Lessons Learned from Developing Emotion Recognition System for Everyday Life

Stanisław Saganowski, Jan Miszczyk, Dominika Kunc, Dzmitry Lisouski, Przemysław Kazienko
stanislaw.saganowski@pwr.edu.pl
Department of Artificial Intelligence, Faculty of Information and Communication Technology, Wrocław University of Science and Technology
Wrocław, Poland

ABSTRACT

Recognizing emotions in everyday life requires a user-friendly and reliable system based on a smartphone and wearables. For over a year, we have been developing the Emognition system, which enables collecting emotionally annotated physiological signals in real-life scenarios. In this work, we describe the system architecture, the components and libraries used, as well as the development, testing, and implementation strategies. We explain in detail the integration with wearables – smartwatch Samsung Galaxy Watch 3 and chest strap Polar H10. The encountered problems and developed solutions are thoroughly discussed. We also provide the advantages and limitations of several frameworks for embedding machine learning models into a resource-restricted mobile application.

KEYWORDS

emotional data, emotionally annotated physiology, emognition system, wearables

ACM Reference Format:
Stanisław Saganowski, Jan Miszczyk, Dominika Kunc, Dzmitry Lisouski, Przemysław Kazienko. 2022. Lessons Learned from Developing Emotion Recognition System for Everyday Life. In *SenSys '22: The 20th ACM Conference on Embedded Networked Sensor Systems, November 06–09, 2022, Boston, MA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3560905.3567759

1 INTRODUCTION

Emotion recognition from physiological signals has many important applications, e.g., help in emotion-based disorders [9], Autism [6], monitoring our well-being [7] and mental health [1], controlling stress [22], human-computer interaction [25], recommendation systems [2], and computer games [13].

The potential market-ready solutions require that machine learning (ML) models used to recognize emotions are precise and reliable. This implies that models should be trained on the data collected in daily life rather than in a carefully designed laboratory environment. Collecting data outside the lab is challenging because we have no control over the process [21]. It is unknown what stimuli will evoke emotions, how often people will experience emotions, when to trigger self-assessment, and how detailed it should be to accurately annotate samples (experienced emotions and related physiological signals). On top of that, self-assessment of emotions is always subjective and perceived differently by each of us.

Because of the problem's complexity, only a few systems for collecting emotionally annotated data in the field have been proposed. One of them is Happimeter [8], which utilizes consumer smartwatches to collect heart rate, self-assessments, and location data. When enough data is gathered, the system can provide mood predictions. The Happimeter consists of mobile (Android/iOS) and smartwatch (Wear OS/watchOS) applications, both of which are publicly available. The technical details of the system have not been revealed by the authors. Hernandez et al. reviewed recent emotion recognition commercial applications and complained that systems do not sufficiently explain their functionality to the users, often hiding the technical limitations, and also that there is too little data collected in everyday life to create reliable models [10].

To overcome some of these problems, we developed the Emognition system (see Fig. 1a), which can enhance the collection of emotionally annotated data in real-life scenarios. The main advantages of the Emognition system include: (1) a simple and intuitive, yet powerful mobile application; (2) integration with consumer wearables that provide access to the raw physiological signals; (3) utilizing convenient wearables – the chest-strap Polar H10 and the smartwatch Samsung Galaxy Watch 3 that is also useful to the users; (4) detecting intense emotions in real-time, utilizing ML model embedded into the mobile application; (5) user-friendly strategy for triggering self-assessments – not too often, unobtrusive notifications, short questionnaires; (6) multiple self-assessment triggers: based on the pre-trained model prediction, quasi-random, on-demand; (7) ability to replace the model and adjust many parameters remotely, without reinstalling the application.

The general idea of the system is explained in [5, 18], while the preliminary validation in everyday life scenario is reported in [11, 20]. A demo of the Emognition system is available online[1].

The main contribution of this work is a detailed description of the Emognition system, its architecture, components and libraries used, validation methods, as well as lessons learned from developing the system, especially encountered problems and implemented solutions.

