SmartPerf: Automated Speed Performance Test from Visual Perspective

Letao Li[1], Likai Qiu[2,*], Shuang Li[3], Jiaqi Wang[4] and Hualiang Zhu[5]
[1,2,3,4]Shanghai, China [5]Beijing, China
liletao@baidu.com, qiulikai@baidu.com, lishuang30@baidu.com, wangjiaqi20@baidu.com, zhuhualiang@baidu.com
*corresponding author

Abstract—Mobile application performance is crucial for enhancing product competitiveness in the market. As mobile application development grows more intricate, automated performance testing has become increasingly challenging. Most testing tools rely on source code or binary files, which limits the evaluated scenarios and requires testers to possess coding skills. This paper presents a full-process intelligent speed performance testing tool, SmartPerf, from automated test case recording to analysis report output. SmartPerf simulates user behaviors to record cross-platform test scripts. Subsequently, the tool segments the recorded screen data into frames during replaying, and the general transition point recognition automatically identifies the first and last frames. SmartPerf also supports uploading long screen recording videos and determines each transition point's first and last frames in continuous multiple actions switching through the temporal convolution segmentation. This paper introduces the user-friendly evaluation metric, mAcc@5. The results indicate an accuracy of 0.9 for general transition point recognition in single actions and 0.76 for temporal convolution segmentation in continuous multiple actions, significantly reducing labor costs. The conclusions given by the tool reports reflect the performance quality effectively, consistent with the market reality.

Keywords–Automated Performance Testing; Record and Replay; GUI; Temporal Convolution Segmentation

1. Introduction

Nowadays, mobile applications have been widely used in all walks of life [1], such as search, online shopping, reading, etc. People often ignore the performance of mobile applications, which generally includes application launch time, CPU usage, fluency, and operation response time. The speed performance focus on the interaction speed between users and Apps, that is, the feedback response speed of Apps to a user's input signal. Nielsen [2] proposed that speed must be the overwhelming criterion for user experience design at an earlier time. The quality of speed performance is directly related to user experience, affecting user retention and product revenue [3]. Since mobile application technologies have become increasingly intricate, On the one hand, the fragmentation of mobile applications, such as cross-system, cross-device, and hybrid applications, increases the cost of automated performance testing. Some front-end automated tools like Appium, which integrate traditional frameworks, rely on scripts like Xpath or ID to locate widgets, which are relatively unstable. Therefore, tools like Sikuli [4] and Airtest [5] based on visual solutions have emerged. LIRAT [6] utilizes layout and image recognition to achieve cross-platform record and replay. Robotic arms to operate real devices is also a popular front-end automated testing solution [7][8][9], where non-invasive and fully automated simulation of user behaviors achieves recording and replaying. On the other hand, it becomes complex and dynamic for user interface display and interaction. Technologies such as JavaScript and template engines implement front-end page chunking and dynamic updating through asynchronous requests, callback functions, etc., to improve the response speed of the program, which brings challenges to performance testing.

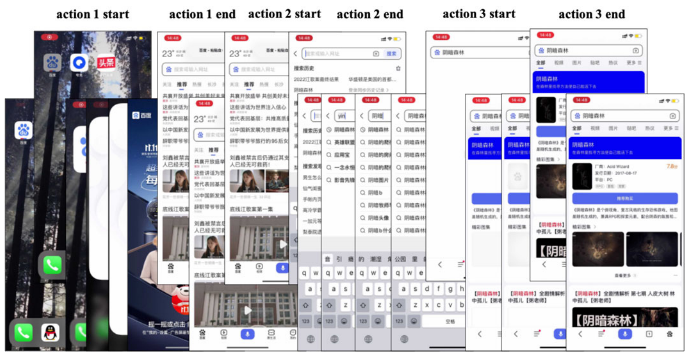

Figure 1. An Example: Continuous Multiple Actions for Baidu App

Speed performance testing can play a crucial role in preventing experience degradation, verifying performance improvements, and gaining insights into the competitive position of industry products. Previously, testers typically conducted performance testing at the system test level. However, there have been suggestions to shift performance testing to an earlier stage, such as the unit test level, to enhance the assurance of a smooth and problem-free process [10]. An automated evaluation tool that efficiently and accurately completes the evaluation loop while delivering credible evaluation reports is crucial for enhancing evaluation efficiency.

