Towards Context-aware Mobile Privacy Notice: Implementation of A Deployable Contextual Privacy Policies Generator

Haochen Gong[1,*], Zhen Tao[1,2*], Shidong Pan[1,2,†], Zhenchang Xing[1,2], and Xiaoyu Sun[1]

[1]*School of Computing, Australian National University*
[2]*CSIRO's Data61*
[*]Equal contribution [†]Corresponding author (shidong.pan@data61.csiro.au)

***Abstract*—Lengthy and legally phrased privacy policies impede users' understanding of how mobile applications collect and process personal data. Prior work proposed Contextual Privacy Policies (CPPs) for mobile apps to display shorter policy snippets only in the corresponding user interface contexts, but the pipeline could not be deployable in real-world mobile environments. In this paper, we present PRIVSCAN, the first deployable CPP Software Development Kit (SDK) for Android. It captures live app screenshots to identify GUI elements associated with types of personal data and displays CPPs in a concise, user-facing format. We provide a lightweight floating button that offers low-friction, on-demand control. The architecture leverages remote deployment to decouple the multimodal backend pipeline from a mobile client comprising five modular components, thereby reducing on-device resource demands and easing cross-platform portability. A feasibility-oriented evaluation shows an average execution time of 9.15 s, demonstrating the practicality of our approach. The source code of PRIVSCAN is available at https://github.com/buyanghc/PrivScan and the demo video can be found at https://www.youtube.com/watch?v=ck-25otfyHc.**

***Index Terms*—Contextual Privacy Policy, Privacy Notice, Privacy Policy, GUI Understanding**

I. INTRODUCTION

Mobile applications are an integral part of people's digital lives and collect a significant amount of personal data from users. In response to growing privacy concerns regarding data collection practices, mobile application companies are required to provide users with privacy notices, e.g., privacy policies, that describe their privacy practices. Regulations, such as the European Union's General Data Protection Regulation (GDPR) [1] and California's Consumer Privacy Act (CCPA) [2], require developers to provide understandable privacy notices to inform users about how their data is collected and used. Privacy policies have become the most common means of data practices disclosure. Mobile app platforms, such as Google Play and Apple App Store, also require applications on the platform to provide privacy policy links [3], [4].

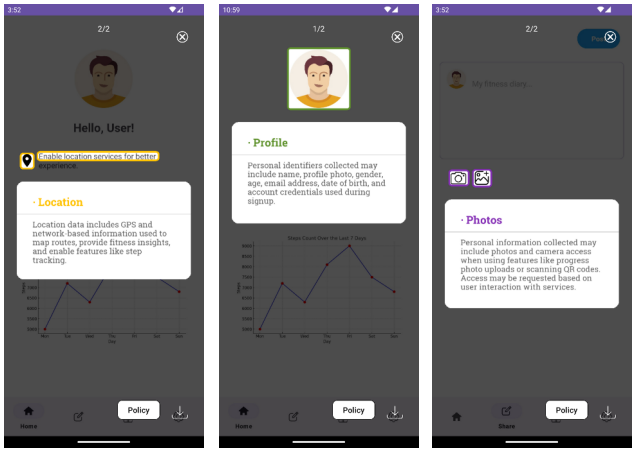
Although mobile applications require users to accept their privacy policies, these policies are often so lengthy that most users bypass them to gain immediate access to the app's functionality [5]. When users attempt to read the privacy policies, they often encounter complex legal language that is difficult for non-experts to understand [6]. Similarly, developers also face difficulties in adequately interpreting and understanding privacy policies due to limited legal knowledge, ultimately affecting their ability to implement compliant software behaviors [7]. Moreover, privacy policies are isolated from actual usage contexts, making it difficult for users to relate the abstract descriptions to their real-time interactions [8], thereby failing to adequately engage user awareness.

In response to these challenges, efforts have been made to design more usable privacy notices tied to their immediate context. Both iOS and Android introduce intrinsic permission popup mechanisms [9], [10], which notify users the first time the app attempts to obtain sensitive device permissions. Code-based IDE plugins, such as Matcha [11], have also been proposed to guide developers in creating accurate privacy nutrition labels for their apps. However, these approaches often fall short in efficacy, as they do not present contextually relevant statements in the privacy policy to users.