SenSys '22, November 6–9, 2022, Boston, MA, USA

ACM ISBN 978-1-4503-9886-2/22/11…$15.00
https://doi.org/10.1145/3560905.3567759

[1] https://youtu.be/zgJw4krZ5tU

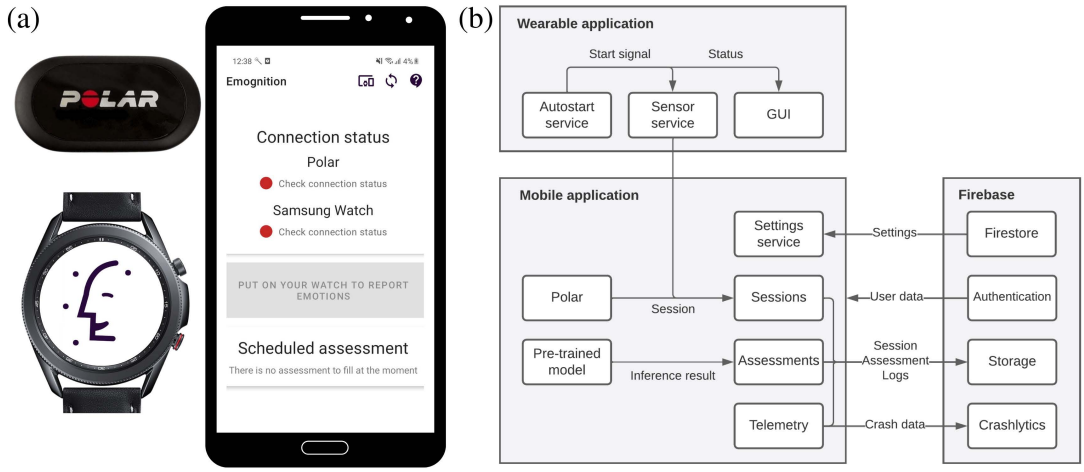

Figure 1: The Emognition system. (a) Devices compatible with the Emognition system, from left: Polar H10 chest strap, Samsung Galaxy Watch 3 with Emognition Tizen app, a smartphone with Emognition Android app. (b) High-level system architecture.

2 EMOGNITION SYSTEM

The Emognition system consists of three distinct components. The **mobile application** is the main component of the system. It is responsible for communicating with wearables, especially retrieving data. The mobile app processes the data in real-time and performs reasoning whether the user is experiencing intense emotions. If so, a self-assessment will be triggered. The collected data is transferred to the Firebase backend service. The **smartwatch application** is a hybrid Tizen app for Samsung Watch 3. It collects raw data from the device's sensors and stores them internally. Then, the data is transferred in real-time or synchronized later on. Finally, the **Firebase backend** is used for authorization, long-term data storage, and configuration of various aspects of the system, e.g., scheduling assessments. We also utilize the Crashlytics utility, which lets us monitor potential issues with the mobile app.

2.1 Mobile application

2.1.1 Architecture. The application consists of three main packages:

- **Application** – presentation layer. The UI is implemented using both legacy XML view definitions, and Jetpack Compose components. The presentation layer communicates with the business logic layer using asynchronous data streams (Kotlin Flows).
- **Infrastructure** – a framework for abstracting away key Android and third-party components. Provides streamlined APIs which simplify implementations in both presentation and business logic layers.
- **Model** – business logic layer defines how data is created and transformed. It utilizes APIs from the infrastructure layer to access the persistent data storage services (SQLite database, Cloud Storage, and Firestore).

Application resources, such as drawables or XML view definitions, are tied to flavor-specific source sets. Flavors themselves are described in the **Application flavors** section.

2.1.2 Session concept. The physiological data from different devices have to adhere to a shared *session* definition. A single session represents data collected from the start of signal recording until the event that terminates the session – in case of the smartwatch: battery runs out or device is placed on the charger; in case of the chest-strap: a device has to be manually disconnected from the mobile app. Ideally, there would only be one session per day, lasting from the morning to the evening. During the session's duration, raw data from sensors is saved to the corresponding session *chunks* (files). Once the configured size of data is reached, the files are closed, and a new chunk is created. We currently use two-minute chunks for all supported devices. Depending on the wearable functionality, session files can be created either on a mobile device (*streaming* approach) or on a wearable and synchronized later (*file transfer* approach). The streaming method offers immediate access to the raw sensor data, is easier to implement, and does not require any internal storage on the wearable device. Sometimes, it is the only method offered. From our experience, the file transfer is considerably more effective in real-life applications, as brief disconnection periods are very common in everyday life, e.g., when we go to a distant room or floor without a smartphone. The file transfer approach requires more effort to integrate into the system (data storage and synchronization logic) and delivers data to the smartphone after the file is saved, preventing data loss, which may be crucial for some use cases.

2.1.3 Assessments. The assessment module is a critical component primarily used for labeling physiological data by the participants. There are three possible triggers of a brief questionnaire described in detail in [18]:

- **Pre-trained model** – if the outcome of the reasoning on data from the last chunk suggests the user is experiencing an intense emotion, a self-assessment is triggered. This will temporarily suspend the real-time reasoning (we do not want to overload the user with more assessments, and to save battery life). The idle time can be adjusted in the settings.
- **Scheduler** service schedules assessments for a given day. Based on the settings (expected session length, the maximum number of assessments per day – *frequency*, and offset variation), surveys are planned in quasi-random intervals. Besides, they also may

serve as a validation mechanism of the pre-trained model. We can expect that responses from such planned assessments contain less high arousal states than from triggered by the mode.
- **On demand** – study participants can report their emotions manually anytime, provided a session is running.