This article introduces SmartPerf, an automated performance testing tool based on real devices. The automated recording and replaying module with image recognition can record and replay test cases with one click. SmartPerf records each operation's automatic replaying process video and calibrates the first and last frames through a general transition point algorithm after splitting frames. Earlier, Jovic et al. [11] proposed the necessity of speed performance testing from the GUI perspective. The back-end scripting method greatly differs from the user's perception, where the triggering response and termination of events cannot correspond one-to-one with the start and end of the front-end page. Therefore, using the visual method to determine the transition point of the page is closer to the user's perception. In addition, driven by business needs, SmartPerf also supports uploading long screen recording data containing continuous multiple operations and automatically performs speed performance analysis on the response time of each action (see Figure 1). Replacing and measuring the response time of each action separately with one screen recording directly giving the response time of each action in multiple operations greatly improves efficiency. In the report stage, the evaluation results display a confidence interval, and comparative conclusions on the merits and demerits are given by recording the performance data of historical versions. The challenges we need to solve are:

- How to efficiently and stably record and replay automated test cases?
- How to efficiently determine the first and last frames because mobile application scenarios are diverse?
- How to conduct confidence assessment for speed performance testing and provide users with credible reports?

2. Related Work

There are two main approaches for speed performance testing: those based on log points and those based on screen recording analysis. The log point-based scheme [12] involves the insertion of collection points at critical positions within the code, followed by statistical analysis of the logged data to derive speed metrics for the evaluation scenario. This approach offers the advantages of being cost-effective and easy to implement, as the results are generated through sampling and statistical analysis of online log data, which closely reflects the code's actual running environment. However, it has limitations, as it requires access to the source code and cannot be used to evaluate competitive applications. In complex business scenarios involving asynchronous data requests, inserting log points at each stage to accurately capture relevant moments during actual operations proves challenging. Moreover, to prevent any adverse impact on the performance of the business itself, simple data collection is often the primary choice, which may not necessarily yield speed metrics closest to the user's perception.

Jovic et al. [10] proposed a method that involves capturing event response times through logging, tracking latency information for each GUI event, and utilizing cumulative latency distribution to characterize the overall responsiveness of the program. Another performance testing tool, AppSpeedXray [13], adopts a different approach by recording test cases using UI Automator and collecting data from binary files. It also analyzes CPU utilization, loading time, rendering time, and other pertinent metrics to generate a comprehensive performance score for the application.

Visual solutions have been employed for speed analysis in both industry and academia [14]. Stagesepx [15] proposes a solution by splitting input videos into frames, classifying the stages, and automatically generating chart reports. It classifies individual frames, such as stable pages, blank pages, and loading status pages for cold start scenarios. If not recognized, it is considered as another state. Essentially, it is a classifier based on Support Vector Machine (SVM) and Histogram Of Oriented Gradient (HOG) or Convolutional Neural Network (CNN) classifier. One drawback of this classifier is that it needs to utilize previous and subsequent frame data, leading to discontinuous classification states, where the next state category includes the previous state. Confidence interval analysis result in a more accurate page transition time. Kim et al. [9] proposed a scheme using a robot arm to automatically capture video frames, using CNN to recognize page components and page similarity to identify the first and last frames. They utilize the "speed index" to evaluate the performance, which calculates the average time for the visible parts of a page to be displayed. The advantage is comprehensively judging the user's perception of page loading speed, unaffected by a single indicator such as FMP, FP, etc. The disadvantage is that it could be more intuitive for business people and easier to explain. However, The above tasks lack identification of solutions for continuous multi-action scenarios, and video action segmentation technology is introduced to solve this problem. CARL [16], the latest sequence contrastive learning for action segmentation, encodes the frame-wise action and then classifies each frame using classification networks. It is suitable for short videos within 1000 frames and cannot adapt to our long video scenarios.

Both methods first rely on automated UI testing tools like Appium and UI Automator to simplify the code writing or enable simulators to display execution status. We propose an automated performance testing tool to address the above challenges, closing the loop of the testing life cycle.

