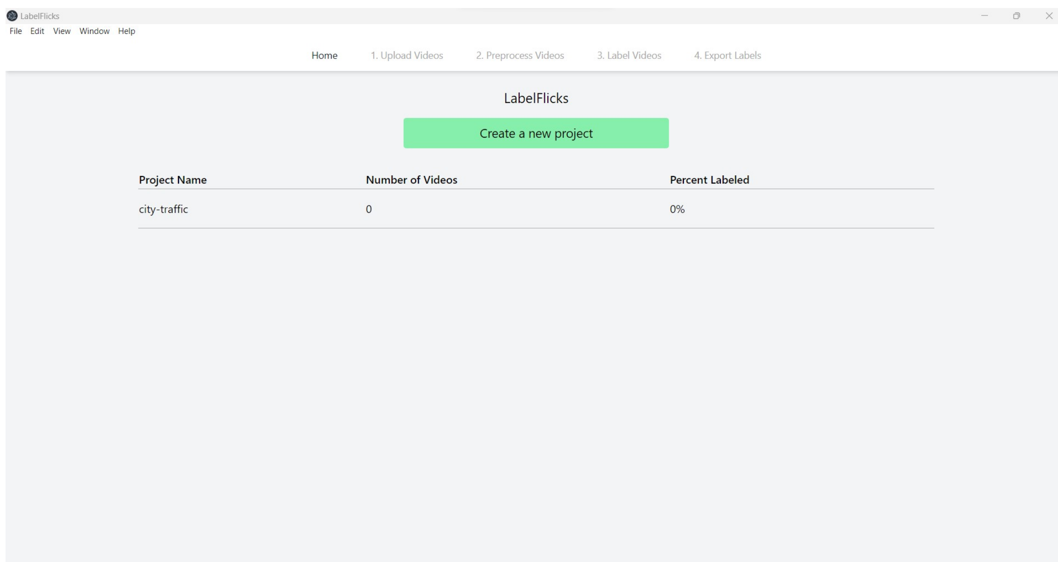

Figure 4.3: Home screen with one project listed

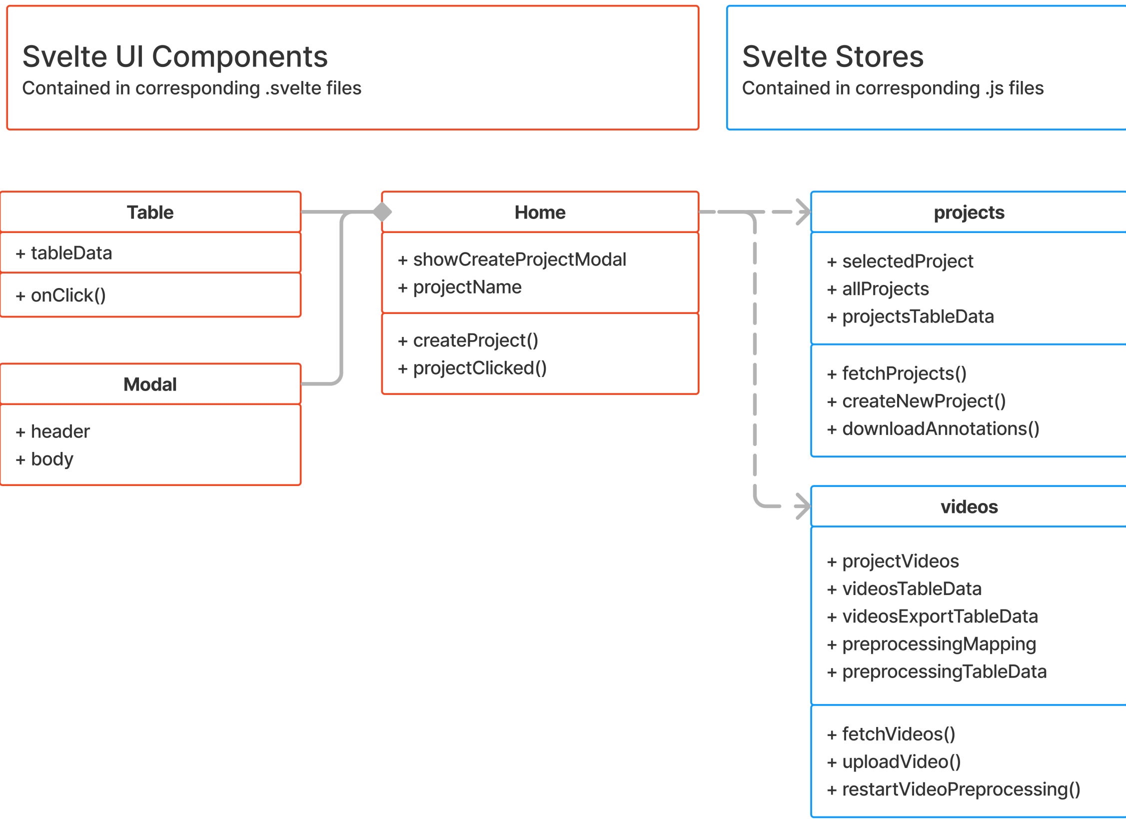

Figure 4.4: Home screen class diagram, which makes use of Svelte components and stores.

4.1.2 Video Upload Screen

Once the user has selected a project, the remaining links in the navigation bar along the top of the screen become active and they are directed to the video upload screen (shown in Figure 4.6). This screen displays the project name, the videos that exist in the project, and a few buttons. The user can click the "upload a video" button to open a modal (shown in Figure 4.5) that will allow them to select an MP4 file from their local file system. The "next step" button will be activated and turns green only after the user has uploaded at least one video file, allowing the user to progress to the next step in the process.

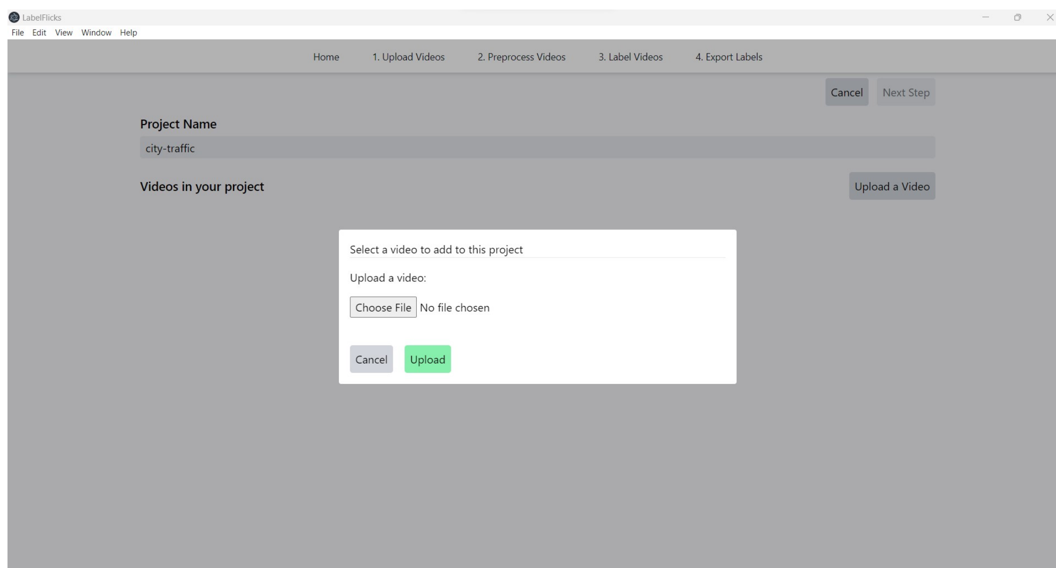

Figure 4.5: Video upload screen with modal for uploading a new MP4 file

The class diagram for the video upload screen is provided in Figure 4.7 and it shows the key dependencies required to render the page.

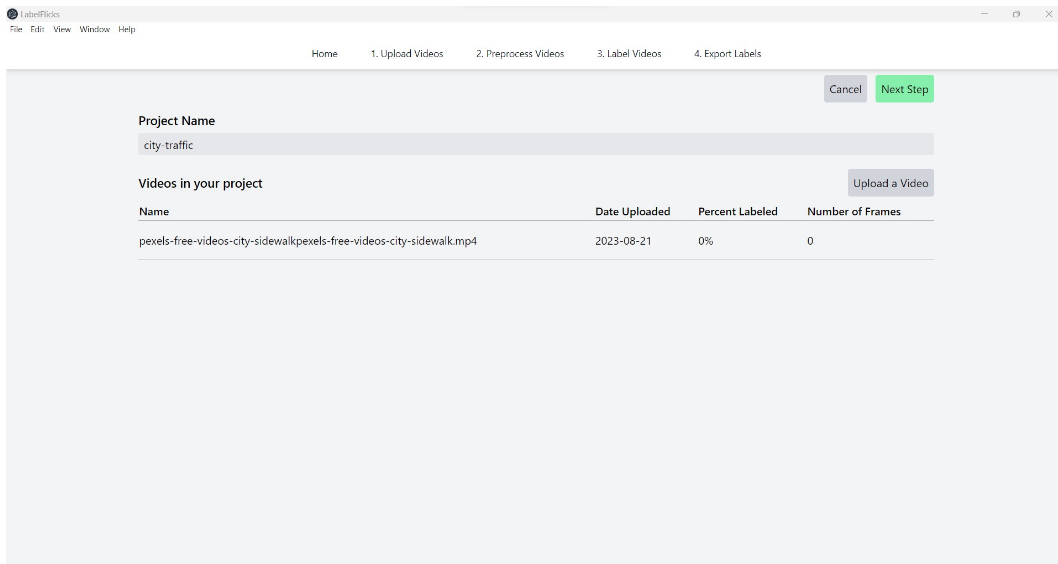

Figure 4.6: Video upload screen with one uploaded video listed on the screen

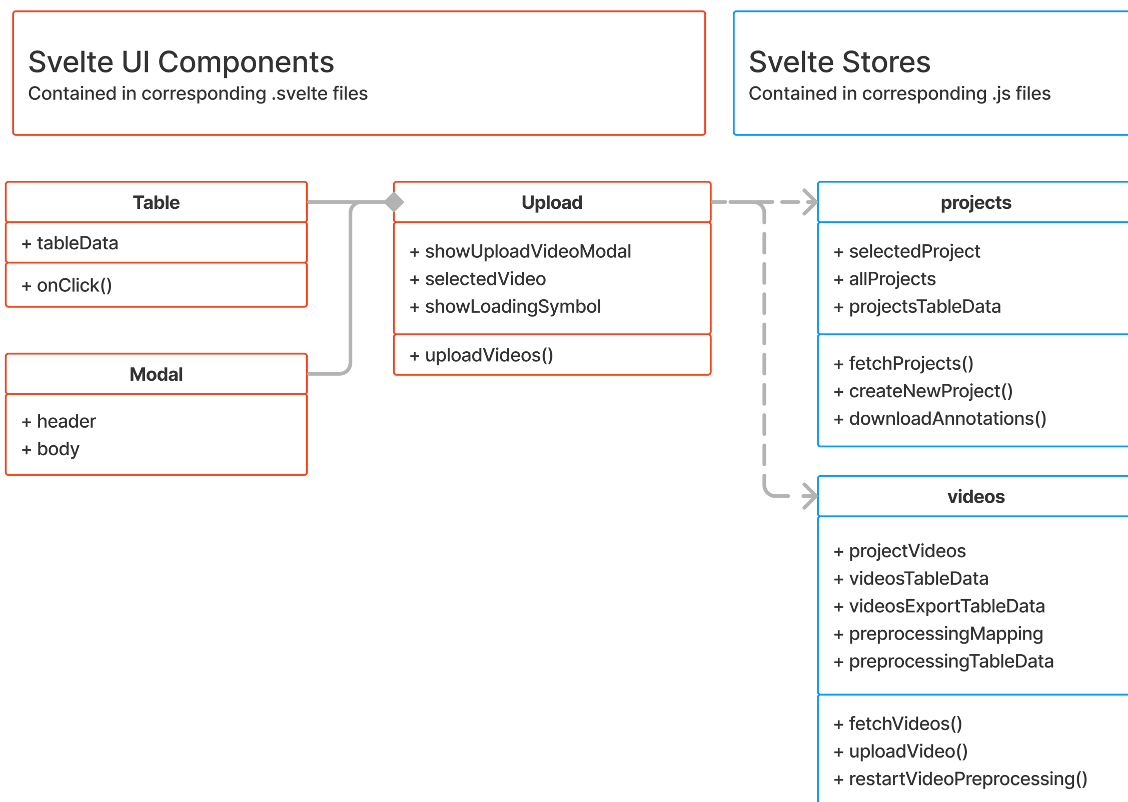

Figure 4.7: Video upload screen class diagram, which makes use of Svelte components and stores.

