Visual scrolling detection to enhance GUI agent training

Chena Lee
Stanford University
chenalee@stanford.edu

Abstract

We address the problem of automatically detecting user scrolling actions (and, more generally, mouse-based GUI actions) from screen-recorded videos. We implement and evaluate two approaches: (1) a Lucas–Kanade (LK)–based optical flow pipeline with post-processed motion filtering, and (2) a TimeSformer-based transformer model trained with soft temporal labeling to handle annotation uncertainty.

We use the GUI-World dataset, where annotations are sparse and temporally imprecise. The LK method achieves higher recall and lower precision (0.237 precision, 0.369 recall across 500 videos), despite extensive post-processing and tuning. The TimeSformer model offers more flexible multi-class detection and achieves improved scroll detection results (0.414 precision, 1.000 recall across over 4000 videos) when trained from scratch using BCEWithLogitsLoss and oversampled scroll data. Notably, we found that precision was impacted by missing or delayed annotations, and that many model-predicted scrolls appeared valid despite being labeled as false positives.

We describe our methods, dataset, experiments, and analysis of both quantitative and qualitative results, including observed failure modes and annotation noise. We conclude with proposed future work leveraging large-scale unlabeled video corpora and suggest that self-supervised temporal modeling could mitigate annotation challenges in GUI action detection.

1. Introduction

1.1. Problem Statement and Motivation

Detecting scrolling actions in user interface (UI) videos is crucial for training GUI agents that can navigate realistic, dynamic software environments. Unlike clicks or keystrokes, scrolling involves sustained, often subtle vertical motion that is not trivially localized to a single frame. Accurately identifying when and how users scroll enables downstream agents to understand context shifts (e.g., revealing new UI elements) and execute tasks in complex desktop applications.

Our goal is to automatically label scrolling segments (and, by extension, other mouse actions) from unlabeled or sparsely annotated videos, reducing reliance on manual annotation. Once annotated, these segments can be used to train deep learning–based GUI agents that generalize to real-world software—particularly in domains like healthcare, where rigid systems and poor interoperability pose challenges.

We note that manual annotations in datasets like GUI-World are often imprecise, especially for scrolling. Annotators provide only a single frame index for each action, typically delayed by several frames relative to when the scroll ended. This introduces ambiguity in both evaluation and training.

1.2. Input and Output Definitions

Input: A variable-length video of GUI activity (e.g., screen recording of a desktop application), represented as a sequence of RGB frames $\{F_t\}_{t=1}^{T}$

Output (LK Method): A set of time ranges $S = \{[t_{s1}, t_{e1}], [t_{s2}, t_{e2}], \dots\}$ indicating frame ranges where vertical scrolling occurs.

Output (Transformer Method): For each contiguous clip of L frames (e.g., L=32), the model outputs a sequence of soft-labeled probabilities $p_{t,c}$ over C action classes (e.g., scroll, click, none), for t = 1, ..., L.

1.3. Contributions

- Revisit and implement a sparse optical flow–based LK pipeline for scrolling detection, including sliding-window smoothing and post-processing filters to reduce false positives.
- Train a TimeSformer (spatiotemporal transformer) model on the GUI-World dataset, using soft labeling (Gaussian weights over a "label window" of frames) to handle annotation uncertainty.
- Compare LK and transformer approaches on identical

data splits, analyzing both quantitative metrics (precision, recall, loss) and qualitative failure modes.

- Highlight the impact of class imbalance and imprecise annotations on training, and evaluate robustness via controlled overfitting.
- Discuss future directions, including large-scale unlabeled pretraining inspired by OpenAI's Minecraft agent.

2. Related Work

Prior work related to detecting GUI-based actions in videos spans three broad categories: classical optical flow methods, transformer-based video models, and GUI-Agent related work.

Classical Optical Flow:

- Lucas–Kanade (LK) [1] remains a lightweight, interpretable baseline for motion tracking. Its sparse feature matching is susceptible to UI noise and jitter.
- Farnebäck [9] and DeepFlow approaches offer dense alternatives, but introduce high memory use and false positives in desktop GUI settings.
- FlowNet [2] and RAFT [3] use CNNs to estimate flow, improving robustness to large displacements, but require large labeled datasets and GPU runtime.