The concept of Contextual Privacy Policies (CPPs) has been proposed as a user-centered alternative [12], [13] that aims to extract shorter and more digestible segments from the traditional privacy policy, and dynamically display them in relevant contexts on the user interface (UI). CPPs help users become aware of potential data collection through UI elements and understand the relevant privacy disclosures in privacy policies. Such situated awareness bridges the gap between intention and action, facilitating informed consent in context.

In this paper, we present **PRIVSCAN**, the first CPP Software Development Kit (SDK) for mobile applications. The underlying approach of this tool was proposed in our previous work [14], known as the **SEEPRIVACY** framework. It leverages computer vision and natural language processing techniques to analyze user interface screenshots alongside corresponding privacy policy texts. It maps UI elements to specific personal data type and displays relevant policy policy snippets in context to help users understand the potential data practices associated with their interactions. While this framework marks a significant step toward contextualizing privacy, it remains at an experimental stage, lacking the capability for deployment in real-world mobile application scenarios, which

2643-1572/25/$31.00 ©2025 IEEE
DOI 10.1109/ASE63991.2025.00361



(a) Location (b) Profile (c) Photos

Fig. 1: Figure 1a displays CPPs for a location icon and text instruction; Figure 1b presents CPPs for a user profile icon; Figure 1c shows CPPs for device camera and album icons.

significantly limits its practical applicability.

To close this gap, we present the first CPP SDK for mobile applications. Our SDK introduces a lightweight, floating interface button that runs in real time on mobile apps. Upon user interaction, i.e., tapping the button, the system analyzes the current screen, identifies interface elements involved in accessing personal data, and highlights them with overlays. As shown in Fig. 1, for each identified element, **PRIVSCAN** presents the associated data type and relevant policy snippets. Developers can easily integrate this SDK into their applications, enabling privacy-aware design and enhancing user trust.

In Section II, we outline the engineering decisions and tool architecture. In Section III, we present an efficiency evaluation to show the feasibility of **PRIVSCAN**. The performance and human evaluation of the CPP generation pipeline were conducted in our previous work [14]. We conclude in Section IV.

II. TOOL ARCHITECTURE

A. Engineering Decisions

We implement a client-side SDK featuring a floating interaction button, which communicates with the CPP generation framework from our previous work [14] via a remotely hosted API. In this section, we introduce the engineering decisions behind the implementation, with consideration of efficiency, user experience and engineering complexity.

Deployment via a Remote API. The **SEEPRIVACY** pipeline involves both computer vision (CV) and natural language processing (NLP) models that are resource-intensive. On-device execution would therefore incur long inference times and reduced accuracy, especially on mid- to low-end smartphones [15]. Offloading the pipeline to a server ensures greater stability, delivering uniform performance across heterogeneous hardware. This remote-API pattern is now a common practice for mobile AI services and simultaneously offers clear cross-platform benefits, where the Python implementation can be reused across different platforms without costly re-engineering, thereby preserving compatibility with our original research pipeline.

SDK-Based Distribution. Distributing **PRIVSCAN** as a lightweight SDK promotes easy adoption and customization, thereby improving overall developer usability. Developers can integrate the SDK into their apps and tailor the tool to the characteristics of their apps. The SDK also guarantees a seamless in-app user experience, as CPPs appear within the host app. In contrast, a standalone monitoring software would demand continuous background execution and elevated permissions, e.g. overlay windows and usage access, whereas the SDK incurs lower overhead and aligns with the principle of least privilege.

Floating Button Interface. To minimize user effort, we designed a floating interactive button that hovers over the app interface. This widget reduces interaction cost and increases engagement, where a single tap reveals the policy snippet in the corresponding user interface context in a non-disruptive manner. The result is higher usability for app users, consistent with previous findings in mobile user experience that low-effort UI elements increase engagement and task completion, while complex interactions reduce adoption [16].

B. Architecture and Workflow

Below, we present the architecture of **PRIVSCAN**. As illustrated in Fig. 2, the tool is composed of five components. On the client side, two frontend modules, Screenshot Helper and Result Viewer are responsible for capturing the GUI screenshots and displaying CPPs, respectively. On the backend side, an API Client handles interactions the remote CPP generation service, and a central Controller orchestrates synchronization among all internal components. Together with the remote API Endpoint, these five components constitute the **PRIVSCAN** workflow.