4.1.3 Preprocessing Screen

Once the videos are uploaded, they enter the preprocessing stage and the users are directed to the preprocessing screen (shown in Figure 4.8), which shows them the preprocessing state of each of their videos. This screen of the application employs the long-polling strategy to ping the backend every 20 seconds to check on the status of each video. Once all videos have reached the "Completed" status, the "Next Step" button will turn green and allow the user to progress to the next screen.

The class diagram for the preprocessing screen is provided in Figure 4.9 and it shows the key dependencies required to render the page.

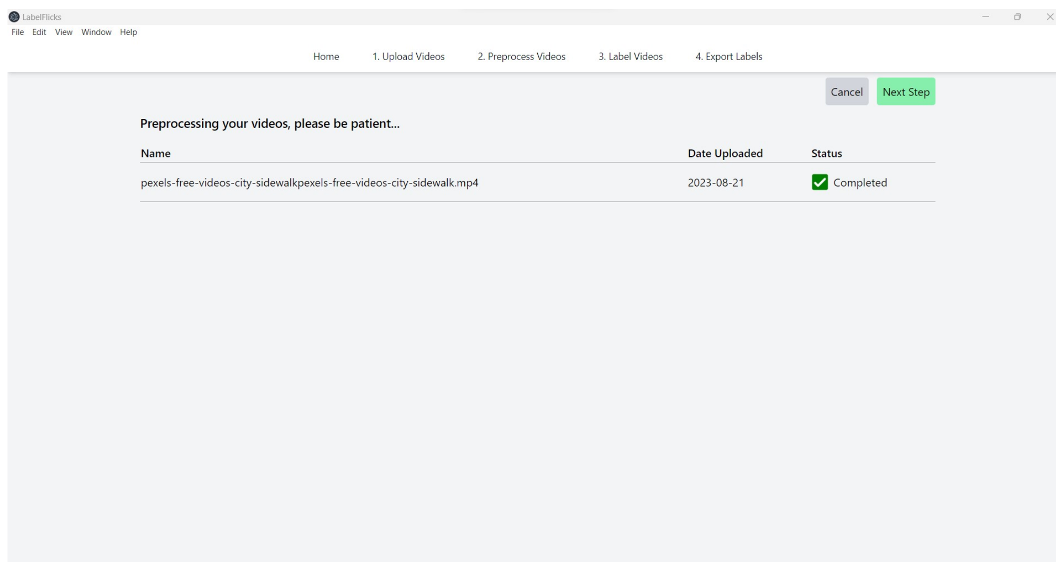

Figure 4.8: Preprocessing screen with the uploaded video showing the completed status

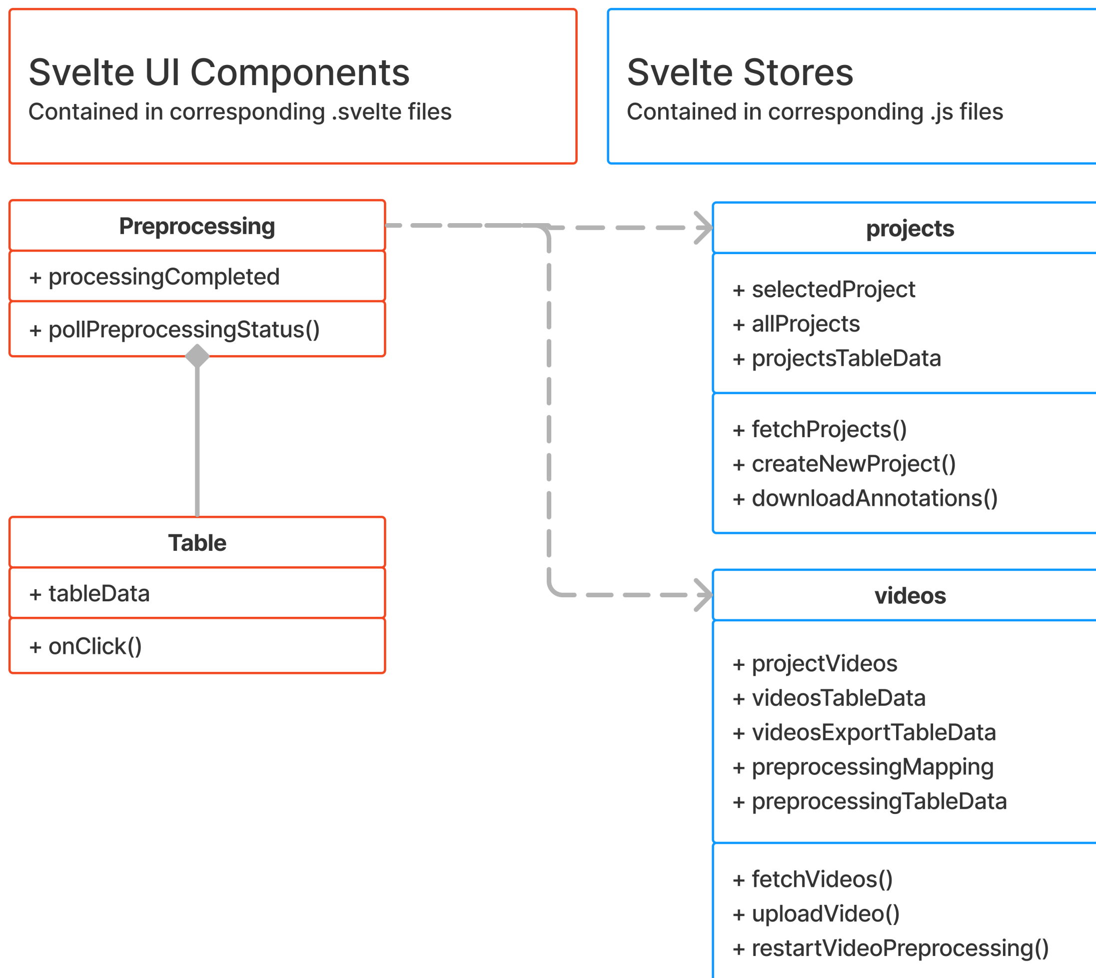

Figure 4.9: Preprocessing screen class diagram, which makes use of Svelte components and stores.

4.1.4 Labeling Screen

The labeling screen (shown in Figure 4.10) consists of two main parts: the frames player and labeling timelines.

The frames player makes up the left half of the screen and presents the user with a way to flip through the frames as if they were watching the video but at a slower speed and with labeled bounding boxes overlaid. The controls for the frames player are located below in the form of the “back”, “play/pause”, and “next” buttons, as seen in Figure 4.11. There is also a narrow gray bar and a text caption below the frames player to indicate approximately how far into the video the user has gone. There is a dropdown menu above the frames player to allow the user to select a video they had uploaded to their project. Finally, there is a checklist below the frames player to allow users to hide or show individual bounding boxes. Users may click on the “delete” label at the bottom left of any bounding box to permanently remove it from the project if they do not need it.

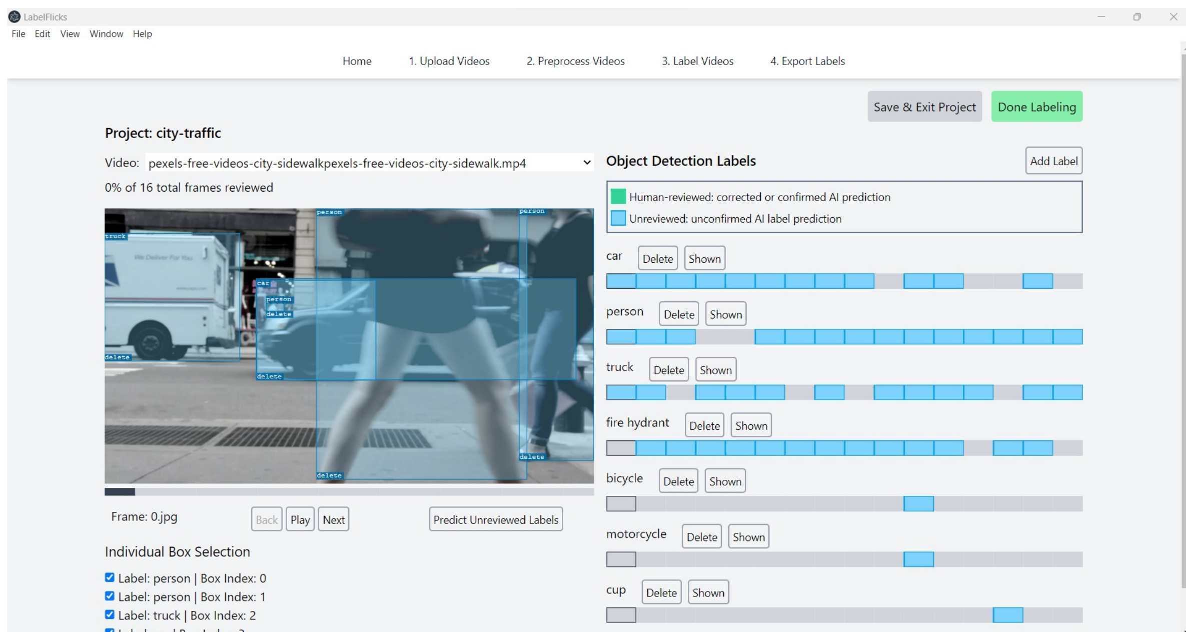

Figure 4.10: Labeling screen with zero human-reviewed frames indicated by the text statement in the top left, the blue segments in the Labeling Timelines, and the blue overlaid bounding boxes

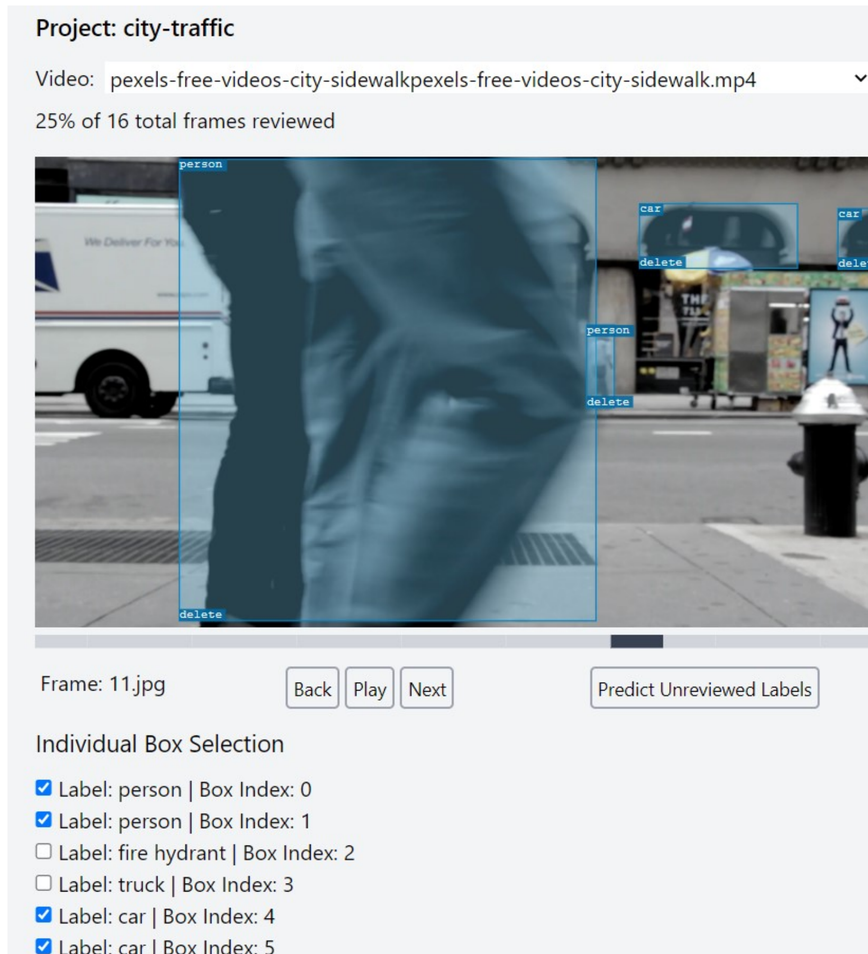

Figure 4.11: Labeling screen frames player with individual frames selected

The labeling timelines make up the right half of the screen. Each timeline corresponds to one label that belongs to the project. These labels include both entities predicted by LabelFlicks during preprocessing and labels created by the user. Each timeline consists of segments, each segment representing a frame. Each frame segment is colored blue if it contains bounding boxes that the LabelFlicks AI predicted as the timeline's label. The frame segments turn green (shown in Figure 4.12) once the user proceeds to the next frame after making any necessary corrections. The green color indicates that the user either corrected the label predictions or that the user looked at it and the label required no change. The color scheme was restricted to these two colors so the user could just focus on the end goal of getting all of the frames to the human-reviewed state.