3. Methodology

SmartPerf, a full-process intelligent speed performance testing tool, mainly includes automated record and replay, speed performance analysis, and confidence reports. The testing tool is designed to prevent product deterioration and conduct competitive benchmarking. The framework is shown in Figure 2.

3.1. Automated Record and Replay

SmartPerf connects local smartphones to complete the recording of automated test cases such as click, swipe and reboot. Record and analysis are performed on the PC side, reducing resource consumption on the recording phones through efficient and streamlined driver modules. This paper is mainly about speed performance analysis, and we will only briefly describe the pre-execution automation without going into details.

It is challenging to address widgets in varying scenarios, and our automated tool provides two locating modes: "attribute positioning" and "visual positioning". We parse and reconstruct the page source, employing "visual positioning" to locate

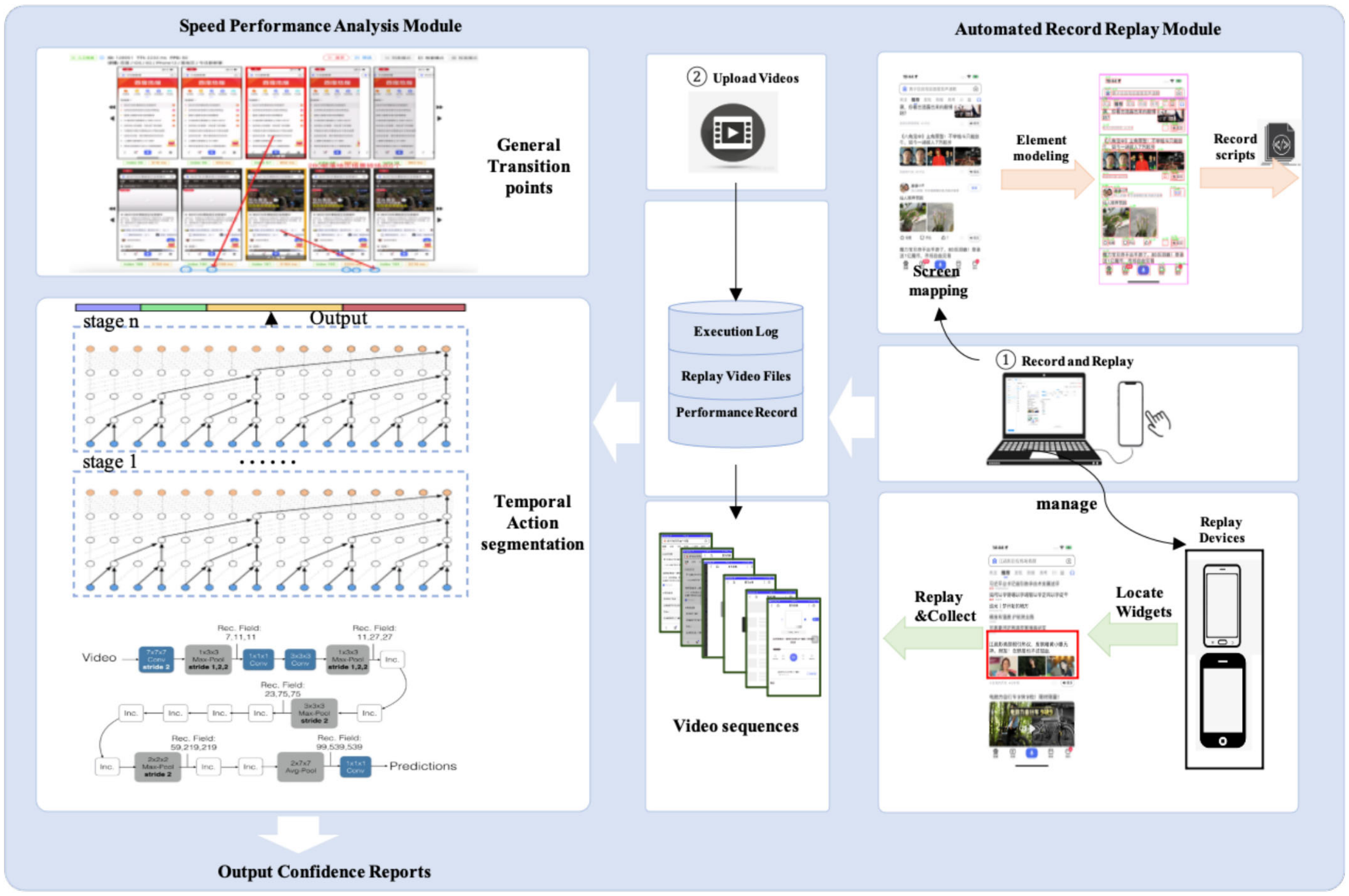

Figure 2. The Framework of SmartPerf: Automated Speed Performance Test from Visual Perspective