Transformer-Based:

- TimeSformer [6] introduced factorized spatial-temporal attention for action recognition.
- ViViT [10] and Video Swin Transformer [11] extend Vision Transformers (ViT) [7] to handle long sequences with patch-level semantics.
- VPT [4] shows the value of large-scale pretraining on unlabeled video, highlighting the potential of using GUI screen recordings in a similar way.

GUI-Agent:

- UGround [5] maps natural language commands to UI regions using LLMs and synthetic screenshots, but does not handle motion detection or temporal dynamics. Dataset they used is synthetically generated.
- Screen2Vec [12] learns GUI screen embeddings.
- DeepClick [13] detects click actions using CNN+LSTM, a simpler task than modeling scroll, which requires capturing sustained movement.

Many existing GUI datasets are synthetic or mobile-based, and fail to capture the complex scrolling behavior present in desktop applications. Our work contributes a side-by-side evaluation of classical vs. modern approaches using a noisy real-world dataset.

3. Methods

3.1. Lucas{Kanade (LK) Based Motion Segmentation

This section outlines the implementation details for both the classical Lucas–Kanade (LK) optical flow pipeline and the transformer-based TimeSformer model. We describe the feature extraction and post-processing used in the LK method, the model architecture and loss function used for the transformer, and the evaluation logic applied to both.

Our optical flow pipeline builds on the Lucas–Kanade method to detect scrolling based on sparse vertical displacements. For each frame pair, we compute feature tracks using $cv2.goodFeaturesToTrack$ and estimate motion via $cv2.calcOpticalFlowPyrLK$. We extract vertical motion vectors and apply a direction threshold and consistency filter to detect candidate scrolling frames.

To reduce noise, we use a sliding window of size 5 and trend-based labeling followed by a hysteresis smoothing step that requires consistent trends to persist for multiple frames before changing class. We then merge adjacent candidate segments using track continuity heuristics: if a sufficient number of tracked points show consistent displacement direction and magnitude across a gap, the segments are fused.

Specifically, if two scrolling segments are separated by a short gap ($\leq$ 10 frames), we analyze the optical flow tracks from the last frame of the first segment and the first frame of the second. If a sufficient number of tracked points ($\geq$ 25% or at least 100) continue their motion across this gap in the same direction (scroll up or down), we consider this evidence of consistent scrolling and merge the segments. This continuity heuristic is essential to address intermittent tracking loss and small pauses during user scrolling, which otherwise result in fragmented detections.

To filter out false positives, we require segments to pass multiple post-hoc checks: total vertical displacement must exceed 40 pixels; the 75th percentile of track length must be above 20 pixels; at least 30 "long" tracks must exist in the segment; and the average vertical motion per frame must exceed 2.0 px/frame. Visualizations of track overlays and flow fields are generated to assist manual inspection.

3.2. TimeSformer-Based Classification

We train a TimeSformer model from scratch on GUI-World clips. Each clip consists of 32 RGB frames, sampled with a stride of 2, and resized to 128×128 resolution.

Because annotations label only a single (approximate) midpoint frame per scroll, we apply soft labeling with a Gaussian kernel spanning ±10 frames. This creates a temporally smoothed ground truth label for the "scroll" class.

The model is trained using $BCEWithLogitsLoss$, with a high pos_weight to address the extreme class imbal-

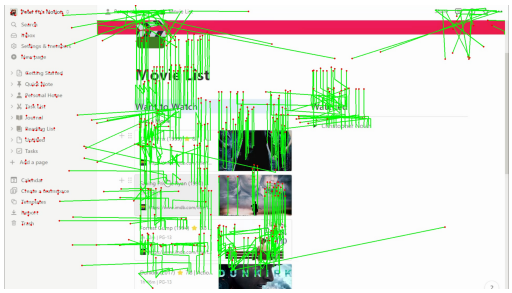

Figure 1. Tracks created to identify scrolling behavior

ance between ”scroll” and ”no scroll” labels. Empirically, we found that training heavily penalized false negatives improved the model's ability to detect rare scroll segments, but also led to overfitting when trained for long durations. To further counteract class imbalance, we oversample scroll-labeled training clips by a factor of 2×. This ensures better representation of rare scroll events during training. All inputs are resized to 128×128, and no pretrained weights are used.

Our implementation adopts a simplified but faithful version of the TimeSformer architecture. Each video clip of shape is first passed through a patch embedding module that flattens each frame into non-overlapping patches and projects them into a fixed embedding dimension. We use 4 transformer encoder layers with 4 heads each.