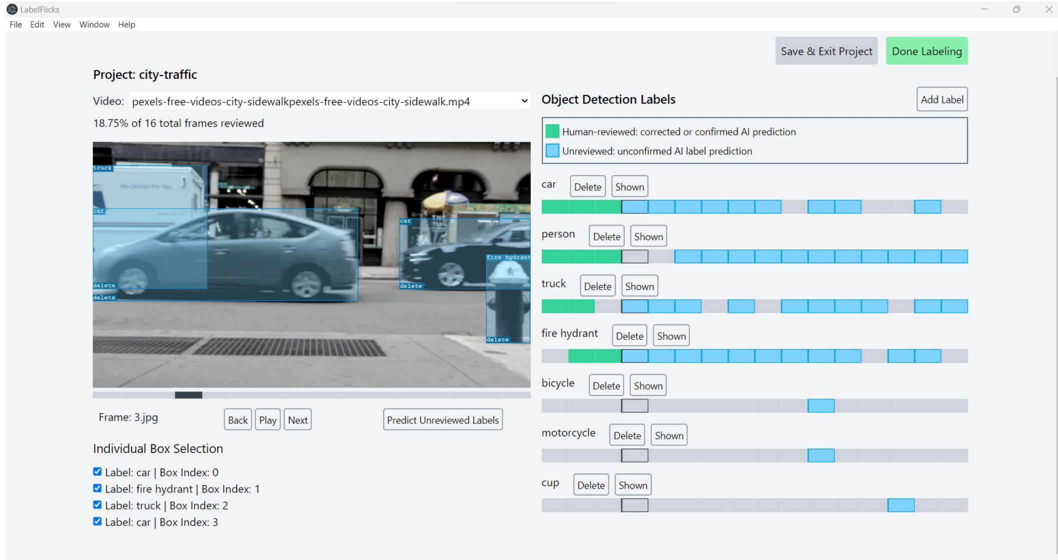

Figure 4.12: Labeling screen with some initial frames marked as human-reviewed

The labeling timeline segments also serve as a way for the user to navigate to different frames in the video. The user can simply click on any segment and the frames player will update to show the correct image and overlaid bounding boxes. Figure 4.13 shows how the selected segment is outlined after clicking ahead to a later frame.

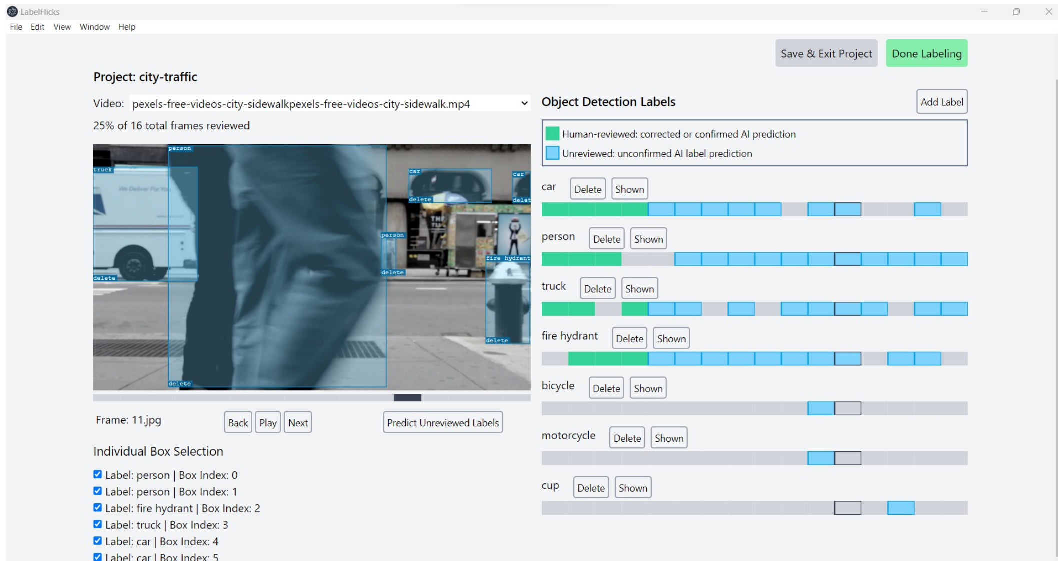

Figure 4.13: Jumped ahead to a later frame using a labeling timeline

Above each timeline are two buttons: the delete and hidden/shown buttons. As seen in Figure 4.14, clicking the “shown” button will remove the bounding box overlays associated with the label from the frames player and change the button to say “hidden” instead. Clicking the “hidden” button will toggle the visibility of the label again and display the bounding boxes. Clicking the “delete” button will remove the label from the project, and any boxes that had that label would be reassigned to the most commonly used label in the project instead. This is a heuristic that could be improved later, which is mentioned in the Future Work section.

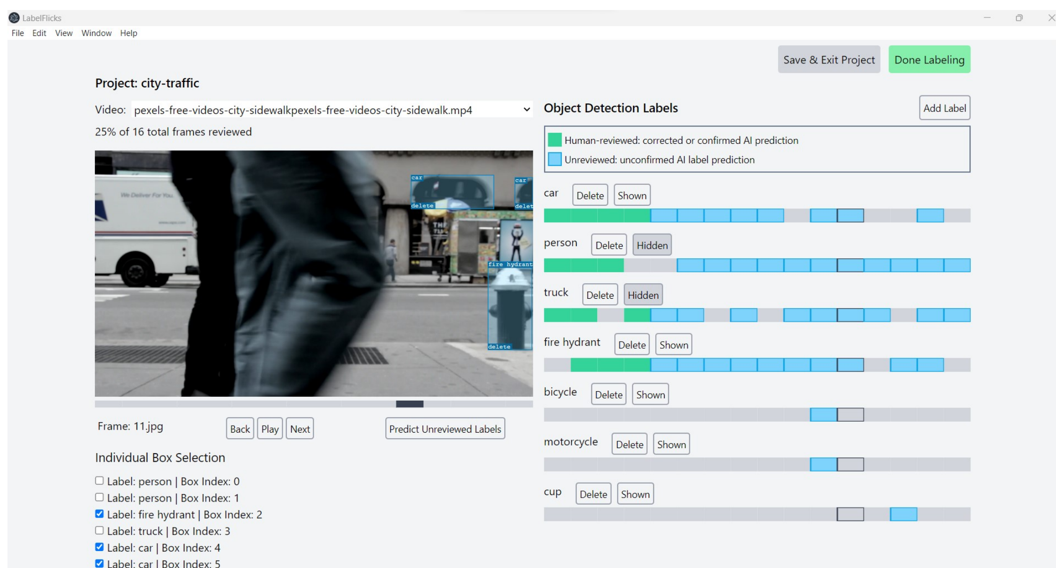

Figure 4.14: Labeling screen with Labeling Timelines marked as “hidden” to remove the corresponding overlaid boxes

Above the labeling timelines is the “Add Label” button. Clicking it will bring up the modal for creating a new label, as shown in Figure 4.15. The new timeline will initially be empty and contain no blue or green segments, but it can be iteratively populated by AI predictions and user corrections.

In order to correct the label on a bounding box, the user must click on the name of a box, which will bring up a dropdown menu of all the labels currently used in the project, as shown in Figure 4.16. When a new label has been selected, the bounding box will turn green, as shown in Figure 4.17. At this point, the user may want to submit their edits to the

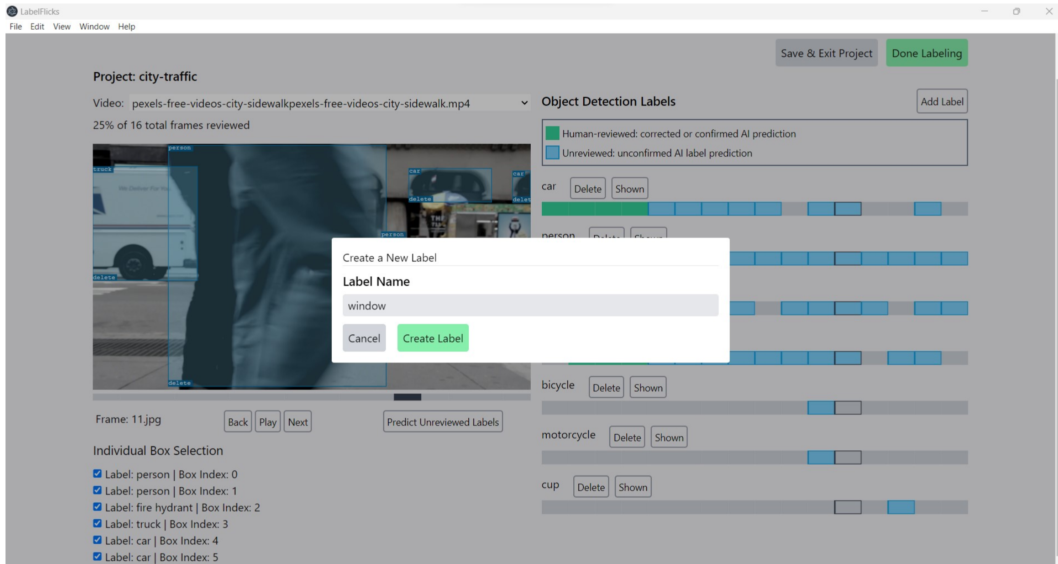

Figure 4.15: Labeling screen with modal for creating a label

LabelFlicks AI assistant to tune the predicted labels on unreviewed bounding boxes in later frames, so they may click the “Predict Unreviewed Labels” button near the bottom right of the frames player. Figure 4.18 shows the informational tooltip that appears when the user hovers over this button.

The class diagram for the labeling screen is provided in Figure 4.19 and it shows the key dependencies required to render the page.