images and key icons using CNN. We group the pages through line detection and utilize OCR to model the pages.
Throughout the automated execution, for Android, we utilize AutomatorX (ATX) to gather widget information, including coordinate, ID, text, etc., and employ Android Debug Bridge (ADB) for automated operations. For iOS, communication with mobile devices is established through WebDriverAgentRunner (WDA). To minimize the impact on the resource consumption of hosts, we opt for tidevice to initiate WDA.
Finally, for obtaining high frame rate videos, we utilize scrapy for screen recording and ffmpeg [17] for frames extraction on the Android platform. On the iOS platform, we utilize the optimized iOS-minicap to obtain image streams and the corresponding timestamps.

3.2. Speed Performance Analysis

Efficiently calibrating the first and last frames becomes a crucial task. Motivated by real business scenarios, we present two solutions. The first is the general transition recognition, capable of condensing video frames into several transition points, which indicate the page changes. Users can readily identify the first and last frames from these transition points. This method exhibits high accuracy during the switch between individual operations and does not necessitate data training for coarse-grained annotation, making it applicable across all scenarios. The second is an action segmentation scheme based on temporal convolution networks, where multiple actions are delineated, facilitating the identification of the first and last frames of each action. If GTR is used for multi-action scenes, such as a 1200-frame video with three consecutive actions generating 23 transition points, manual calibration would be troublesome. We strive to identify a universal method to extract the transition points for each action.

General Transition Point Recognition. The essence of the first and last frame recognition lies in determining the state changes of page transitions. Typically, a page transition undergoes multiple state changes, and different businesses may have diverse criteria for identifying the last frame, some requiring FMP, others FP, or 80% of the similarity with a stable page. This highly customized approach complicates the task for evaluators and reduces availability. A feasible generalization approach is to provide the positions of each state change and allow users to confirm afterward.
Calculating the similarity between consecutive frames and employing binary search to rapidly obtain the labels for each page, we can adjust the similarity threshold to obtain different numbers of state changes. Usually, the similarity threshold is fixed, and a higher threshold results in fewer state changes,

while a lower threshold leads to more. Finally, merging is performed to retain only one state change if the distance between adjacent state changes exceeds x frames. Through this process, a sequence of 280 frames can be compressed into six state changes, which can be quickly located using tool shortcuts. Cross-frame calibration enables the first frame determination in Android scenarios, and OCR can assist in identifying cross-frames with an accuracy approaching 100%.

Temporal Action Segmentation. Screen recording videos are split into frames at 60 fps, while common video frame rates are 30 fps or even 15 fps. Speed performance testing relies on a higher frame rate to reduce speed evaluation errors, which also poses challenges for subsequent first and last frame recognition.

GTR is used for initial pre-annotation and later refined by the annotator. The annotation process involves identifying the first and last indices of each action, requiring the preparation of 60 videos for each task. For video embeddings, we extract RGB features, which have shown satisfactory results in experiments, avoiding the time-consuming optical flow extraction. The I3D model [19] inflates 2D convolutions into 3D convolutions, thereby incorporating temporal information and directly utilizing common 2D networks. We adopt the Inception network as a backbone, maintaining the overall structure unchanged. The I3D model is pre-trained on ImageNet for feature extraction, skipping the feature training step. The input size is $(224, 224)$, and the dimension of features is $(N, 1024, T)$, where T means the length of each sequence and N means the number of videos.

MS-TCN++ [18] is utilized to implement action segmentation, which infers the action category for each frame in a video. It mainly relies on temporal convolution networks to aggregate features, which utilizes causal convolution [20] to obtain more historical information to capture longer-range dependencies and increase the receptive field through dilated convolution, eventually achieving frame-level classification of videos. To further improve the classification effect, MS-TCN++ utilizes a multi-stage TCN [20] to further fine-tune the classification results and increase the accuracy of action segmentation.