Screenshot Helper. The Screenshot Helper captures the current user interface of the application. This screenshot serves as one of the two inputs of **PRIVSCAN**, alongside the privacy policy URL. To eliminate irrelevant visual noise, this module excludes all non-content elements from the screenshot, such as system GUI components, e.g., status bars, and interface elements of **PRIVSCAN**, e.g., floating buttons. A Bitmap containing an interference-free screenshot of the user interface is created and passed to the Controller and eventually transmitted to the remote server as input to generate CPPs.

Result Viewer. The Result Viewer module displays generated CPPs in a manner that is visually coherent and minimally disruptive to the original app interface. The results are rendered as annotated images, each derived from a screenshot of the current interface and leveraging a bounding box to enclose the GUI element that has been identified as related to a specific data type. To avoid overwhelming the user's perception by displaying multiple CPPs on the screen, we create one image per data type and display only one image to the user at a time. When there are elements involving different

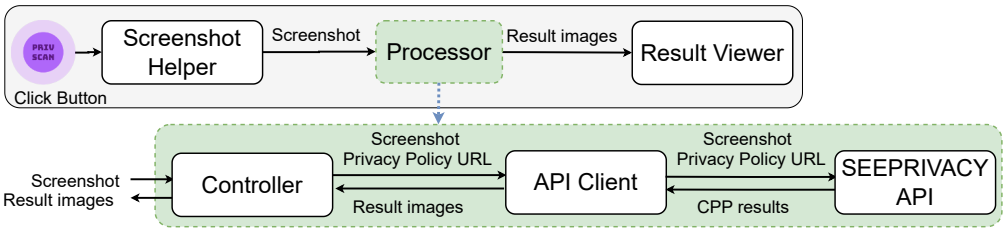


Fig. 2: An overview of **PRIVSCAN** SDK.

data on an interface, users can view the CPP of each data type by swiping left or right. We designed a structure-aware heuristic that adaptively positions CPP boxes, balancing visual association and layout constraints. Instead of fixed placement, the algorithm first extracts all vertical boundaries of annotated icons and interface edges, and computes the gaps between adjacent boundaries. Treating each gap as a candidate region, it selects the tallest gap to minimize overlap and visual clutter. This process yields a consistent, context-aware display without static templates. To enhance the readability of CPPs, we leverage a large language model API, e.g., GPT-4o mini [17], to summarize the original text extracted from the privacy policy to avoid displaying lengthy text directly on the screen. The currently displayed image can be saved to the device's media storage when the user presses the save button. The user can exit the viewer at any time by tapping the close icon.

Controller. The controller module coordinates other modules in the workflow. Upon initialization, the controller renders a draggable floating button on the screen, allowing users to trigger the CPP generation process within the application. The button's position is dynamically adjustable by users, and its size and color can be customized. The controller reacts to a valid click event by inserting a visual mask into the root layout as a full-screen *FrameLayout* with a semitransparent background, temporarily obscuring the underlying interface. At the center of the overlay, a circular *ProgressBar* is presented to inform the user that **PRIVSCAN** is currently working. Subsequently, the controller initiates interaction with the Screenshot Helper module to capture the current interface. Once the Screenshot Helper returns a *Bitmap* object representing the captured screen, the controller compresses it into a byte array for network transmission. This byte array, along with the policy URL, serve as inputs required by the CPP generation API. They are submitted through an asynchronous HTTP request dispatched by the API Client to the remote service.

If GUI elements related to data collection are detected on the interface, the controller invokes the result viewer module to render the generated CPPs. Otherwise, a *Toast* message is displayed to notify the user that no element is identified as related to potential privacy data collection. In addition to browsing CPPs, users also have easy access to the application's full privacy policy through a jump button placed in the lower right corner. In summary, the controller serves as the entry point and central coordinator of the interaction workflow.