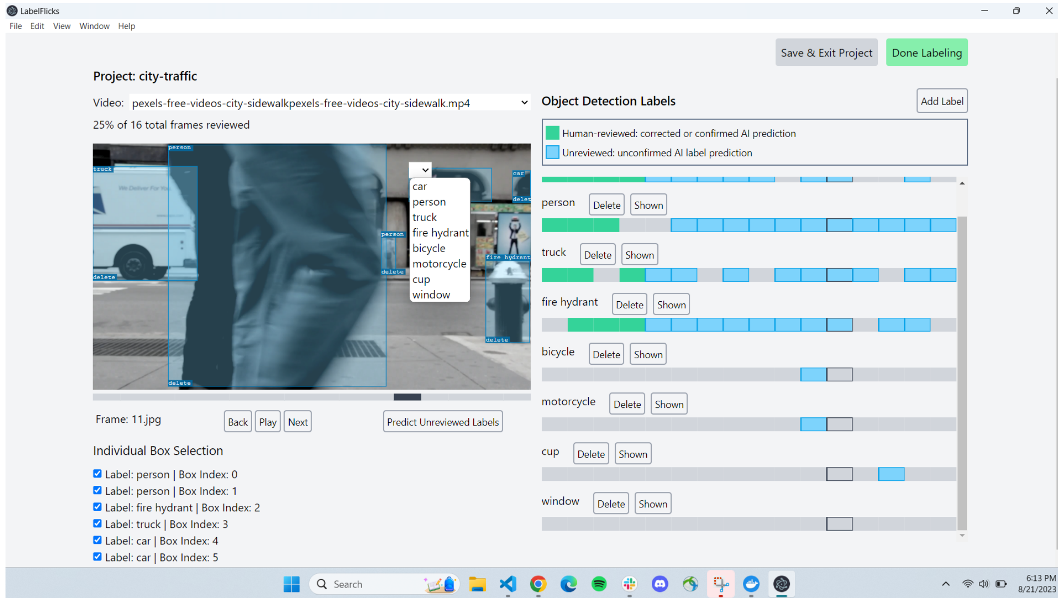

Figure 4.16: Labeling screen showing dropdown for selecting the correct label

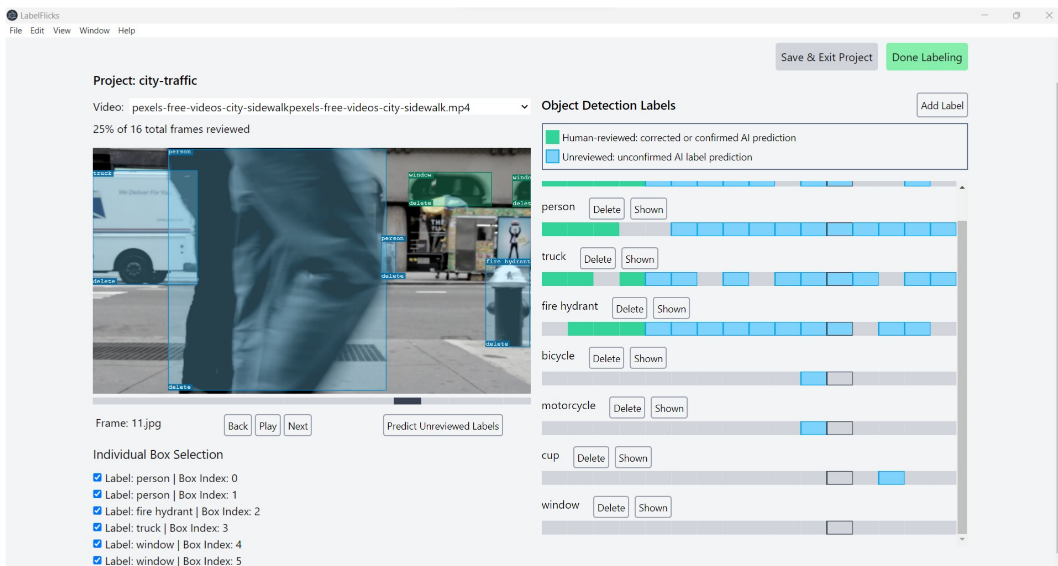

Figure 4.17: Labeling screen with a couple of corrected labels

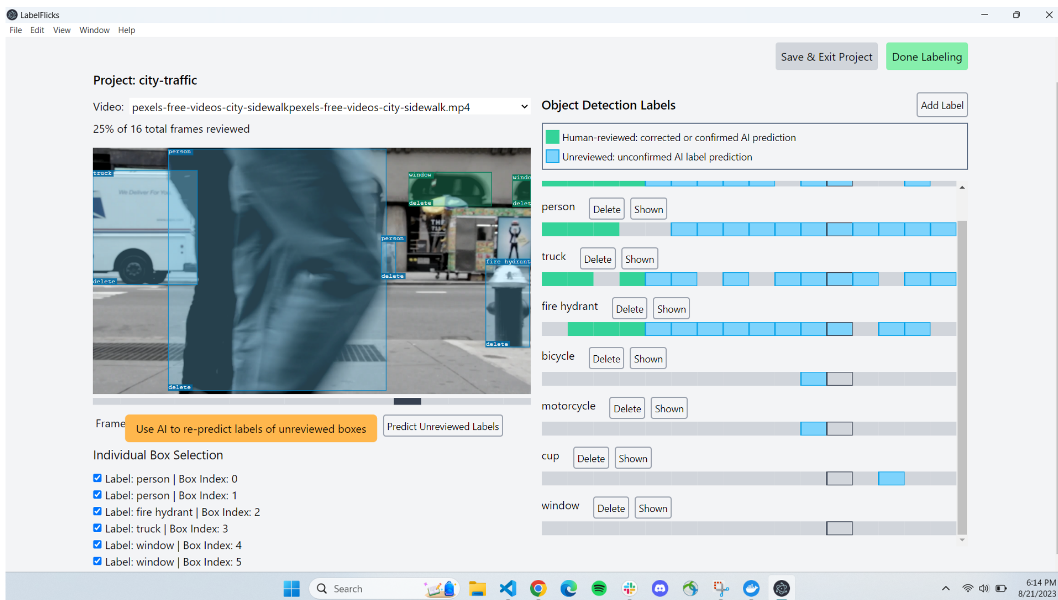

Figure 4.18: Labeling screen showing tooltip explaining the “Predict Unreviewed Labels” button

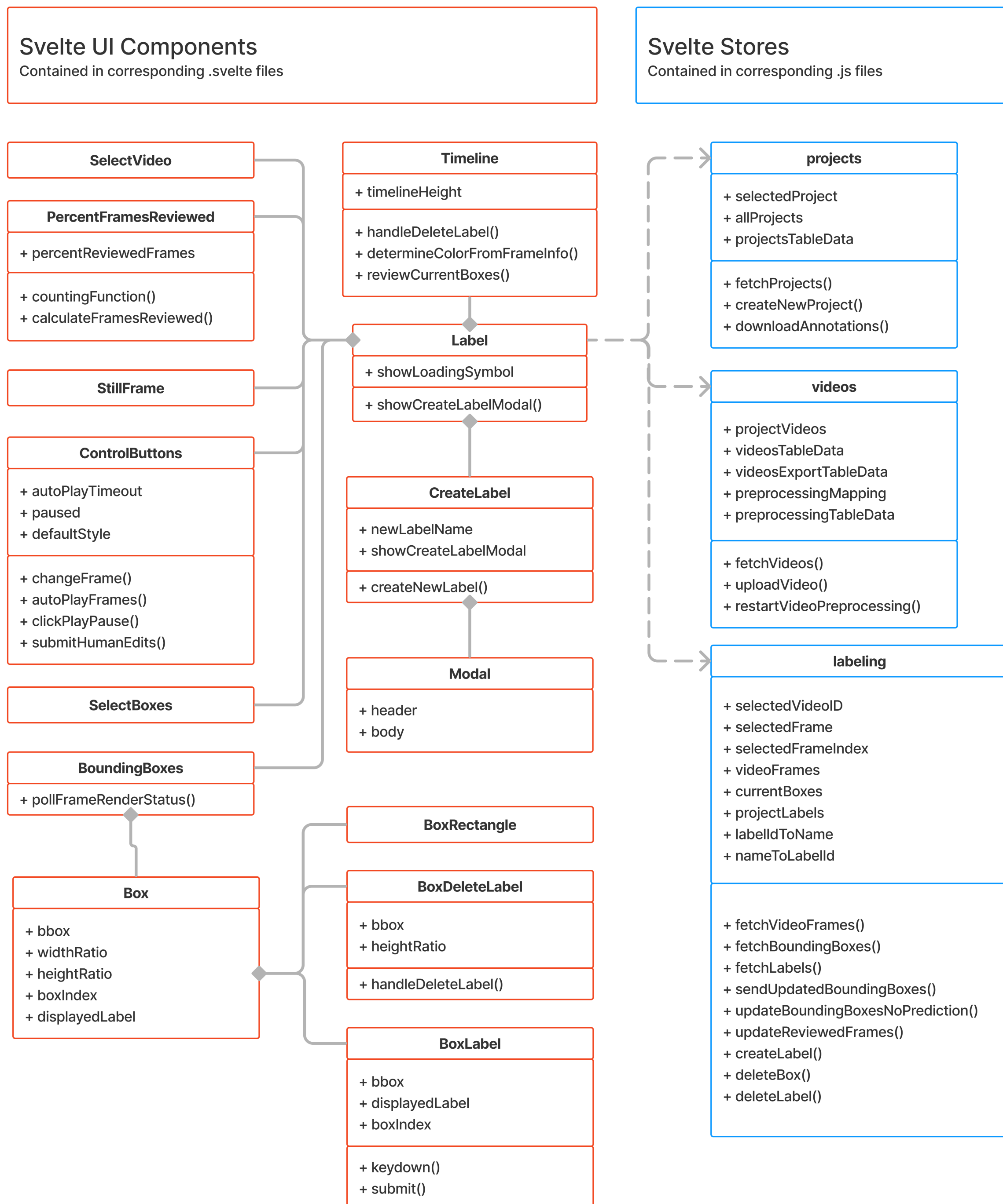

Figure 4.19: Labeling screen class diagram, which makes use of Svelte components and stores.

4.1.5 Export Labels

The final screen of the LabelFlicks desktop application, as shown in Figure 4.20, allows users to save their annotations to their local file system. A user can click the “Download annotations” button to extract all annotation information from the backend’s database and when that process is complete, the application will provide an alert containing the path to where the annotations are saved. Currently, LabelFlicks saves the bounding boxes from all videos in the project in text files that are named using each frame’s database UUID. The boxes are saved in YOLO format (one of the common annotation formats) where each line describes one box: “labelUUID center_x center_y width height”. Once the user has copied the path from the alert, they may click the “Exit Project” button and return to the home screen.

The class diagram for the export screen is provided in Figure 4.21 and it shows the key dependencies required to render the page.

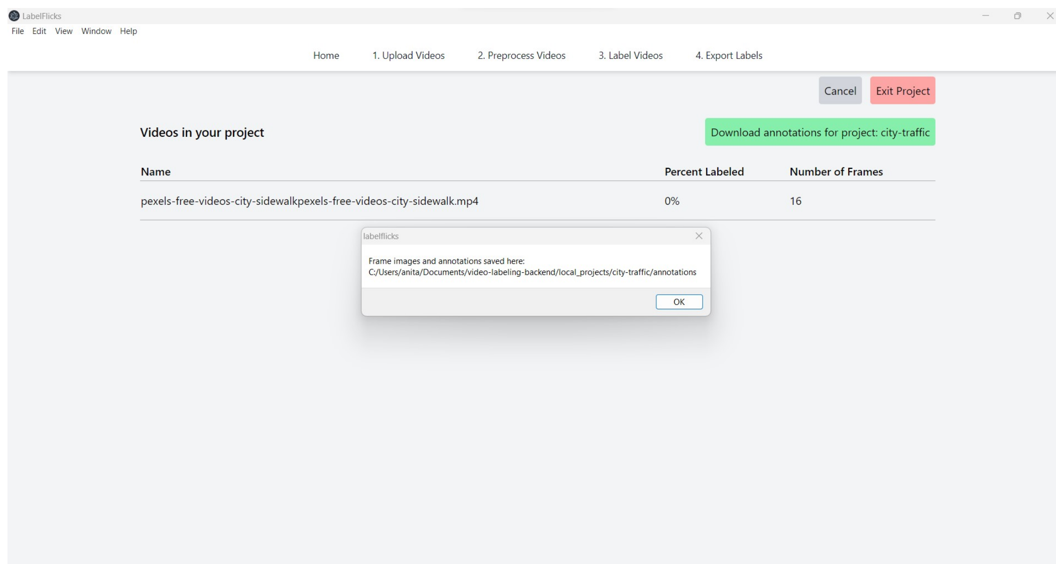

Figure 4.20: Export Labels screen with alert stating where to find the annotations