Since the video sequences are continuous, cross-entropy loss can easily lead to over-classification or incoherent results. Therefore, MS-TCN++ introduces an additional loss $\mathcal{L}_{T-MSE}$ to smooth the results and improve coherence (see Eq. 1, 2 and 3).

$$\mathcal{L}_{T-MSE} = \frac{1}{TC}\sum_{t,c}\hat{\Delta}_{t,c}^2 \tag{1}$$

$$\hat{\Delta}_{t,c} = \begin{cases} \Delta_{t,c} & : \Delta_{t,c} \leq \tau \\ \tau & : otherwise \end{cases} \tag{2}$$

$$\Delta_{t,c} = |\log y_{t,c} - \log y_{t-1,c}| \tag{3}$$

Where T is the frame length of the video, C is the number of classes, and $y_{t,c}$ is the predicted confidence score of class c. If the behavior classification difference between the adjacent frames is larger, it avoids sudden changes in the classification results on the time axis, thereby achieving continuity of classification at each stage.

In other words, this additional loss penalizes large differences in predicted labels between adjacent frames. By minimizing this loss, the model is encouraged to output smooth label predictions over time and avoid sudden label changes between frames.

3.3. Confidence Reports

Speed performance testing is affected by many factors, such as network, time period, etc. To minimize errors, we adopt round-robin evaluation for the tested scenarios to obtain groups of the first and last frame verification data. On this basis, to effectively eliminate accidental errors, the Grubbs criterion, which performs better for small samples and is internationally recommended, is selected. Suspicious values are calculated by arranging the first and last frame verification data, and the calculation formula is as Eq. 4.

$$G_i = \frac{|x_i - \bar{x}|}{s} \tag{4}$$

where $\bar{x}$ is the mean value of the test data, and s is the standard deviation. We utilize the Grubbs table to determine if the calculated z-score (G_i) exceeds the critical value G_α at a given significance level α. Suppose $G_i > G_\alpha$, the corresponding data is considered as an outlier and removed. This process is repeated until there are no more outlier data points, resulting in D_e without any abnormal values.

Since we cannot obtain the results of all users under the evaluation scenario, we further obtain the interval estimation of unknown parameters based on sample statistics of D_e, namely confidence interval, as Eq. 5.

$$P(\mu - 1.96\frac{\sigma}{\sqrt{n}} \leq M \leq \mu + 1.96\frac{\sigma}{\sqrt{n}}) = 0.95 \tag{5}$$

Where n is the number of the test data, μ and σ are the mean and standard deviation of the sample data. Therefore, $(\mu - 1.96\frac{\sigma}{\sqrt{n}}, \mu + 1.96\frac{\sigma}{\sqrt{n}})$ is the confidence interval of the unknown parameter M. The confidence interval can quantify the uncertainty of estimation and provide a rough heuristic approach that the probability of the true mean being greater than the obtained upper limit is about 2.5% (if the confidence level is 95%, α is 0.05 and the $z_{\alpha/2}$ is 1.96 from the confidence interval table). In order to visualize the confidence interval and to provide a clear comparison with other applications, the reports of SmartPerf introduce the box plot. The width of the box reflects the size of the sample, in which the larger the sample size, the wider the box.

4. Evaluation

Assuming the stability of the record and replay module, which provides reliable screen recording and frame data, we assessed the accuracy and applicability of the proposed first and last frame calibration scheme and the effectiveness of the reports.

4.1. Accuracy of General Transition Point Recognition

We collected speed performance testing requirements from the industry, including 24 different testing tasks covering six applications. Each scenario was executed 70 times, with two different smartphones selected from both iOS and Android. During the replay process, it is more appropriate to confirm the first frame utilizing cross-frame for Android devices.

We analyzed the correlation between algorithm accuracy and manual effort. Without using the algorithm, manual calibration for a single scenario took 30 seconds. However, achieving 100% calibration accuracy with 15 frames further reduced the manual effort to 20 seconds. Additionally, using five frames with 90% accuracy only took 10 seconds for screen recording calibration. Furthermore, if there was a 95% accuracy with 0 frame difference, no manual calibration was needed, achieving full automation. Based on this analysis, we computed the compressed accuracy metric mAcc@5, considering any transition point hit within a range of 5 frames as a positive hit.