The core of the model alternates between spatial and temporal attention:

- Spatial attention is applied across patches within each frame by reshaping the sequence as , where is the number of patches per frame.

- After spatial encoding, the output is reshaped back to and averaged over patches to yield one token per frame.

- Temporal attention is then applied across the frames using a positional encoding of shape , added to each frame embedding.

Finally, the model predicts a classification score for each frame using a linear head over the temporally encoded tokens.

3.3. Evaluation

To handle annotation uncertainty, we allow detected segments to match ground-truth scrolls with up to ±10 frame tolerance. A true positive is defined as any predicted segment overlapping a labeled scroll region within this window. We compute precision, recall, and F1 score across all test videos. We also report per-video averages to assess consistency.

4. Dataset and Features

We used the GUI-World dataset, which consists of over 4000 videos capturing real user interactions with desktop applications. Each video includes a sparse set of annotations indicating GUI actions such as clicks, hovers, and scrolls. The annotations are temporally imprecise, typically annotating just a single frame per action and with some delay. This is particularly problematic for scrolling, which by nature spans multiple frames and cannot be meaningfully captured by a single timestamp.

From the full dataset, we extracted approximately 231 videos for training and testing. Each frame is resized to 128×128 resolution and stored as RGB. To feed the model, we sample contiguous clips of 16–32 frames at a stride of 2–4, resulting in overlapping clip windows. Frames are normalized but no augmentations (e.g., cropping, flipping) were applied.

For each annotated scroll, we assign soft labels using a Gaussian kernel centered on the annotated frame, covering nearby ±10 frames. In cases with no annotation, frames default to ”none” class. The dataset is highly imbalanced, with frames annotated as scroll events constituting less than 1% of all frames. Scrolls often go unlabeled, complicating supervised training and evaluation.

The annotations are stored in structured JSON format, which includes frame-specific events along with metadata and QA descriptions. For example:


```
{
"system": "Windows",
"app": ["Todoist"],
"goal": "Create a new team",
"keyframes": [
{"frame": 24, "mouse": "hover"},
{"frame": 291, "mouse": "click", "keyboard": "input
{"frame": 330, "mouse": "scroll"}
],
"video_path": "software/12.mp4"
}
```


This annotation sparsity impacts both training and evaluation. During evaluation, the model may detect real scrolls that were never labeled by annotators, which would incorrectly count as false positives. Thus, the reported precision may underestimate true model performance.

5. Experiments, Results, and Discussion

5.1. Experimental Setup

We trained and evaluated both the LK-based and TimeSformer-based models using the GUI-World videos. For TimeSformer, we trained from scratch using the Adam optimizer with a learning rate of $5e^{-4}$, and batch size of 16. We trained for up to 10 epochs. No pretrained weights were used.

To handle label imbalance, we used `BCEWithLogitsLoss` with a high positive class

weight (e.g., `pos_weight` = 100.0) to emphasize scroll detection. We did not perform cross-validation due to compute constraints and instead reserved a portion of the videos as a validation set.

LK-based detections were produced using optical flow post-processing with tunable thresholds and segment merging logic (see Section 2). All hyperparameters (motion threshold, track count, segment length) were optimized manually based on qualitative inspection.

5.2. Metrics and Definitions

We evaluate model performance using precision, recall, and F1 score. A true positive is defined as a predicted scroll segment that overlaps any annotated scroll within ±10 frames. Let TP, FP, and FN denote true positives, false positives, and false negatives respectively:

$$\text{Precision} = \frac{TP}{TP + FP}, \quad \text{Recall} = \frac{TP}{TP + FN},$$
$$F1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

Note that because annotations are sparse and incomplete, some model-correct scroll detections are labeled as false positives. Thus, precision may underestimate true model quality.

LK Method Results (500 videos)

Metric	Value
Overall Precision	0.237
Overall Recall	0.369
Overall F1 Score	0.289
True Positives	52
False Positives	167
False Negatives	89
Avg Precision / Video	0.528
Avg Recall / Video	0.801
Avg F1 Score / Video	0.533

Table 1. Evaluation metrics for LK method on 500 GUI videos.

Interesting true failure cases included a video where an UI item was added by a user. This caused all components below the new UI item to be pushed down, making LK pipeline think this is scrolling.