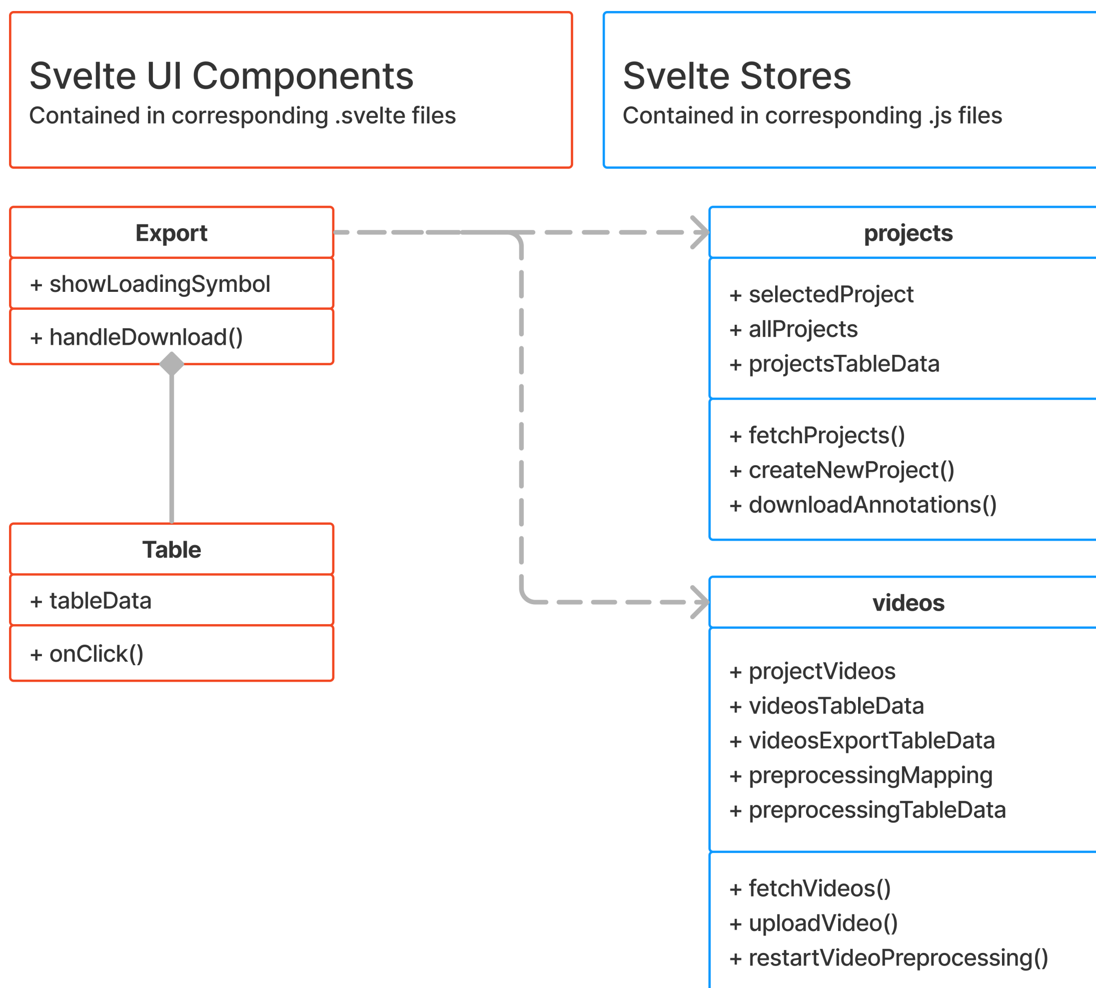

Figure 4.21: Export labels screen class diagram, which makes use of Svelte components and stores.

4.2 PostgreSQL Database

Currently, LabelFlicks stores all videos and frame images on the local file system and all other data in the PostgreSQL database. The database is run inside of a Docker container and uses the official postgres Docker image. The database schema and all interactions with the database from the FastAPI server are defined using SQLAlchemy. Figure 4.22 shows the database's Entity-Relationship Diagram (ERD) using crow's foot notation to denote cardinality.

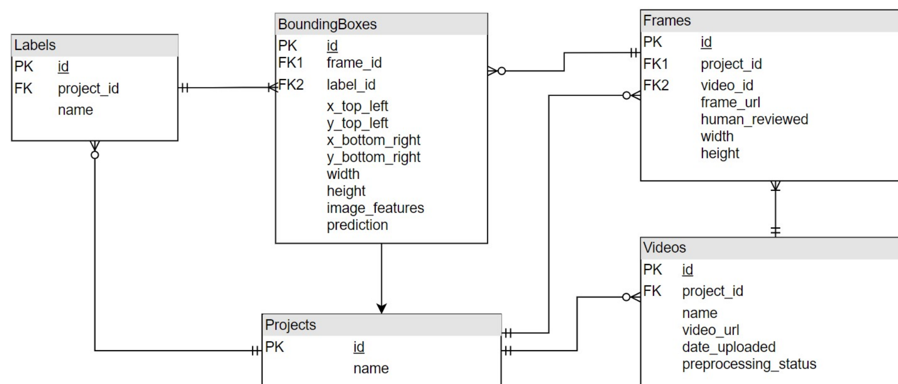

Figure 4.22: LabelFlicks database diagram that shows the relationships between the Projects, Videos, Frames, BoundingBoxes, and Labels tables.

4.3 FastAPI Server

The LabelFlicks REST API is served by a FastAPI server and was developed using Python. The class diagram is shown in Figure 4.23. The main monolith server is outlined in yellow, the storage-related components are outlined in green, and the ML-related components are outlined in purple.

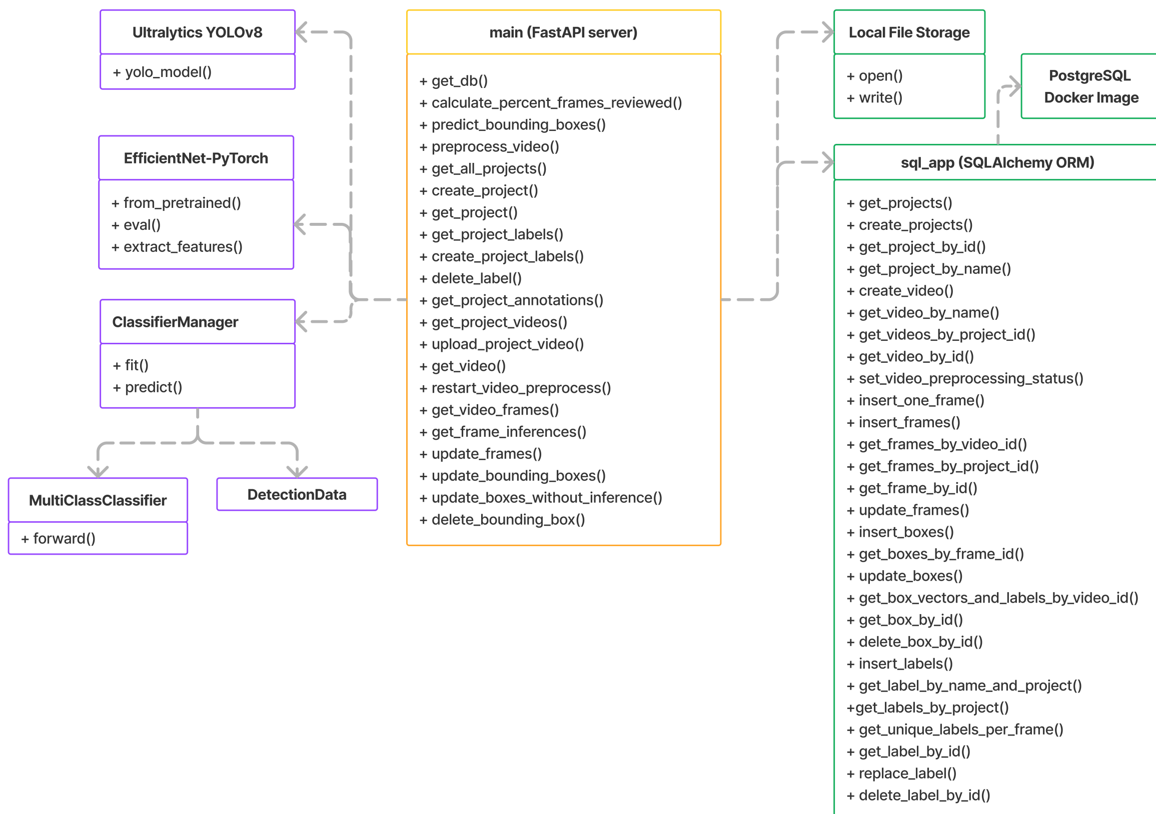

Figure 4.23: Class diagram of the main backend components

All currently implemented endpoints are featured in Figure 4.24. Each endpoint will be explained as part of a sequence diagram illustrating the "happy path" for how they are used in the desktop UI.

GET /projects Get All Projects

POST /projects Create Project

GET /projects/{project_id} Get Project

GET /projects/{project_id}/labels Get Project Labels

POST /projects/{project_id}/labels Create Project Labels

DELETE /projects/{project_id}/labels/{label_id} Delete Label

GET /projects/{project_id}/annotations Get Project Annotations

GET /projects/{project_id}/videos Get Project Videos

POST /projects/{project_id}/videos Upload Project Video

GET /videos/{video_id} Get Video

GET /videos/{video_id}/preprocess Restart Video Preprocess

GET /videos/{video_id}/frames Get Video Frames

GET /frames/{frame_id}/inferences Get Frame Inferences

PUT /frames Update Frames

PUT /boundingboxes Update Boxes Without Inference

POST /boundingboxes Update Bounding Boxes

DELETE /boundingboxes/{box_id} Delete Bounding Box

Figure 4.24: Screenshot of all currently implemented FastAPI endpoints

4.3.1 Home Screen

As shown in Figure 4.25, the desktop UI must call the GET `projects` endpoint in order to fetch projects from the database. When the user clicks the "create project" button, the UI will call the POST `projects` endpoint and insert the new project information into the database and return the new project's information to the UI.

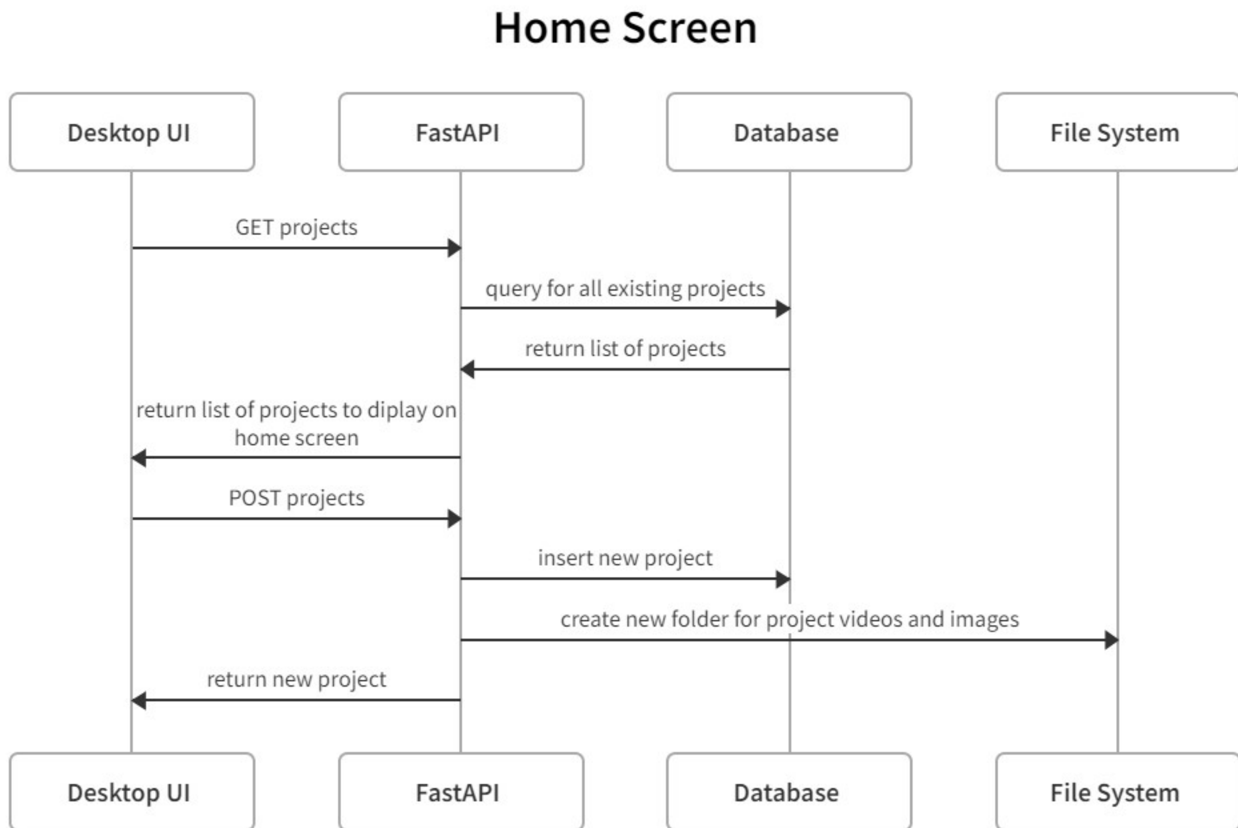

Figure 4.25: Sequence diagram of API calls used for the home screen

4.3.2 Video Upload Screen

As shown in Figure 4.26, the video upload screen will send a GET request to `projects/projectId` and `projects/projectId/videos` in order to get the necessary information about the project and its videos. When the user clicks the "upload video" button, the UI will make a POST request to `projects/projectId/videos` so the new video can be inserted into the database. The MP4 video file will be saved to the local file system used by the backend and the FastAPI server will also start a background task for preprocessing the video.