From Table I and Table II, the average accuracy with mAcc@3=0.814 and mAcc@5=0.9 in 24 scenarios could achieve a 66% reduction in labor cost.

TABLE I
ACCURACY OF GTR ON ANDROID

Device	Scene ID	mAcc@3	mAcc@5
Device0	1	0.56	1
Device0	2	0.9	1
Device0	3	0.91	0.93
Device0	4	0.84	0.95
Device0	5	0.95	1
Device0	6	0.97	1
Device0	7	0.81	1
Device0	8	0.92	0.96
Device1	9	0.9	1
Device1	10	0.66	1
Device1	11	0.88	1
Device1	12	0.38	0.47
Device1	13	1	1
Device1	14	0.26	0.38
Device1	15	0.95	0.95
Device1	16	0.61	0.88

4.2. Accuracy of Temporal Action Segmentation

We obtained screen recording data from crowdsourced data suppliers, specifically capturing the process of the three applications from cold start, through the search input, to navigating to search results page. The dataset consists of 300 video recordings with 100 pieces per App, covered 7 iOS devices, 8 different OS versions, and network conditions including WiFi, 4G, and 5G. Each video recording includes the first and last frames of three different actions, representing six distinct action categories. On average, each video is approximately 20 seconds long.

TABLE II
ACCURACY OF GTR ON IOS

Device	Scene ID	mAcc@3	mAcc@5
Device3	17	0.68	0.76
Device3	18	1	1
Device3	19	0.57	0.61
Device3	20	0.96	0.96
Device3	21	0.75	0.9
Device3	22	0.84	0.9
Device4	23	0.98	1
Device4	24	1	1

TABLE III
ACCURACY OF ACITON SEGMENTATION

Application	F1@50	Edit	Acc	mAcc@5
APP1	83.6	89.7	91.01	70.6
APP2	97.4	98.0	97.7	84.0
APP3	81.9	90.8	92.5	73.9

To evaluate the accuracy of action segmentation, we chose 3 metrics commonly used in the industry: F1-Score@50, edit distance, and frame-wise accuracy (ACC). Since the uneven distribution of actions, for example, the search input action covers a longer period, while other action inputs are shorter (see in Figure 3), leading to metrics being more heavily influenced by long actions and hardly reflecting the effects of each action fairly. Moreover, we care about the accuracy of each segmentation boundary, and we introduce the mAcc@5, which is the average boundary accuracy of each category.

Table III and Figure 3 show the performance on APP2 is better, while the performance on APP1 scenarios is worse. This is because the loading of APP1 involves more complex page changes, leading to larger annotation errors and worse learning effects. The average accuracy of mAcc@5 is 0.76.