2.1.4 Data synchronization. The assessment and session data can be synchronized with the Firebase backend automatically or manually. To facilitate such periodic work, we utilize Android's *WorkManager API* which allows us to schedule tasks using flexible time windows. Uploaded files can then be transferred to secure offline storage using our custom download tool. Once synchronized, session files are deleted from the user's device. Additionally, session file synchronization has certain time-related constraints to make sure that the pre-trained model can always process the latest data. Chunks can only be synchronized once their age exceeds a preconfigured threshold.

2.1.5 Settings. Application settings can be configured for each user through Firestore. The available settings can be used to customize (personalize) the parameters of assessments, assessment scheduler, and pre-trained model modules. New settings are fetched from the server automatically every time a new session is started. The application can also function offline with the latest copy of the custom configuration or built-in default values if no settings have been manually configured.

2.1.6 Telemetry. To gather information about the application health, we employed two solutions – Crashlytics and database synchronization. Crashlytics allows us to automatically collect real-time crash information from all users, provided they have an active Internet connection. Any emerging stability issue is then quickly forwarded to the development team for further investigation. The latter solution was developed to monitor the application's state and its critical components. Among others, it can be used to investigate whether certain features, such as pre-trained model or accessory connection, are working as expected. The SQLite database snapshot synchronization can be triggered automatically or manually by the user. The file is sent to Firebase Storage, from which it can be retrieved by developers or data analysts using a custom downloading tool. The database contains information about sessions, assessments, file transfers, and logs which provide an insight into system performance and operation.

2.1.7 Application flavors. Product flavors represent different versions of the application. Each of these flavors can have its own features while sharing common source code and resources. To simplify the user interface and hide redundant information from study participants, we decided to implement two variants of the app, *internal* and *end-user*. The internal version provides access to additional information such as file transfer log or remote settings configuration. As our internal team often uses it to validate various pre-trained model implementations, it has been configured to run inference sessions continuously. On the other hand, the end-user version is the production version of our app – meant to be used by the study participants. It features a simplified user interface and a different set of default configuration values.

2.1.8 Background operation mechanisms. To make sure that our application can work in the background without being interrupted by the operating system, we employed two main solutions. The Android's *AlarmManager* service is utilized for tasks that have to be handled at a precise moment in the future (e.g., triggering scheduled assessments). The *WorkManager* API is used for both deferrable and immediate tasks such as data synchronization or pre-trained model operation. The latter also utilizes a *foreground service*, which is run by the *WorkManager* itself. This feature allows us to execute long-running operations, as is the case with, e.g., real-time inferencing or collecting data from a Polar H10 chest strap. The study participant can also monitor the current state of running background modules through a non-dismissible notification.

2.2 Smartwatch integration

We integrated the Samsung Watch 3 into the Emognition system because it is more stable and battery efficient than Wear OS alternatives. The overview of other possible wearables is provided in [19]. Using the Samsung Watch 3 sensors, we continuously collect the following data: (1) the reflected LED light (the raw photoplethysmogram – PPG) sampled at 25 Hz; (2) heart rate and PP interval, both sampled at 12.5 Hz; (3) 3-axis accelerometer data (ACC) sampled at 50 Hz; (4) 3-axis gyroscope at 50 Hz, (5) 4-axis rotation at 50 Hz; (6) pressure at 1 Hz; and (7) ambient light at 1 Hz.

2.2.1 Architecture. The smartwatch application consists of three projects (depicted in Fig. 1b):
- **GUI** (web application) – application frontend;
- **Autostart service** (native service) – responsible for starting and terminating the sensor service when the charging state is changed. Provides sensor service status to the GUI application;
- **Sensor service** (native service) – collects data from sensors and handles bilateral communication with the mobile application.

Each module is built separately and packaged as a single hybrid application.

2.2.2 Custom build process. Due to countless issues with *Tizen Studio* (the official Tizen development IDE based on Eclipse), we decided to migrate to *Visual Studio Code* as our primary development platform. This migration required the implementation of a custom build pipeline based on *Tizen CLI* and *PowerShell*. The build process is automated by using tasks that call functions defined inside a custom *PowerShell* script. This script internally uses the *Tizen CLI* interface to perform actions such as installing or launching the app.

2.2.3 Communication. We utilize the Samsung Accessory Protocol (SAP) to facilitate bilateral communication with the Android app. The SAP consists of multiple independent modules which can be used to send messages or transfer files between devices. Communication is held through a Bluetooth connection. Both apps use a predefined set of messages, which are transferred as JSON strings. They can be used to, e.g., retrieve information about available sessions or send synchronization requests for specific *chunks*. In legacy app versions, the communication tunnel was established through a service connection, which is a continuous communication channel. Due to problems with sudden disconnects (most likely due to the bandwidth being exhausted by file transfers), we recently moved

to a connection-less solution called *SAMessage*, which significantly improved the connection resiliency.