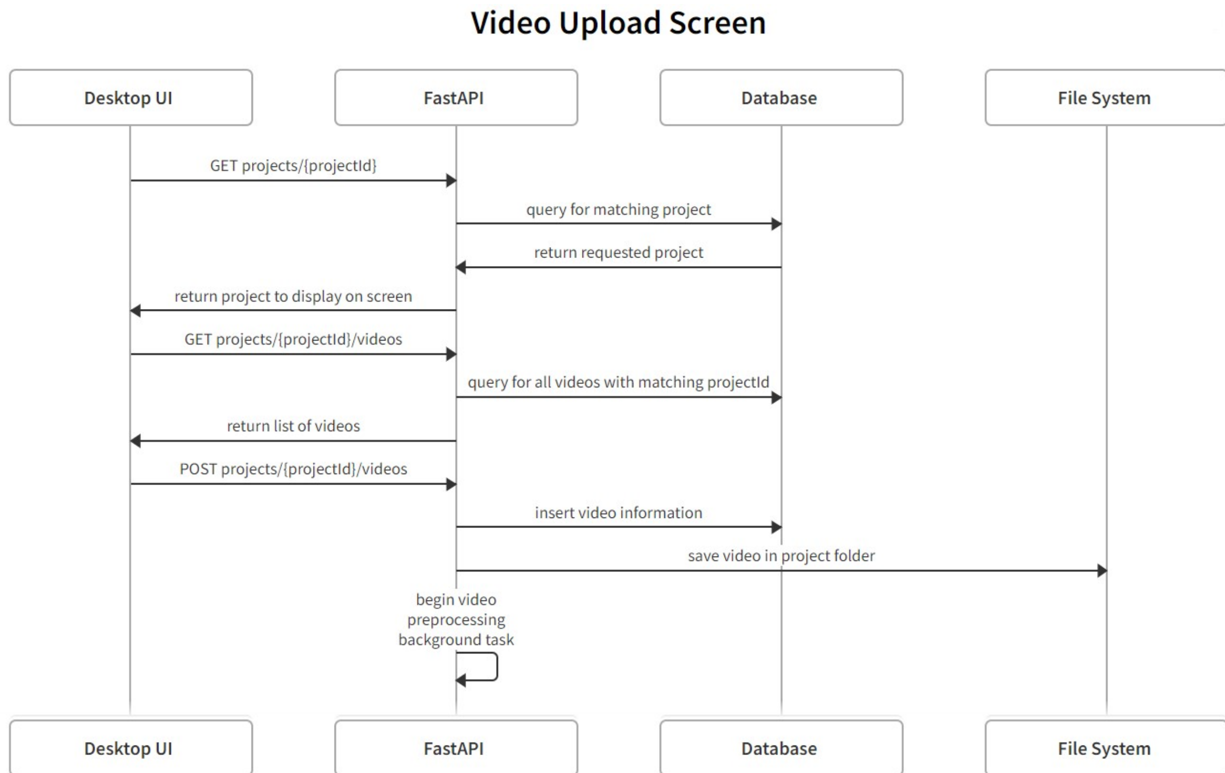

Figure 4.26: Sequence diagram of API calls used for the video upload screen

4.3.3 Preprocessing Screen

The preprocessing sequence diagram, shown in Figure 4.27, is a continuation of the video upload diagram. Once the preprocessing background task starts, FastAPI sets the preprocessing status for that video to "in_progress" in the database. The server then uses OpenCV to chop up the video into a series of frames, basically taking a snapshot of the video every second and saving them as images on the local file system. Information about each frame is inserted into the database, then the Ultralytics YOLOv8 object detection model is applied to the frame image to generate an initial set of labeled bounding boxes. YOLOv8 is one of the latest revisions of the "You Only Look Once" (YOLO) real-time object detection model [7] and it was chosen for its ability to quickly locate and identify objects compared to other pre-trained deep learning models. Information about each label and bounding box is inserted into the database. The EfficientNet feature extraction model is applied to the portions of the frame image that each bounding box corresponds to in order to get an image feature vector for each box, which is saved as a binary data blob in the database along with each bounding box. Once this process is complete for all frames and bounding boxes of the video, FastAPI sets the preprocessing status to "success".

While the backend server chugs through the preprocessing steps, the desktop UI employs long polling to periodically check the preprocessing status for each uploaded video. The UI uses a GET request to `projects/projectId/videos` and checks the preprocessing status of each returned video in order to update the preprocessing screen as needed.

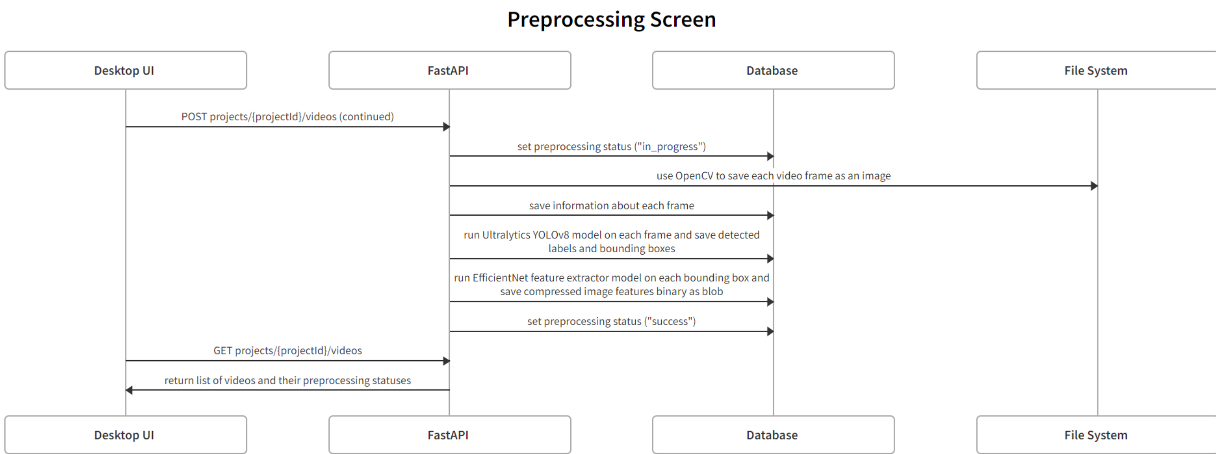

Figure 4.27: Sequence diagram of API calls used for the preprocessing screen

4.3.4 Labeling Screen

Figure 4.28 shows the API calls used when the UI initializes the labeling screen, either after navigating to the screen for the first time or after selecting a different video from the project videos dropdown menu. The UI makes a GET request to `projects/projectId/labels` to fetch all project labels from the database and render them as labeling timelines on screen. The UI makes an additional two API calls–GET `projects/projectId/frames` and GET `projects/projectId/inferences`–to get the video frames for this project and the bounding boxes for the currently displayed frame. The frame image is displayed in the frames player and the bounding boxes are SVGs overlaid on top.

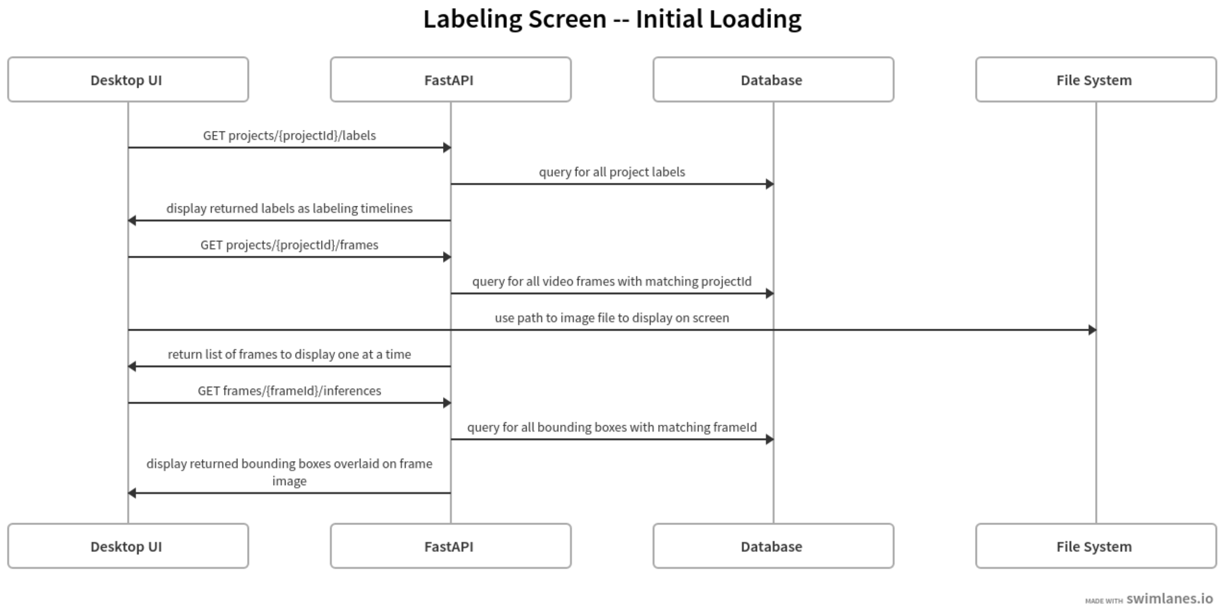

Figure 4.28: Sequence diagram of API calls used for populating the labeling screen

The top half of Figure 4.29 shows how after the user clicks on a segment in a labeling timeline, the UI will update the frame player by finding the next frame image on the local file system and make a GET request to `frames/frameId/inferences` to fetch the bounding boxes for the next frame. This sequence flow differs from the bottom half of Figure 4.29; clicking the “play”, “next”, or “back” button rather than a timeline segment will make two additional requests: PUT `boundingboxes` and PUT `frames`. These two requests work to mark the pre-

vious frame and its bounding boxes as "human-reviewed". This way, the colors of the labeling timeline segments and bounding box are updated the next time the UI refreshes the screen using the GET `projects/projectId/labels` and GET `frames/frameId/inferences`.

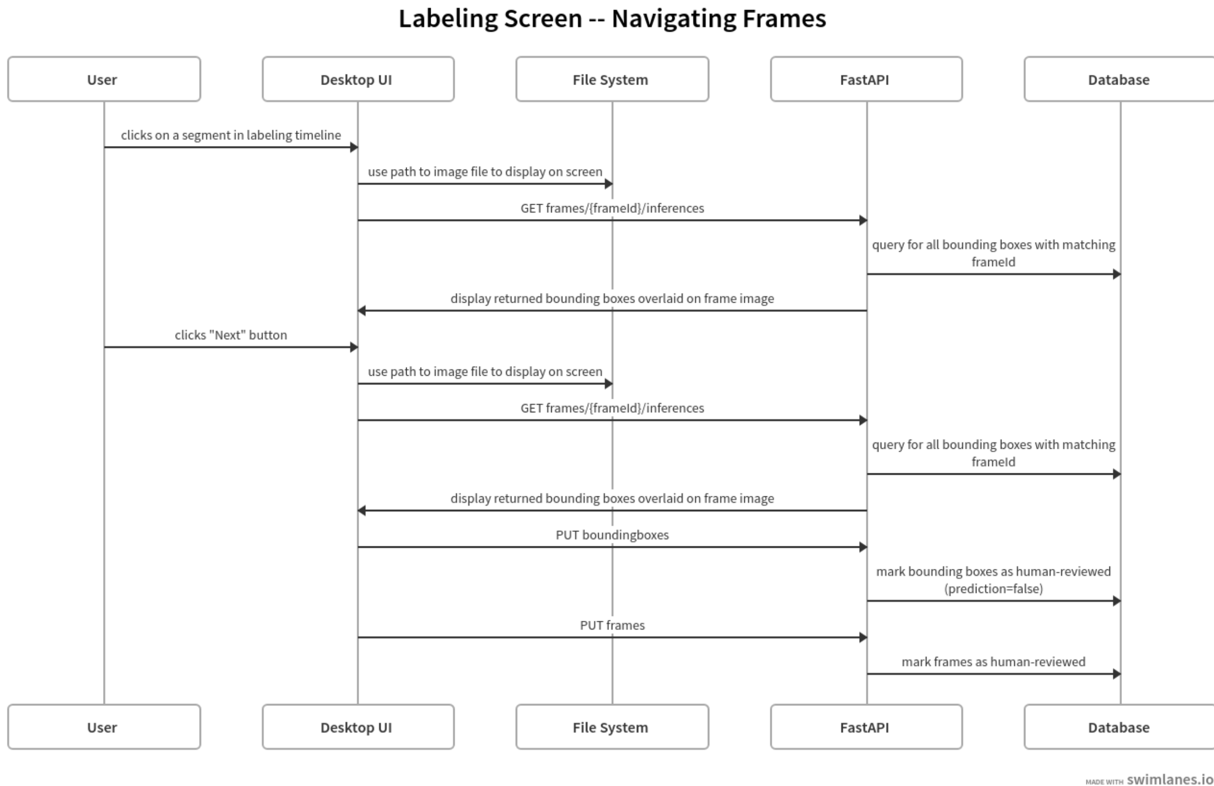

Figure 4.29: Sequence diagram for API calls when navigating between frames

Figure 4.30 shows the sequence of actions after the user clicks on the "Predict Unreviewed Frames" button. First, any edited bounding boxes from the current frame is sent in a POST request to `boundingboxes?project_id=projectId&video_id=videoId`. The current frame and its bounding boxes are marked as human-reviewed, then the project's labels and remaining bounding boxes are fetched from the database. FastAPI creates an instance of the MultiClassClassifier, written in PyTorch, and trains it using all of the human-reviewed boxes from the project. Currently, the MultiClassClassifier is a two-layer neural network with a ReLU activation and is hardcoded to train for five epochs (this is a hyperparameter that could be tuned in future improvements assuming the model is not replaced by a better alternative).