API Client. This module mediates data transactions between the controller and the cloud-hosted **SEEPRIVACY** framework that generates the CPPs. It serves as the primary conduit for uploading inputs and receiving results. This module takes two key inputs: (1) the UI screenshot generated by the Screenshot Helper, and (2) the associated privacy policy URL. To initiate the request, the module submits an asynchronous task, constructing a *multipart/form-data* payload and dispatching it to the **SEEPRIVACY** FastAPI server. Once the server completes processing, it returns images with annotated CPPs. Each image corresponds to a data type category identified in the screenshot. To accommodate performance and memory constraints on mobile devices, the module employs a two-stage decoding strategy. First, it measures the image dimensions and computes an appropriate scaling factor, e.g., limiting the maximum image dimension to 2000 pixels. The module then performs actual decoding to construct a *Bitmap* object of the image. After decoding, the module delivers the result images to the controller.

SEEPRIVACY API. The API serves as the backend pipeline for generating CPPs. To support flexible deployment and seamless integration with the mobile SDK, the API is packaged as a standard web service and deployed on the Google Cloud Run platform [18]. Our goal is to decouple the image and text processing pipeline from local execution and expose it as a scalable HTTP interface, ensuring consistency while significantly reducing the migration and deployment costs. We present the API implementation in this paper, and more details about the backend pipeline are provided in our previous work [14]. The pipeline is containerized using Docker [19]. Built upon the *FastAPI* framework [20], the pipeline takes a GUI screenshot and the privacy policy URL as input, which are either retrieved from a caching layer or fetched anew and saved as an HTML file for reuse, reducing response time. This caching strategy also mitigates the risk of anti-bot defenses such as IP blocking triggered by excessive access.

In summary, through the coordination of the five components, **PRIVSCAN** integrates contextual privacy policies into practical mobile application scenarios with high modularity.

III. EVALUATION

To assess the execution efficiency of the CPP generation pipeline, we conducted an evaluation focusing on the end-to-end elapsed time of three key components: Context Detection, Segment Extraction, and CPP Presentation component. For more detailed descriptions of the pipeline components, as well as the performance and human evaluations, please refer to our previous work [14].

TABLE I: Execution time per component under three input screenshots: (i) privacy-data–related icons only, (ii) privacy-data–related text only, (iii) mixed icons and text, and the average time cost in seconds.

Component	Icon Only (s)	Text Only (s)	Mixed (s)	Average (s)
Context Detection	3.55	2.92	3.20	3.22
Segment Extraction	0.63	0.79	0.64	0.69
CPP Presentation	4.48	5.77	4.59	4.95
Overall	9.04	9.74	8.68	9.15

We implemented a dummy Android application that integrates our SDK to emulate realistic mobile usage scenarios and invoke the CPP generation pipeline. The dummy app can be run on a Samsung Galaxy S21 5G with 8GB RAM, running Android 13. We used the GPT-4omini [17] to generate a dummy privacy policy for this app. The app simulates user interactions through privacy-related GUI elements. To cover a variety of interfaces, we designed four application pages: (1) a home page containing both text and icons related to location data collection; (2) a posting page containing icons for profile, device camera, and album; (3) a settings page containing text related to user account; and (4) a rewards page that does not contain any element related to personal data collection. We used first three pages to measure the execution time for each component in CPP generation pipelines: Context Detection, Segment Extraction, and CPP Presentation component.

Table I summarizes the execution times of each component under the three input conditions. Each value is the mean of three runs per screenshot. The overall end-to-end execution time ranges from 8.68 s to 9.74 s, indicating that the tool remains within the acceptable latency range of 8-12 s for complex task in the real world [21]. Notably, the Context Detection and Segment Extraction components remain relatively stable across different screenshots, contributing 3.22 s and 0.69 s on average, respectively. Context Detection processes the screenshot to identify sensitive UI elements utilizing computer vision techniques, and Segment Extraction extracts the related segments from the privacy policy. A more detailed time measurement of each phase in these components are available at https://github.com/buyanghc/PrivScan. CPP Presentation generates the results and shortens the policy segments using LLMs [17]. It accounts for the largest share of the total execution time, with a peak of 5.77 s under the text-only condition. We attribute this partly to cumulative network variance of the LLM API used to shorten policy text, which reflects real usage, where multiple calls can introduce additional delay. The overall execution time is slightly larger than the sum of the three components as small glue operations between modules were not measured.

Overall, the findings highlight that interface characteristics affect performance. Future optimizations may focus on caching and multimodal prompt engineering to reduce the overhead of CPP generation. Nonetheless, the evaluation confirms that our SDK can run feasibly in mobile apps within the acceptable latency range.