2.2.4 Battery consumption. During typical use of the Samsung Watch 3, the battery lasts for about 2-3 days. However, the smartwatch resources are heavily exploited due to continuous data collection from multiple sensors (reading and storing data) and communication with the mobile app. This results in significantly shorter battery life on a single charge. When using the Emognition system, the version equipped with a 340 mAh battery (Samsung Watch 3 45mm) lasts for about 12-14 hours, while the smaller version (41mm, 247 mAh) for 8-10 hours. In fact, due to the high battery consumption, our smartwatch application could not be published to the Galaxy Store, so we have to install it manually through the Tizen CLI. The possible solutions to extend battery life include reducing the sampling frequency or the number of sensors used or collecting data for shorter periods throughout the day. This can be adjusted depending on the use case.

2.3 Chest strap integration

The second device which is integrated into the Emognition system is the Polar H10 chest strap. The device provides: (1) electrocardiogram (ECG) sampled at 130 Hz; (2) heart rate and RR interval, both with variable sampling; and (3) 3-axis ACC at 25 Hz, 50 Hz (used by us), 100 Hz or 200 Hz.

The integration process began with the creation of a client on the mobile app end that communicates directly with the chest strap. It includes operations to connect with and collect data from the Polar H10 device.

The biggest downside of the Polar H10 device is lack of the internal storage[2], which means the device has to be connected to the smartphone all the time to stream the signals. If we lose connection, even for a brief moment (e.g., we move to the other office/floor without the smartphone), the data falling within this period are irretrievably lost. The Polar SDK has the ability to automatically connect the chest strap to a paired smartphone. However, it requires that the mobile app is working in the foreground, which is not always the case. To overcome this, we developed a custom auto-connect service that initiates the connection as soon as the paired device is within the Bluetooth range.

The process of obtaining data is initialized by a custom worker, created using the *WorkManager* API. When the connection to the Polar H10 device is lost, the worker terminates the entire process. When conditions are restored, the process is automatically resumed. We also needed to implement a service that can check whether data streams are available. The service is associated with the *ChunkWriter* class, which is responsible for creating *chunks* and writing to them. When a *chunk* contains data from a predefined period of time (in our case two minutes), the *ChunkWriter* closes the corresponding files and starts writing to a new *chunk*.

The Polar H10 device delivers sensor data in batches, which for the ECG signal usually contain around 73 measurements and are delivered approximately every 560 ms. Each batch is assigned a single timestamp which, according to the documentation, is the timestamp of the last measurement. The timestamp is represented in nanoseconds since 2000-01-01 00:00:00 (Zulu time zone). We estimate timestamps of the remaining measurements based on the sampling frequency.

One of the known problems with the Polar SDK is unexpected disconnections during data streaming[3]. In our case, it occurs once in several dozen sessions, and our work-around is to reconnect the device automatically. Equally often, we observe another problem – the Polar H10 stops broadcasting the HR data. Despite several ideas, we were unable to find a fully working solution. Hence, we calculate HR from the ECG signal. To connect with the Polar H10, the mobile app requires location rights, including location in the background.

2.4 Integration with other wearables

The modular architecture of the Emognition system allows for fairly easy integration with new wearables. We plan to develop a common interface to make the implementation even more convenient.

2.4.1 Aidmed chest strap. For instance, we integrated Aidmed wearable [4], which is an ECG chest strap similar to the Polar H10. The Aidmed offers ECG (sampled at 512 Hz) and ACC (50 Hz) signals, skin temperature (0.1 Hz), as well as breathing-related data such as bioimpedance, airflow through the nose/mouth (measured with pressure sensor), and microphone recordings.

Apart from the medical certification and more data, the Aidmed's most significant advantage over Polar H10 is its internal memory, allowing to store sessions on-device (64 MB of memory should store a couple of hours of ECG, ACC, and HR data depending on the sampling frequency). If the connection with the phone is lost, the Aidmed begins to store data internally. Once the connection is restored, Aidmed can synchronize with the phone data saved locally.

On the other hand, the Polar H10 is smaller and runs on a battery (instead of an accumulator) that can last for about 400 hours (vs. about 16 hours in the case of the Aidmed).

The implementation of the Aidmed is very similar to the Polar H10. We also have a client class to work directly with the device, an auto-connect service, and a data collection service that is connected to the *ChunkWriter*.