Once training completes, the model predicts the labels for all non-human-reviewed bounding boxes in the project and returns a simple 200 response to the UI to let it know it can refresh the screen.

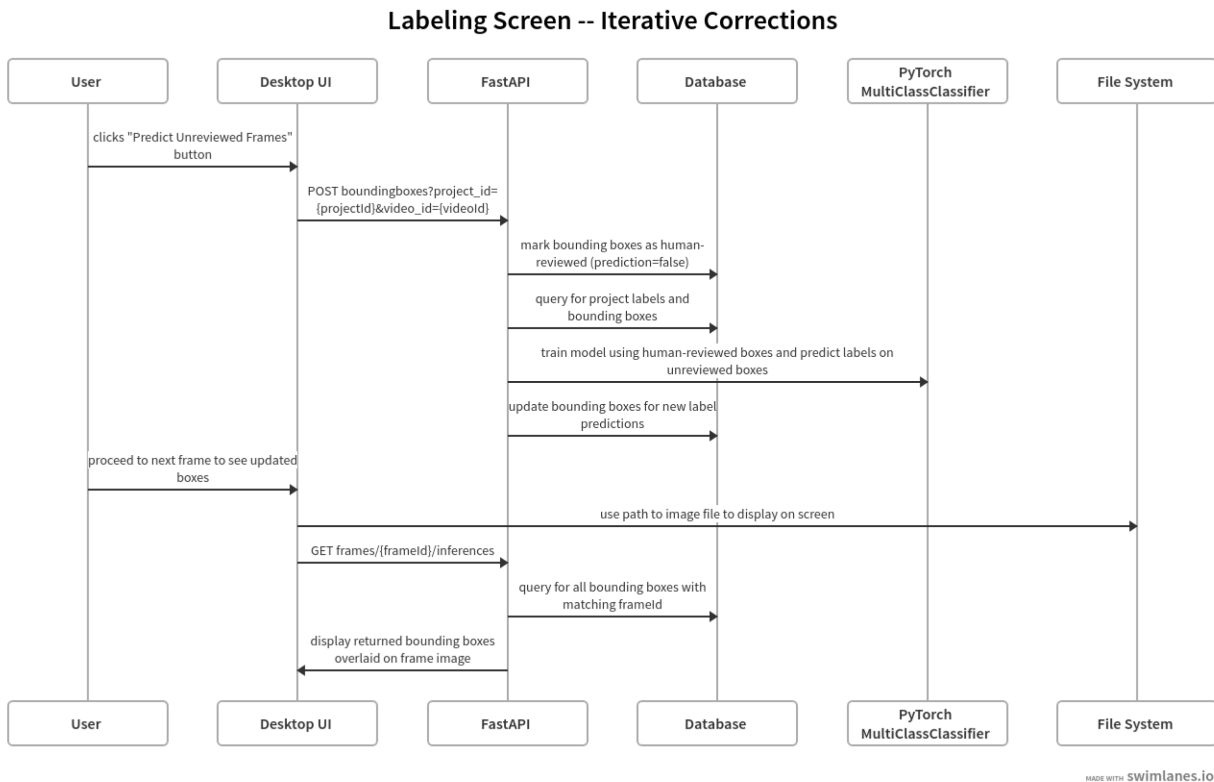

Figure 4.30: Sequence diagram for API calls when iteratively correcting boxes using AI assistance

The final sequence diagram for the labeling screen, shown in Figure 4.31, illustrates the sequence flows for deleting a bounding box, creating a new label, and deleting a label. When a user clicks on the "delete" tag in the bottom left corner of a bounding box, the UI makes a DELETE request to `boundingboxes/boxId`, which will delete the box with the specified ID from the database. When the user clicks the "add label" button, the UI makes a POST request to `projects/projectId/labels` so that the new label can be inserted into the database. Finally, when the user makes a DELETE request to `projects/projectId/labels/labelId`, the matching label will be removed from the database and all bounding boxes that used to have that label will be assigned the most

commonly used label in the project. This is a heuristic that could be improved later, as mentioned in the Future Work section.

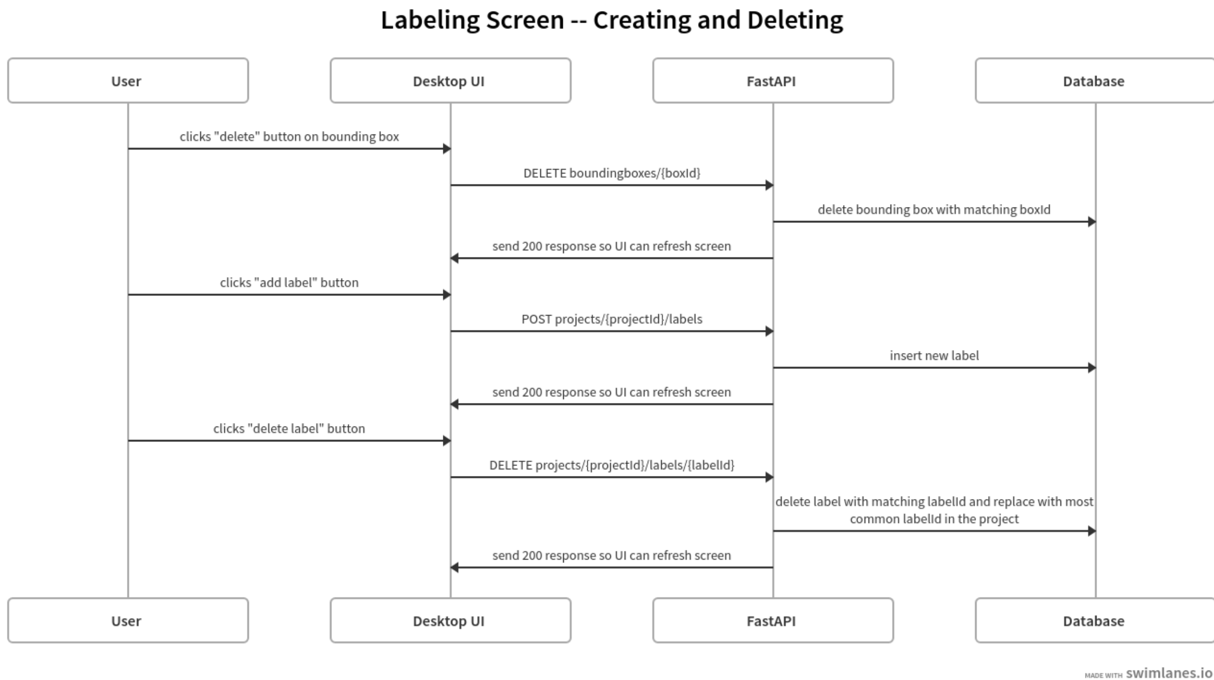

Figure 4.31: Sequence diagram for API calls when deleting boxes, creating labels, or deleting labels

4.3.5 Export Labels Screen

As shown in Figure 4.32, when the user clicks on the “download annotations” button, a GET request is made to `projects/projectId/annotations`. FastAPI queries the database for all relevant information about the project annotations, such as the project labels, frames, and bounding boxes. All labels are written out to a text file on the local file system to the default save location. A text file is created for each frame of the video and each line in the file describes one box in the frame using the YOLO format: “labelUUID center_x center_y width height”. FastAPI returns the default save location to the UI to be displayed to the user.

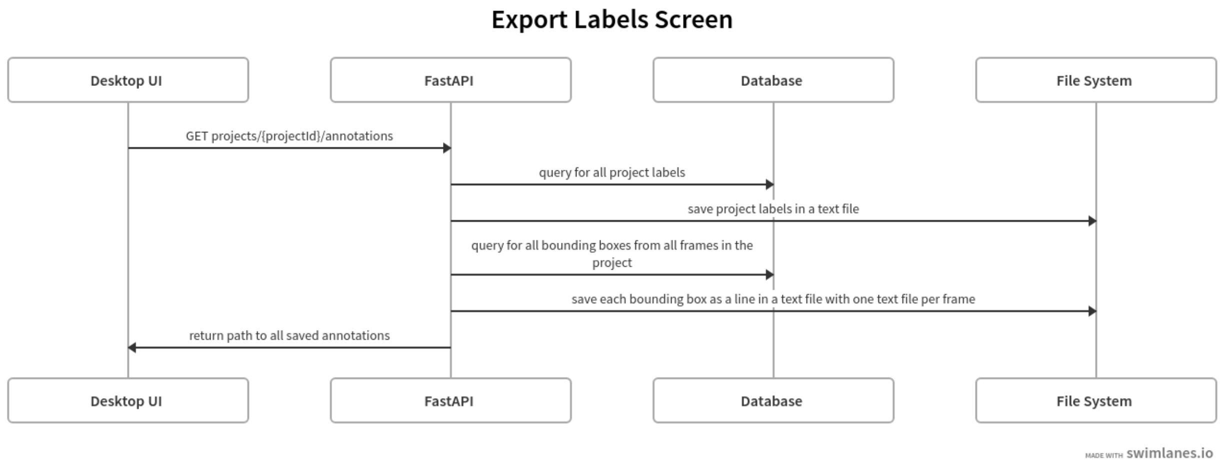

Figure 4.32: Sequence diagram of API calls used for the export labels screen

4.4 Scope and Current Limitations

In its current state as a PoC, the main limitation of LabelFlicks is its ability to be useful to real users. The design choice to use the YOLOv8 model as the pre-labeling object detection model means that LabelFlicks will only really be able to pre-label objects in videos as long as they belong to one of the COCO dataset's 80 classes as that is what the model was pre-trained on. Of course, the world consists of more than 80 objects, but currently there are no features available to allow the user to create more bounding boxes or train the small classifier to detect future instances of new object categories. People hoping to create an entirely new custom dataset from scratch with no object classes found in the COCO dataset [8] [9] would likely not be able to do so.

Additional technical limitations of LabelFlicks include the following:

- Video file format: LabelFlicks currently supports only MP4 videos as it is one of the most commonly used video file formats.
- Editing bounding boxes: LabelFlicks currently assumes that the Ultralytics YOLOv8 object detection model is powerful enough to detect all important objects in each frame so it does not provide annotation tools for moving, resizing, or creating new bounding boxes.
- Small classifier limitations: LabelFlicks currently assumes the main purpose of the human-in-the-loop workflow is for correcting the labels on bounding boxes. As such, the small classifiers trained as part of this workflow are unable to learn what it means when a bounding box is deleted and they are unable to predict any new bounding boxes for objects that YOLOv8 might have missed. Additionally, the small classifiers are trained only on reviewed boxes for one video at a time, meaning the learning is not transferred to other videos in the same project.
- Data storage: Local file storage is heavily used in the current implementation of LabelFlicks as it was the most readily available form of storage during initial development.