IV. Conclusion

Concerns regarding data collection practices of mobile applications are growing, while privacy policies remain lengthy and disconnected from real-time usage contexts, making it difficult for users to understand how their data is collected and used. In this work, we extend the framework in our previous work [14] with a deployable SDK, **PrivScan**, which can be easily integrated into mobile apps and provides real-time, non-intrusive contextual privacy policies via a floating interactive button. We believe **PrivScan** illustrates the practicality of CPPs and is a step towards its broader real-world adoption.

References

[1] "General data protection regulation (GDPR)," https://eur-lex.europa.eu/eli/reg/2016/679/oj/eng, 2016, accessed: 2025-05-15.

[2] "California consumer privacy act of 2018," https://oag.ca.gov/privacy/ccpa, 2018, accessed: 2025-05-15.

[3] "Developer program policy." https://support.google.com/googleplay/android-developer/answer/16329168, Accessed: 2025-07-15.

[4] "App review guidelines." https://developer.apple.com/app-store/review/guidelines/, Accessed: 2025-07-15.

[5] I. Wagner, "Privacy policies across the ages: Content and readability of privacy policies 1996–2021," *arXiv preprint arXiv:2201.08739*, 2022.

[6] A. M. McDonald, R. W. Reeder, P. G. Kelley, and L. F. Cranor, "A comparative study of online privacy policies and formats," in *International Symposium on Privacy Enhancing Technologies Symposium*. Springer, 2009, pp. 37–55.

[7] Z. Tao, S. Pan, Z. Xing, X. Sun, O. Haggag, J. Grundy, J. Li, and L. Zhu, "Privacy bills of materials (pribom): A transparent privacy information inventory for collaborative privacy notice generation in mobile app development," *Proceedings on Privacy Enhancing Technologies*, 2025.

[8] Y. Shvartzshnaider, N. Apthorpe, N. Feamster, and H. Nissenbaum, "Going against the (appropriate) flow: A contextual integrity approach to privacy policy analysis," in *Proceedings of the AAAI Conference on Human Computation and Crowdsourcing*, vol. 7, 2019, pp. 162–170.

[9] "Requesting authorization to capture and save media." developer.apple.com/documentation/avfoundation/requesting-authorization-to-capture-and-save-media, Accessed: 2025-07-12.

[10] "Permissions on android." https://developer.android.com/guide/topics/permissions/overview, Accessed: 2025-07-12.

[11] T. Li, L. F. Cranor, Y. Agarwal, and J. I. Hong, "Matcha: An ide plugin for creating accurate privacy nutrition labels," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 2024.

[12] D. Feth, "Transparency through contextual privacy statements," in *Mensch und Computer 2017-Workshopband*. Gesellschaft für Informatik eV, 2017, pp. 10–18 420.

[13] M. Windl, N. Henze, A. Schmidt, and S. S. Feger, "Automating contextual privacy policies: Design and evaluation of a production tool for digital consumer privacy awareness," in *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, 2022.

[14] S. Pan, Z. Tao, T. Hoang, D. Zhang, T. Li, Z. Xing, X. Xu, M. Staples, T. Rakotoarivelo, and D. Lo, "A {NEW}{HOPE}: Contextual privacy policies for mobile applications and an approach toward automated generation," in *33rd USENIX Security Symposium (USENIX Security 24)*, 2024, pp. 5699–5716.

[15] A. Ignatov, R. Timofte, W. Chou, K. Wang, M. Wu, T. Hartley, and L. Van Gool, "Ai benchmark: Running deep neural networks on android smartphones," in *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*, 2018, pp. 0–0.

[16] P. Priyadarshini, "The impact of user interface design on user engagement," *International Journal of Engineering Research & Technology*, vol. 13, no. 3, 2024.

[17] "Gpt-4o mini." https://platform.openai.com/docs/models/gpt-4o-mini, Accessed: 2025-07-16.

[18] "Google cloud run." https://cloud.google.com/run?hl=en, Accessed: 2025-07-16.

[19] "Docker." https://www.docker.com/, Accessed: 2025-07-12.

[20] "Fastapi." https://fastapi.tiangolo.com/, Accessed: 2025-07-16.

[21] B. Shneiderman, *Designing the user interface: strategies for effective human-computer interaction*. Pearson Education India, 2010.