2.4.2 Wear OS-based smartwatches. The number of smartwatches running Wear OS is growing rapidly. Even Samsung moved from its own Tizen OS to Wear OS since Samsung Galaxy Watch 4. The Wear OS has many advantages for the developers: (1) more convenient development of applications due to the Android SDK availability and modern developing tools – Kotlin and/or Java vs. C and/or C# in Tizen and Android Studio; (2) better integrity and straightforward port of functions between Android app on a smartphone and Wear OS app on a smartwatch; (3) the Wear OS logging is easily manageable and can be used even in the production version whereas the logging mechanism in Tizen works only during development (smartwatch has to be connected with the development tool); (4) the smartwatch app distribution through the Google Play is more smooth and quicker than through the Galaxy Store.

Unfortunately, the Wear OS-based smartwatches still have several critical issues: (1) very short battery life – only six hours in the

[2]The Polar H10 can store internally only one session (up to 30h) containing only HR data sampled with 1 Hz.

[3]https://github.com/polarofficial/polar-ble-sdk/issues/222

case of the Fossil Gen 5; (2) unstable BT connection between smartphone and smartwatch; (3) the manufacturers implement sensors functionality themselves, hence the documentation of sensors is often very limited; (4) the manufacturers can impose access restrictions to some sensors/data, e.g., in the case of Samsung Watch 4 developers are required to register to the Partner App Program to gain access to the Samsung Privileged Health SDK.

Due to the aforementioned requirement, we were unable to integrate Samsung Watch 4 with the Emognition system. We did, however, successfully integrate Fossil Gen 5. It provides the raw PPG with a 20 Hz sampling rate, ACC, gyroscope data, altitude, ambient light, GPS, and other data. Due to the short battery life and unstable connection, we do not plan to include the Fossil Gen 5 in our studies. We hope that the future Wear OS devices will be free from the critical issues mentioned above. Until then, in our opinion, the Samsung Watch 3 with Tizen OS is the best choice for studies aiming to collect raw physiological signals.

2.5 Cloud backend

We utilize the Firebase cloud solution as the backend of the Emognition system because it facilitates, among others: straightforward implementation in the mobile app, user authentication and authorization, data storage, and tracking mobile app stability. The Firebase Authentication module supports multiple authentication methods. We decided to use anonymized login and password, both generated by the Emognition system. We use the Storage module to persist user-related data, such as physiological data (sessions), self-assessments, and application telemetry (logs). The Firestore module is useful for storing and adjusting settings and parameters used across the Emognition system. The Crashlytics module helps in collecting, managing, and analyzing application crashes and crash reports. It allows us to diagnose and troubleshoot application issues. For each application flavor and build type (i.e., *debug* or *release*) we use a separate Firebase project.

3 ML MODEL EMBEDDING

One of the vital features of the Emognition system is the ability to recognize a user's intense emotions. To achieve this, a pre-trained machine learning model embedded into the mobile application performs reasoning based on continuously incoming chunks of physiological data. To perform real-time reasoning on the smartphone instead of the typical cloud-based architecture, we had to overcome two challenges: (1) preprocessing data locally on the smartphone, using Kotlin-compatible or Java libraries, so it can be used in the classification task, and (2) embedding/converting the pre-trained model into the mobile application code.

Our typical pipeline to perform emotion recognition includes signal processing, feature extraction, model training, model validation, and classification. For research purposes, we use a pipeline developed in the Python environment. It utilizes the most popular ML libraries: NumPy [14], SciPy [24], Sklearn [23], TensorFlow [26], and many other used to extract features from signals. Only some of the methods/libraries used for data preprocessing could be embedded into the mobile app unaltered. For others, we had to rewrite them in Kotlin or find alternatives. In a few cases (e.g., signal resampling), we had to make some simplifications.

Regarding the ML model embedding/conversion, we found the following solutions: PyTorch Mobile [17], TensorFlow Lite[27], sklearn-porter[12], and ONNX Runtime[16]. See Tab. 1 for their features and limitations. We evaluated all of them except the PyTorch Mobile. Most frameworks work exclusively with their related Python library. Only the ONNX Runtime enables the conversion of models created with different tools to a common .ort format. All frameworks except the sklearn-porter enable model optimization and provide prediction probability (besides providing the predicted label). The most significant disadvantage of the sklearn-porter is the necessity to include the translated model source code in the mobile application during compilation. This makes it impossible to remotely switch the model, as the mobile app has to be rebuilt and updated. Other frameworks enable easy remote model replacement (in our case, implemented using cloud storage). In addition, all frameworks, except sklearn-porter, feature the ability to embed model-specific metadata in converted files, simplifying the reasoning and model storage processes, as all important information and settings are available together with the model itself.

In our opinion, the ONNX Runtime is the best solution available on the market today. Its implementation in the Emognition system allowed us to significantly simplify the logic and source code. Additionally, model personalization is as easy as specifying the model ID and its parameters (e.g., confidence) in the user-related settings. The only drawback of the ONNX Runtime we have identified so far is the complex integration process which involves building the library from the source. It is more complicated than other frameworks we evaluated.