Transformer (TimeSformer) Results:

These experiments demonstrate key trade-offs in scroll detection performance. A moderate pos_weight of 5.0 already yields strong recall (¿0.9), but increasing to 10.0 helps eliminate remaining false negatives, achieving 100% recall. However, this comes at the cost of precision, which begins to degrade with prolonged training—dropping from 0.423 after 1 epoch to 0.267 after 3 epochs. This suggests that the model becomes increasingly prone to overfitting, interpreting ambiguous motion patterns (e.g., mouse hovers) as scrolls. Notably, recall remains consistently high, indicating that false negatives are relatively easy to eliminate with sufficient positive weighting and oversampling, while precision must be managed more carefully through regularization or better labeling.

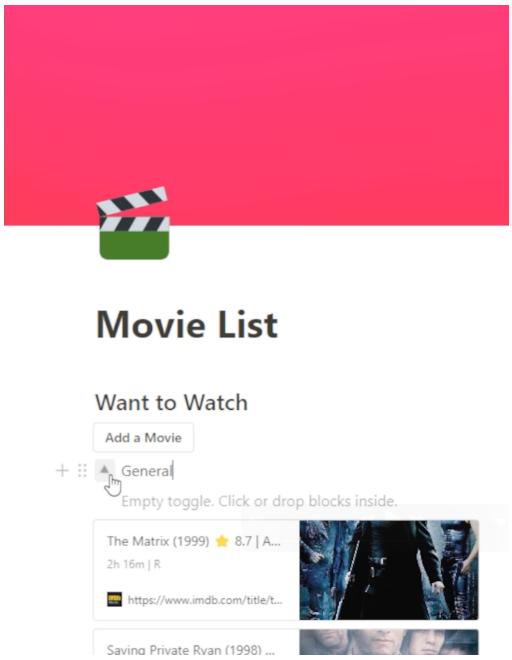

Figure 2. False positive case. Added UI item pushed bottom components down at once.

Metric	Value
Overall Precision	0.423
Overall Recall	1.000
Overall F1 Score	0.594
True Positives	22638
False Positives	30933
False Negatives	0

Table 2. Evaluation metrics for TimeSformer on over 4000 GUI videos. pos_weight = 10.0, 1 epoch training

Metric	Value
Overall Precision	0.407
Overall Recall	0.903
Overall F1 Score	0.561
True Positives	19692
False Positives	28687
False Negatives	2106

Table 3. Evaluation metrics for TimeSformer on over 4000 GUI videos. pos_weight = 5.0, 1 epoch training

Metric	Value
Overall Precision	0.407
Overall Recall	0.997
Overall F1 Score	0.578
True Positives	21733
False Positives	31647
False Negatives	65

Table 4. Evaluation metrics for TimeSformer on over 4000 GUI videos. pos_weight = 5.0, 2 epoch training

Metric	Value
Overall Precision	0.267
Overall Recall	1.0
Overall F1 Score	0.422
True Positives	21798
False Positives	59834
False Negatives	0

Table 5. Evaluation metrics for TimeSformer on over 4000 GUI videos. pos_weight = 5.0, 3 epoch training

- Validation loss reached as low as 0.024 after 100 epochs under overfitting.
- After 1 epoch, training loss reached 0.4848 with validation loss 0.4760 with pos_weight 10.0.

5.3. Hyperparameters

We explored the effect of changing `pos_weight` in the BCE loss to penalize misclassified scroll frames more heavily. Lower weights led to underprediction, while high values improved recall but worsened precision due to false positives on mouse movement frames. Frame stride, clip length, and embedding dimensions used were stride=2, clip=32, embed=192.

5.4. Qualitative Results and Error Modes

Despite strong quantitative performance, qualitative inspection revealed key limitations of both the model and the dataset. Many apparent false positives involved mouse movements, or drag gestures that resemble short scrolls. These cases reflect the difficulty of defining clear semantic boundaries between GUI gestures in the absence of high-quality annotation.

In several examples, the model correctly detected scrolling segments that were visually obvious (e.g., vertical list movement, scrollbar motion), but no corresponding labels existed in the dataset. These cases were penalized during evaluation due to annotation sparsity. Conversely, the model sometimes split a single scroll action into multiple detections—especially when the user paused or wobbled mid-scroll. Since the dataset annotates only one frame per

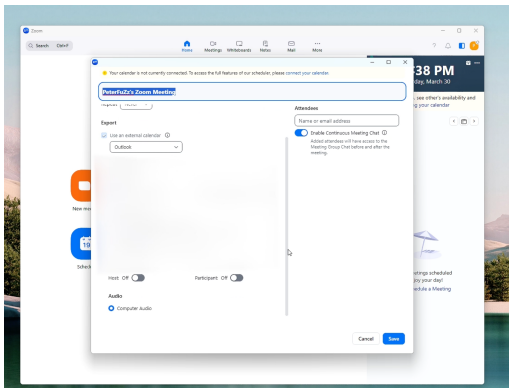

Figure 3. Original dataset did not label scroll action

action, these splits count as redundant true positives or even false positives depending on overlap logic.