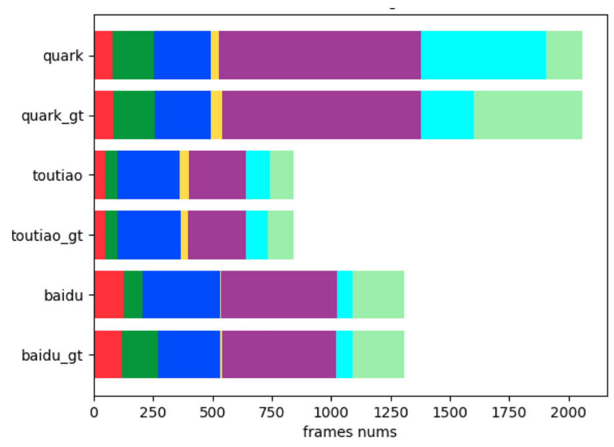

Figure 3. Segmentation Results with TCN of Three Apps in Multi Actions

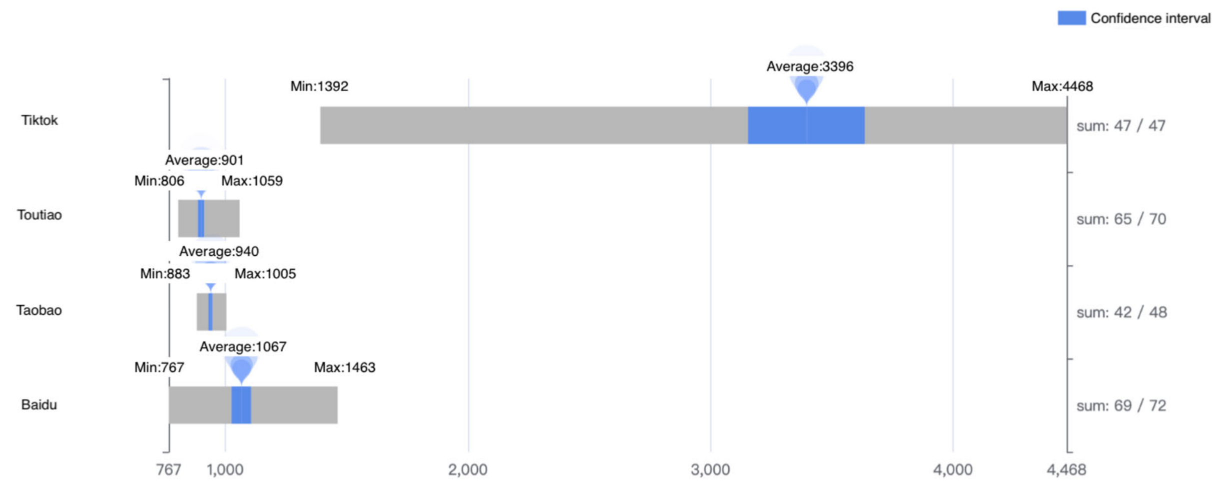

Figure 4. An Example: Box-and-whisker Plot of Cold Start Speed on iPhone13

The computation of video embedding is more time-consuming, and the inference of MS-TCN++ on a single video takes about 100ms.

4.3. Effectiveness of Confidence Reports

The expectation of cold start speed proposed by Apple at WWDC 2019 indicates that the pre-main stage is completed within 100ms, the loading of the first frame is realized within 400ms, and the homepage display is realized within 600ms. We conducted speed performance testing to evaluate the cold start time of four Apps. The device was selected in high, medium, and low types, respectively: iPhone 6SP, iPhone XS Max, and iPhone 13. The automated execution of SmartPerf was five steps:

1) Close App to be tested
2) Wait 5s (ensure that the App is completely closed)
3) Click the App icon on the desktop
4) Wait 7s (wait for the homepage to finish rendering)
5) Close App

To ensure the validity of the execution results, we repeat the above steps 70 times for each App. The first and last frames of the execution scenario were as follows:

- First frame: the moment when the device takes an automated click on the App icon.
- Last frame: the moment when the App homepage is fully loaded and stabilized

TABLE IV
COLD START TIME OF APPS

Device	Toutiao	Taobao	Baidu	Tiktok
iPhone 13	901ms	940ms	1067ms	3396ms
s iPhone XS Max	1710ms	1757ms	2326ms	3998ms
iPhone 6SP	3624ms	3822ms	5152ms	7908ms

As shown in Table IV, all four Apps failed to meet the expected cold start time of 600ms set by WWDC. The box plot in Figure 4 shows the confidence interval of the cold-start rendering speed of Apps on iPhone 13. sum is the percentage of valid data, where the denominator is the amount of data that was successfully executed and calibrated, and the numerator is the amount of valid data after the removal of anomalous data. The launch app stage of Tiktok and Taobao contains ads; thus, the data dropped from 70 to around 40 after calibration. Toutiao and Taobao start second on iPhone 13. However, the cold-start rendering speed of the Baidu APP on iPhone 13 is around 1 second (according to the comparative analysis of data on the market, and the evaluation data is consistent with the log data on the market). TikTok cold-start rendering speed on high-end devices reaches more than three seconds.

In the analysis reports, performance degradation is identified, and the identified problematic areas are communicated to the development team to investigate potential fluctuations. Continuous monitoring of online performance data is conducted. For example, it was discovered that the video replaying speed of the iOS product lagged behind by more than 50%, prompting adjustments to the replaying strategy in order to catch up with competitors. In another example, small-scale experiments in the visual search scenario demonstrated that performance optimization led to an increase in page views. After deployment, this increase in overall views was further validated through comprehensive analysis.