4 DEVELOPMENT FLOW

For version control, we utilize Git along with Github – a cloud-based Git hosting service. Github offers many additional collaboration features such as issue tracking, automated workflow support, and task management utilities.

Pull request flow. We configured branch protection rules to simplify the development process and protect our repository against accidental and unreviewed pushes to critical development branches. These rules mandate that any change to the source code has to go through a separate feature branch, which is then compared to the main development branch through a pull request (PR). Proposed changes must be verified by another developer through code review.

Quality assurance. Almost every PR represents a certain feature or bug fix that the QA team can test. All PRs follow a standard description template which consists of a summary of changes and testing instructions. Every approved code review will cause a new application artifact to be created, built, and uploaded. The QA team then uses these artifacts to perform manual tests according to the provided testing instructions.

CI/CD pipelines. The repositories feature multiple automated workflows, which are triggered when certain actions are performed. Currently, supported workflows are described in Tab. 2.

5 DISCUSSION AND LIMITATIONS

During the development of the Emognition system, we encountered and solved many problems. The design of this type of system should be based primarily on the use case, which determines further

Table 1: Machine Learning frameworks for models conversion; (*) denotes frameworks tested in our system.

Feature	PyTorch Mobile	TensorFlow Lite*	sklearn-porter*	ONNX Runtime*
Python library	PyTorch	TensorFlow	Scikit-learn	PyTorch, TensorFlow, Scikit-learn
ML model	Any	Any	Some	Any
ML model optimization	+	+	−	+
Prediction probability	+	+	−	+
Metadata	+	+	−	+
Remote model swap	+	+	−	+
Available Mobile APIs	Android, iOS	Android, iOS	None, only converted model code	Android, iOS

Table 2: CI/CD workflows implemented in our repository

Name	Trigger	Description
Continuous integration	On push	Builds the application and runs unit tests. Uploads test results as artifacts on completion.
Create testing artifacts	On approved code review	Builds the application in *debug* or *release* configuration depending on the target branch. Artifacts are uploaded to Github for use by the QA team.
Create release	On *master* push	Creates a Github release, and in the case of the Android app, builds both application variants and submits them to the Google Play store.

choices and decisions. For instance, which physiological signals are required. From there, wearables that offer these signals and meet other criteria (integration with the system, size, battery life, usability for the user, price, etc.) will be identified [19]. In our case, the smartwatch is more comfortable and has additional functions for the user, but the PPG collected during motion is of poor quality. On the other hand, the chest strap gives a more accurate ECG but is uncomfortable and does not offer any features to the user.

It is good to decide early on whether we need to perform real-time reasoning and embed the ML model into the mobile application or if the reasoning in the cloud is acceptable. Furthermore, do we need to collect data continuously, or perhaps it is enough to obtain basic physiological data such as HR periodically?

We should also take into account possible external issues – bugs in third-party SDKs (e.g., known Polar H10 problems, unstable SAP connection), insufficient device and SDK documentation, as well as OS restrictions. With each new version of the Android OS, the restrictions on running applications in the background, data transfer, and excessive battery consumption are becoming increasingly stringent. At some point, it might be necessary to develop a custom Android OS (custom ROM), which grants all the required permissions to the application and does not put the app to sleep. Our current solution requires the user to add the Emognition mobile app (and related apps such as Galaxy Wearable and Samsung Accessory Protocol) to the list of exceptions that are not optimized.

5.1 System validation

The system validation can be divided into two parts: technical validation and system usability validation.

First, it is important to test the application in a thorough and systematic manner by performing unit tests, integrity tests, and manual tests. Moreover, it is critical to always double-check the final version/setup of the system in the staging environment, as there might be some differences in the compilation and release processes between the test and production versions.

In our system, we utilize all of the aforementioned testing methods. Our repositories feature CI workflows that automate the process of compiling and running unit tests for each commit that is pushed to a remote branch. Integration tests are performed manually before deploying a new application version. These could not be automated because they require access to a physical device. Manual testing is also a vital part of our development workflow – the QA team validates each PR according to the provided testing instructions. They also verify key elements and features of the system before a candidate release is pushed to the production.

To test the usability of the system, we conducted a pilot study in everyday life scenario. The 13 participants (six females) took part in a three-month-long trial. In this trial, only the smartwatches were utilized (the chest strap was not used). The participants were prompt with a self-assessment up to six times per day. Some questionnaires were triggered randomly, and some were based on the ML model. We managed to collect over 1200 self-reports, out of which almost 500 reported intense emotion. The questionnaires triggered by the ML model reported intense emotion more often than those triggered at random. The personalized ML model was up to 38% better in catching an intense emotion moment than the random triggering. For more details please refer to [11, 20].