Initial designs considered connecting LabelFlicks to a cloud storage provider, but that feature was ultimately deemed out of scope for an initial PoC.

All the above limitations are included in the considerations for Future Work.

5. Future Work

As mentioned in the previous section, LabelFlicks has limited real-world usefulness, but there are many possible future directions it can take with further development. The following subsections will provide a non-exhaustive overview of these possible use cases and corresponding future work.

5.1 Custom Dataset Creation

If LabelFlicks were to continue improving as an open-source data labeling tool, the main PoC limitations must be addressed in addition to building out new features to better support annotators.

- Customizable pre-labeling detection model: To better support the creation of custom datasets with non-COCO objects, LabelFlicks could offer users a way to select or provide their own pre-trained detection model to apply during the preprocessing step. This model could be pre-trained on a dataset with objects that are more similar to what the user may be trying to label and the user would not have to manually create bounding boxes for those objects.
- More capable human-in-the-loop model: The small label classifier currently used in LabelFlicks could be swapped out for an object detection model–either the same as the pre-labeling detection model or different–that would be able to predict both new labels and bounding boxes in future frames. A more capable assistive model in the human-in-the-loop workflow can help to fill in the gaps left by the pre-trained model used in the preprocessing step. This model could also go a step beyond the current LabelFlicks classifier by retraining on all human-reviewed annotations in the selected project rather than just for the selected video, thereby allowing the assistive model to be helpful across all videos in the project rather than just one video at a time.

However, care must be taken to ensure this new model would not overly increase the latency of each human-in-the-loop iteration.

- Add and resize bounding boxes: LabelFlicks could allow users to add new bounding boxes and resize existing bounding boxes. The human-in-the-loop model could then learn from the user's actions (e.g. deleting a box means it was unnecessary, resizing a box means it was incorrect, adding a box means it was overlooked) and provide better predictions in later frames.

- Handle deleted labels more intelligently: When a user deletes a label, LabelFlicks currently assigns the boxes that used to have that label to the most commonly used label in the project. This is a heuristic that could instead be replaced with new predictions from the human-in-the-loop model.

- Support active and/or batch labeling: LabelFlicks could implement a form of active learning and/or batch labeling [10], allowing the human-in-the-loop model to have a greater role in asking the user to label specific examples that it finds difficult to automatically classify. Allowing users to label frames in batches could also help with maintaining label consistency (i.e. the concepts that users are labeling for should not drift after they see new frames).

- Take advantage of the temporal aspect of videos: LabelFlicks could better exploit the temporal property of videos (i.e. adjacent frames are similar to each other) by using algorithms that track objects across multiple frames. This means that LabelFlicks could provide labeling features in terms of a series of frames rather than individual frames. For example, the user could manually correct the boxes in the first frame of a series of frames, and then the correction would carry over through the rest of the series.

- Support more file formats: LabelFlicks could support all video file formats, not just MP4. LabelFlicks could potentially even support image collections for videos that had already been converted into a series of frames.

- Provide cloud storage options: This feature would serve users who may not have enough local storage space for all of their labeling projects. This would also serve users who

may have data that already exists in some form of cloud storage such as Amazon Web Services, Google Cloud Platform, or Microsoft Azure. If this feature is provided, LabelFlicks must take care to also fully decouple the frontend from the local file system; it currently uses the local file system paths to a video's image files to render the frames player but ideally it would have no dependency on the file system.

- Eliminate the video to frames conversion step: Several of the enterprise data labeling tools appear to bypass the need to convert videos into frames and instead support data labeling while the video plays at its native frame rate. LabelFlicks could potentially mimic this feature and significantly reduce the preprocessing wait time.

- Support polygon masks: LabelFlicks could provide an annotation tool for drawing polygon masks rather than only rectangular boxes around each object. This could allow LabelFlicks to data labeling for other computer vision tasks, such as segmentation.

- Collaborative features: LabelFlicks could offer a collaborative labeling feature as some enterprise products do, allowing small teams of people to work together and complete the labeling process sooner. Given the chosen tech stack, it would not be difficult to convert LabelFlicks into a web application that could be run in different browsers, allowing teammates to more quickly complete labeling projects. Additionally, implementing some centralized online component could also allow LabelFlicks to calculate the inter-rater agreement between teammates so that the final labeled dataset can be more consistent and accurate.

- Conform to standard user experience design heuristics: LabelFlicks could fill in the gaps outlined by core UI/UX principles, such as Nielsen's Heuristics [11]. For example, it could better support user control by providing undo and redo buttons and it could make the status of the AI assistance more visible to the user.

- Make the UI more inclusive and accessible: LabelFlicks could better serve its users by correcting inclusivity bugs identified through the GenderMag process [12] and other such cognitive walkthroughs.

- Conduct user studies to evaluate impact: User studies could be performed using LabelFlicks to evaluate whether it is effective at increasing labeling accuracy, reducing

labeling fatigue, and reducing the amount of time and cognitive effort a user spends preparing their data.

- Design more human-centered interactions: LabelFlicks could be redesigned in some aspects to incorporate ideas from the new research field called human-centered AI (HCAI) [13]. Rather than subjecting the human annotator to the demands of the human-in-the-loop workflow, LabelFlicks could instead provide interactions to allow the user to drive the labeling process instead.

5.2 Use IML to Train While Labeling

Rather than creating a human-in-the-loop workflow for generating a dataset, LabelFlicks could instead provide a way for users to train a computer vision model using the interactive machine learning (IML) paradigm. Some design considerations for pursuing this direction of development are listed below:

- Allow finetuning or training from scratch: LabelFlicks could offer users a way to fine-tune a pre-trained model on a more specific dataset or to train a new object detection model from scratch. This could be presented as an additional screen in the overall workflow.

- UI adjustments: The LabelFlicks desktop application would have to rearrange and incorporate some new features to better support an IML workflow. The labeling screen would likely draw more attention to the model's performance (e.g. accuracy, recall, precision) and the export screen would have to provide a way to export or save the trained model in addition to the annotations if the user wants them.

- Interactive debugging: LabelFlicks could use explanatory debugging [14] or other explainable AI methods to help users build a mental model of how the ML model reasoned its way to incorrect predictions in order to make more effective "bug fixes" to the model.

5.3 Testing Computer Vision Models

LabelFlicks could pivot to being an inspection or testing application for computer vision models. ML models are powerful and produce impressive results in many applications today, but they are still generally viewed as "black boxes" since many developers have no idea of the exact internal workings. LabelFlicks could be made into a human-centered application for iteratively inspecting a model's performance on a test set. Some design considerations for possible future work are listed below:

- Select a model and a dataset to inspect: LabelFlicks could offer a way for users to select or upload a computer vision model they want to inspect along with some videos they want to use the model on. The preprocessing step could still be very useful in decomposing the videos into frames or a series of frames and automatically applying the model to each unit.

- Change labeling screen to inspection screen: Instead of a labeling screen, the desktop application would probably provide an inspection screen instead. This application could allow users to iteratively mark model predictions as correct or incorrect and continually update the model's performance metrics using a human-in-the-loop workflow. This way, the test data can be entirely fresh, unseen data that the model has not seen before and that the user does not have to worry about labeling beforehand.

- Inspect performance on partitions of the test dataset: The inspection screen could provide a way for the user to inspect model performance on subsets of the test data. Subgroup analysis using visual analytics can help users to identify biases and other fairness issues in the model [15].

- Closer inspection of confidence scores: DL models are known to be overly confident in their predictions, which means they can very confidently make an incorrect prediction, which can result in disastrous outcomes in certain domains (e.g. self-driving vehicles). The testing application could implement a way to measure miscalibration of computer vision models (such as the method proposed by Kuppers et al. [16]) and compare that

with the model's confidence scores which can be visualized using color or some other mechanism on the UI.

- Post-inspection reflection: Instead of an export labels screen, the final screen of the desktop application could report the calculated performance metrics and provide the user an opportunity to complete a post-inspection reflection in the spirit of After-Action Review for AI (AAR/AI) [17], which can help the user synthesize and make sense of the information they learned during their inspection and help them identify bugs in their model.

6. Conclusion

This report provided an overview of LabelFlicks, an open-source, proof-of-concept labeling tool for streamlining videos into fully labeled object detection datasets. The field of ML has garnered a lot of attention and excitement in recent years, but data labeling presents a thorn in the side of many developers who want to create their own custom ML models. LabelFlicks was created to address the data labeling bottleneck, but it has limited usefulness in its current form. However, with further development, LabelFlicks has the potential to become a handy open-source tool that offers an ML-assisted workflow for labeling videos, an application for iteratively training and debugging computer vision models, or a tool for inspecting and testing a model's performance on unseen test data.

References

[1] "Object detection." https://www.tensorflow.org/hub/tutorials/object_detection.

[2] G. Huang, I. Laradji, D. Vázquez, S. Lacoste-Julien, and P. Rodríguez, "A survey of self-supervised and few-shot object detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 4, pp. 4071–4089, 2023.

[3] D. Kaur, S. Uslu, K. J. Rittichier, and A. Durresi, "Trustworthy artificial intelligence: A review," *ACM Comput. Surv.*, vol. 55, jan 2022.

[4] "Humans in the loop: The design of interactive ai systems." https://hai.stanford.edu/news/humans-loop-design-interactive-ai-systems.

[5] J. J. Dudley and P. O. Kristensson, "A review of user interface design for interactive machine learning," *ACM Transactions on Interactive Intelligent Systems (TiiS)*, vol. 8, no. 2, pp. 1–37, 2018.

[6] M. Desmond, M. Muller, Z. Ashktorab, C. Dugan, E. Duesterwald, K. Brimijoin, C. Finegan-Dollak, M. Brachman, A. Sharma, N. N. Joshi, and Q. Pan, "Increasing the speed and accuracy of data labeling through an ai assisted interface," in *26th International Conference on Intelligent User Interfaces*, IUI '21, (New York, NY, USA), p. 392–401, Association for Computing Machinery, 2021.

[7] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788, 2016.

[8] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*, pp. 740–755, Springer, 2014.

[9] "Coco dataset." https://docs.ultralytics.com/datasets/detect/coco/.

[10] C. Chung, J. Lee, K. Park, J. Lee, M. Kim, M. Song, Y. Kim, J. Choo, and S. R. Hong, "Understanding human-side impact of sampling image batches in subjective attribute labeling," *Proceedings of the ACM on Human-Computer Interaction*, vol. 5, no. CSCW2, pp. 1–26, 2021.

[11] "10 usability heuristics for user interface design." https://www.nngroup.com/articles/ten-usability-heuristics.

[12] M. Burnett, S. Stumpf, J. Macbeth, S. Makri, L. Beckwith, I. Kwan, A. Peters, and W. Jernigan, "Gendermag: A method for evaluating software's gender inclusiveness," *Interacting with Computers*, vol. 28, no. 6, pp. 760–787, 2016.

[13] B. Shneiderman, *Human-centered AI.* Oxford University Press, 2022.

[14] T. Kulesza, M. Burnett, W.-K. Wong, and S. Stumpf, "Principles of explanatory debugging to personalize interactive machine learning," in *Proceedings of the 20th international conference on intelligent user interfaces*, pp. 126–137, 2015.

[15] Á. A. Cabrera, W. Epperson, F. Hohman, M. Kahng, J. Morgenstern, and D. H. Chau, "Fairvis: Visual analytics for discovering intersectional bias in machine learning," in *2019 IEEE Conference on Visual Analytics Science and Technology (VAST)*, pp. 46–56, IEEE, 2019.

[16] F. Küppers, A. Haselhoff, J. Kronenberger, and J. Schneider, "Confidence calibration for object detection and segmentation," in *Deep Neural Networks and Data for Automated Driving: Robustness, Uncertainty Quantification, and Insights Towards Safety*, pp. 225–250, Springer International Publishing Cham, 2022.

[17] J. Dodge, R. Khanna, J. Irvine, K.-H. Lam, T. Mai, Z. Lin, N. Kiddle, E. Newman, A. Anderson, S. Raja, *et al.*, "After-action review for ai (aar/ai)," *ACM Transactions on Interactive Intelligent Systems (TiiS)*, vol. 11, no. 3-4, pp. 1–35, 2021.