Predicted scroll probabilities showed that high pos_weight values caused the model to saturate quickly—producing extended high-confidence scroll spans around short gestures. This aggressive labeling explains the tradeoff between improving recall and degrading precision. A more nuanced labeling policy, or access to frame-level dense scroll annotations, would help mitigate these artifacts and produce a more faithful segmentation.

In summary, while our model achieves near-perfect recall under some configurations, the noisiness and sparsity of the underlying annotations place a ceiling on achievable precision. Future improvements should focus on improving label quality and explicitly modeling gesture uncertainty.

6. Conclusion and Future Work

In this report, we examined two distinct approaches for detecting scrolling actions in GUI screen recordings: a classical Lucas–Kanade (LK) optical flow pipeline and a transformer-based TimeSformer model. The LK pipeline, while interpretable and lightweight, achieved modest performance (F1 = 0.289), struggling with the subtlety of GUI scroll motion and the noise inherent in sparse tracking. Conversely, the TimeSformer model trained with soft labels and class balancing strategies proved far more effective, achieving an F1 score of 0.594 after just one epoch.

These results highlight that transformer-based architectures are better suited for the temporal and spatial complexity of scroll detection, especially when paired with thoughtful loss weighting and data balancing. Our use of pos_weight and oversampling allowed the model to reach perfect recall, and multiple epochs demonstrated the precision–recall tradeoff inherent in overfitting vs. generalization.

In future work, we aim to address two major limitations: annotation quality and model generalization. Sparse and imprecise annotations currently limit the precision of both

training and evaluation. One promising direction is leveraging large-scale unlabeled video corpora for self-supervised pretraining, inspired by recent advances in Minecraft agents and ViT-based video models. With more compute and access to richer labels or human-in-the-loop corrections, we also plan to incorporate multi-modal cues (e.g., cursor trajectories, text boxes, UI trees) to boost robustness and disambiguate ambiguous visual patterns.

References

[1] Bruce D. Lucas and Takeo Kanade. "An Iterative Image Registration Technique with an Application to Stereo Vision." International Joint Conference on Artificial Intelligence (IJCAI), 1981.

[2] Alexey Dosovitskiy et al. "FlowNet: Learning Optical Flow with Convolutional Networks." IEEE International Conference on Computer Vision (ICCV), 2015.

[3] Zachary Teed and Jia Deng. "RAFT: Recurrent All-Pairs Field Transforms for Optical Flow." European Conference on Computer Vision (ECCV), 2020.

[4] Eric Wu et al. "VPT: Learning Visual Prompt Tuning for Vision Transformers." arXiv preprint arXiv:2203.12152, 2022.

[5] Yujia Liang et al. "UGround: Open-Vocabulary GUI Grounding using Vision-Language Models." arXiv preprint arXiv:2305.17086, 2023.

[6] Gedas Bertasius, Heng Wang, and Lorenzo Torresani. "Is Space-Time Attention All You Need for Video Understanding?" International Conference on Machine Learning (ICML), 2021.

[7] Alexey Dosovitskiy et al. "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale." arXiv preprint arXiv:2010.11929, 2020.

[8] Ze Liu et al. "Video Swin Transformer." IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2022.

[9] Gunnar Farnebäck. "Two-Frame Motion Estimation Based on Polynomial Expansion." Scandinavian Conference on Image Analysis, 2003.

[10] Anurag Arnab et al. "ViViT: A Video Vision Transformer." IEEE/CVF International Conference on Computer Vision (ICCV), 2021.

[11] Huazheng Geng et al. "Screen2Vec: Semantic Embedding of GUI Screens and GUI Elements." arXiv preprint arXiv:2205.15493, 2022.

[12] SeongMin Kim, Yena Lee, and Sung-Bae Cho. "DeepClick: Classification of Mouse Clicking Activities Based on CNN and LSTM." IEEE Access, 2019.