5.2 Users' feedback

The users' feedback should always be gathered to improve the system. Therefore, after conducting the above-mentioned pilot study [11, 20], we asked the participants to fill out the post-study questionnaire regarding their opinion on the system. Nine out of 13 participants completed the questionnaire. In general, all respondents felt comfortable using the system and found it to be convenient. The system did not cause fatigue or irritation. Two participants reported that the study improved their perception of emotions, but the majority said they did not see a difference.

The reported problems included: (1) forgetting to use the system and/or to wear the smartwatch; (2) forgetting to fill out the questionnaires; (3) filling out the questionnaire was not possible in some situations, e.g., while driving a car; (4) the need to manually start the smartwatch application after smartwatch restart; (5) the system drains the smartwatch and the smartphone batteries; (6) notifications about the app working in the background and about new assessment to fill out were similar, thus confusing.

Most of the problems will be addressed by us, but some are difficult to tackle, e.g., it is fairly easy to recognize that someone is traveling by car, but it is much harder to distinguish whether they are drivers or passengers.

5.3 Ethical considerations

When conducting an affective study, especially one that utilizes wearables, we should consider factors that can negatively impact the participants and/or the study itself and prepare the research accordingly to the identified risks. Particularly important are issues related to collecting, processing, storing, and sharing personal and biological information, anonymization, research-related negative emotions, commercial technology validity and reliability, and participants' exclusivity issues. A comprehensive discussion on risks in affective studies can be found in [3, 15].

6 CONCLUSIONS AND FUTURE WORK

We described the crucial concepts behind the Emognition system, provided details about its components and discussed its limitations. Overall, the system is well suited for everyday life studies. Thanks to the embedded ML model detecting intense emotions, the system increases the amount of emotionally annotated data. What is more, in combination with the smartwatch, the system is unobtrusive to the study participants. The modularity of the system facilitates easy extension and integration of new wearables.

When developing such a system, it is important to automate as many processes as possible, especially those performed iteratively, e.g., testing, releasing, and updating the application as well as collecting, storing, and backing up the data. It is also crucial to develop monitoring mechanisms that will alert the study manager if a problem with the system or particular participant occurs. The system should be tested on each development step, i.e., prototype, beta version, and released version. Additionally, several trial tests should be performed to validate the intended goals. After each trial and study iteration, the feedback from the participants should be taken into account.

We hope that lessons learned from developing the Emognition system will contribute to the affective studies, especially those performed in real-life scenarios.

Our next steps include further personalization of ML models, i.e., (re)training models with samples collected from a particular user, as well as obtaining more data from the smartphone to better understand the context of the reported emotional situations.

ACKNOWLEDGMENTS

This work was partially supported by National Science Centre, Poland, project no. 2020/37/B/ST6/03806; by the statutory funds of the Department of Artificial Intelligence, Wroclaw University of Science and Technology; by the Polish Ministry of Education and Science, National Information Processing Institute — the CLARIN-PL Project.