5. Conclusion

This paper introduces two speed analysis algorithms. For single transition actions, the first and last transition points correspond to the first and last frames through general transition point recognition. GTR has been extensively used and validated in the industry. For multiple continuous actions in uploaded screen recording data, each action's first and last frames can be automatically determined according to temporal action segmentation. With a dataset of 60 training videos, the mAcc@5 on APP2 reached 0.84. In subsequent tasks, we continuously train and learn the already calibrated segmentation data, leading to further improvement in precision. Additionally, personalized operators are provided to complement the GTR and meet diverse user needs.

References

[1] Samir, Amira, Huda Amin, and Nagwa Badr. ”A survey on automated user interface testing for mobile applications.” International Journal of Intelligent Computing and Information Sciences 22.2 (2022): 126-136.

[2] Nielsen, Jakob. ”The Need for Speed.” Alertbox, 1997, http://www.useit.com/alertbox/9703a.html.

[3] Li, Deguang, et al. ”The evolution of open-source mobile applications: An empirical study.” Journal of software: Evolution and process 29.7 (2017): e1855.

[4] Yeh, Tom, Tsung-Hsiang Chang, and Robert C. Miller. ”Sikuli: using GUI screenshots for search and automation.” Proceedings of the 22nd annual ACM symposium on User interface software and technology. 2009.

[5] ”Airtest: Automated Testing Private Cloud Solution.” Airtest, https://airtest.netease.com/home/.

[6] Yu, Shengcheng, et al. ”LIRAT: Layout and image recognition driving automated mobile testing of cross-platform.” 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE, 2019.

[7] Mao, Ke, Mark Harman, and Yue Jia. ”Robotic testing of mobile apps for truly black-box automation.” Ieee Software 34.2 (2017): 11-16.

[8] Craciunescu, Mihai, et al. ”Robot based automated testing procedure dedicated to mobile devices.” 2018 25th International Conference on Systems, Signals and Image Processing (IWSSIP). IEEE, 2018.

[9] Kim, Soohyun, Hyunsu Mun, and Youngseok Lee. ”Automatic mobile app speed measurement with robot.” 2020 21st Asia-Pacific Network Operations and Management Symposium (APNOMS). IEEE, 2020.

[10] Mishra, Deepti, and Alok Mishra. ”Complex software project development: agile methods adoption.” Journal of Software Maintenance and Evolution: Research and Practice 23.8 (2011): 549-564.

[11] Jovic, Milan, and Matthias Hauswirth. ”Performance testing of gui applications.” 2010 Third International Conference on Software Testing, Verification, and Validation Workshops. IEEE, 2010.

[12] Hu, Cuixiong, and Iulian Neamtiu. ”Automating GUI testing for Android applications.” Proceedings of the 6th International Workshop on Automation of Software Test. 2011.

[13] Mun, Hyunsu, and Youngseok Lee. ”Appspeedxray: A mobile application performance measurement tool.” Proceedings of the 35th Annual ACM Symposium on Applied Computing. 2020.

[14] Li, Peng Peter, and Shim-Young Ricky Lee. ”Automated Smart TV UI Performance Testing with Visual Recognition.”.

[15] ”Stagesepx.” GitHub, https://github.com/williamfzc/stagesepx.

[16] Chen, Minghao, et al. ”Frame-wise action representations for long videos via sequence contrastive learning.” Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2022.

[17] FFmpeg Developers. ”FFmpeg Tool.” Version be1d324. 2016. [Software]. http://ffmpeg.org/.

[18] Li, Shi-Jie, et al. ”Ms-tcn++: Multi-stage temporal convolutional network for action segmentation.” IEEE transactions on pattern analysis and machine intelligence (2020).

[19] Carreira, Joao, and Andrew Zisserman. ”Quo vadis, action recognition? a new model and the kinetics dataset.” proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2017.

[20] Bai, Shaojie, J. Zico Kolter, and Vladlen Koltun. ”An empirical evaluation of generic convolutional and recurrent networks for sequence modeling.” arXiv preprint arXiv:1803.01271 (2018).