REFERENCES

[1] Amani Albraikan, Basim Hafidh, and Abdulmotaleb El Saddik. 2018. iAware: A real-time emotional biofeedback system based on physiological signals. *IEEE Access* 6 (2018), 78780–78789.
[2] Pedro Álvarez, Francisco Javier Zarazaga-Soria, and Sandra Baldassarri. 2020. Mobile music recommendations for runners based on location and emotions: the dj-running system. *Pervasive and Mobile Computing* 67 (2020), 101242.
[3] Maciej Behnke, Stanislaw Saganowski, Dominika Kunc, and Przemyslaw Kazienko. 2022. Ethical Considerations on Using Wearables in Affective Research. *IEEE Transactions on Affective Computing* (2022).
[4] Łukasz Czekaj, Jakub Domaszewicz, Łukasz Radziński, Andrzej Jarynowski, Robert Kitłowski, and Anna Doboszyńska. 2020. Validation and usability of AIDMED-telemedical system for cardiological and pulmonary diseases. *E-methodology* 7, 7 (2020), 125–139.
[5] Maciej Dzieżyc, Joanna Komoszyńska, Stanisław Saganowski, Magda Boruch, Jakub Dziwiński, Katarzyna Jabłońska, Dominika Kunc, and Przemysław Kazienko. 2021. How to catch them all? enhanced data collection for emotion recognition in the field. In *2021 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*. IEEE, 348–351.
[6] Huanghao Feng, Hosein M Golshan, and Mohammad H Mahoor. 2018. A wavelet-based approach to emotion classification using EDA signals. *Expert Systems with Applications* 112 (2018), 77–86.
[7] Luz Fernández-Aguilar, Arturo Martínez-Rodrigo, José Moncho-Bogani, Antonio Fernández-Caballero, and José Miguel Latorre. 2019. Emotion detection in aging adults through continuous monitoring of electro-dermal activity and heart-rate variability. In *International Work-Conference on the Interplay Between Natural and Artificial Computation*. Springer, 252–261.
[8] Peter A Gloor, Andrea Fronzetti Colladon, Francesca Grippa, Pascal Budner, and Joscha Eirich. 2018. Aristotle said "happiness is a state of activity"—Predicting mood through body sensing with Smartwatches. *Journal of Systems Science and Systems Engineering* 27, 5 (2018), 586–612.
[9] Cheng He, Yun-jin Yao, and Xue-song Ye. 2017. An emotion recognition system based on physiological signals obtained by wearable sensors. In *Wearable sensors and robots*. Springer, 15–25.
[10] Javier Hernandez, Josh Lovejoy, Daniel McDuff, Jina Suh, Tim O'Brien, Arathi Sethumadhavan, Gretchen Greene, Rosalind Picard, and Mary Czerwinski. 2021. Guidelines for Assessing and Minimizing Risks of Emotion Recognition Applications. In *2021 9th International Conference on Affective Computing and Intelligent Interaction (ACII)*. IEEE, 1–8.
[11] Dominika Kunc, Joanna Komoszyńska, Bartosz Perz, Przemysław Kazienko, and Stanisław Saganowski. 2022. Real-Life Validation of Emotion Detection System with Wearables. In *International Work-Conference on the Interplay Between Natural and Artificial Computation*. Springer, 45–54.
[12] Darius Morawiec. [n.d.]. sklearn-porter. https://github.com/nok/sklearn-porter. Transpile trained scikit-learn estimators to C, Java, JavaScript and others.
[13] Grzegorz J Nalepa, Krzysztof Kutt, Barbara Giżycka, Paweł Jemioło, and Szymon Bobek. 2019. Analysis and use of the emotional context with wearable devices for games and intelligent assistants. *Sensors* 19, 11 (2019), 2509.
[14] NumPy [n.d.]. NumPy. https://numpy.org. [Online; accessed 23-February-2022].
[15] Desmond C Ong. 2021. An ethical framework for guiding the development of affectively-aware artificial intelligence. In *2021 9th International Conference on Affective Computing and Intelligent Interaction (ACII)*. IEEE, 1–8.
[16] ONNX Runtime [n.d.]. ONNX Runtime. "https://onnxruntime.ai". "[Online; accessed 23-February-2022]".

[17] PyTorch Mobile [n.d.]. PyTorch Mobile. https://pytorch.org/mobile/home/. [Online; accessed 23-Feb-2022].
[18] Stanisław Saganowski, Maciej Behnke, Joanna Komoszyńska, Dominika Kunc, Bartosz Perz, and Przemysław Kazienko. 2021. A system for collecting emotionally annotated physiological signals in daily life using wearables. In *2021 9th International Conference on Affective Computing and Intelligent Interaction Workshops and Demos (ACIIW)*. IEEE, 1–3.
[19] Stanislaw Saganowski, Przemyslaw Kazienko, Maciej Dziezyc, Patrycja Jakimów, Joanna Komoszynska, Weronika Michalska, Anna Dutkowiak, Adam Polak, Adam Dziadek, and Michal Ujma. 2020. Consumer wearables and affective computing for wellbeing support. In *MobiQuitous 2020-17th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*. 482–487.
[20] Stanisław Saganowski, Dominika Kunc, Bartosz Perz, Joanna Komoszyńska, Maciej Behnke, and Przemysław Kazienko. 2022. The cold start problem and per-group personalization in real-life emotion recognition with wearables. In *2022 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*. IEEE, 812–817.
[21] Stanislaw Saganowski, Bartosz Perz, Adam Polak, and Przemyslaw Kazienko. 2022. Emotion Recognition for Everyday Life Using Physiological Signals from Wearables: A Systematic Literature Review. *IEEE Transactions on Affective Computing* (2022).
[22] Philip Schmidt, Attila Reiss, Robert Duerichen, Claus Marberger, and Kristof Van Laerhoven. 2018. Introducing wesad, a multimodal dataset for wearable stress and affect detection. In *Proceedings of the 20th ACM international conference on multimodal interaction*. 400–408.
[23] Scikit-Learn [n.d.]. Scikit-Learn. https://scikit-learn.org/stable/. [Online; accessed 23-February-2022].
[24] SciPy [n.d.]. SciPy. https://scipy.org. [Online; accessed 23-February-2022].
[25] Feri Setiawan, Sunder Ali Khowaja, Aria Ghora Prabono, Bernardo Nugroho Yahya, and Seok-Lyong Lee. 2018. A framework for real time emotion recognition based on human ans using pervasive device. In *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*, Vol. 1. IEEE, 805–806.
[26] TensorFlow [n.d.]. TensorFlow. https://www.tensorflow.org. [Online; accessed 23-February-2022].
[27] TensorFlow Lite [n.d.]. TensorFlow Lite. https://www.tensorflow.org/lite. [Online; accessed 23-Feb-2022].