Estimating Developers' Cognitive Load at a Fine-grained Level Using Eye-tracking Measures

Amine Abbad-Andaloussi*
amine.abbad-andaloussi@unisg.ch
Institute of Computer Science,
University of St. Gallen
St Gallen, Switzerland

Thierry Sorg
thierry.sorg@unisg.ch
Institute of Computer Science,
University of St. Gallen
St Gallen, Switzerland

Barbara Weber
barbara.weber@unisg.ch
Institute of Computer Science,
University of St. Gallen
St Gallen, Switzerland

ABSTRACT

The comprehension of source code is a task inherent to many software development activities. Code change, code review and debugging are examples of these activities that depend heavily on developers' understanding of the source code. This ability is threatened when developers' cognitive load approaches the limits of their working memory, which in turn affects their understanding and makes them more prone to errors. Measures capturing humans' behavior and changes in their physiological state have been proposed in a number of studies to investigate developers' cognitive load. However, the majority of the existing approaches operate at a coarse-grained task level estimating the difficulty of the source code as a whole. Hence, they cannot be used to pinpoint the mentally demanding parts of it. We address this limitation in this paper through a non-intrusive approach based on eye-tracking. We collect users' behavioral and physiological features while they are engaging with source code and train a set of machine learning models to estimate the mentally demanding parts of code. The evaluation of our models returns F1, recall, accuracy and precision scores up to 85.65%, 84.25%, 86.24% and 88.61%, respectively, when estimating the mental demanding fragments of code. Our approach enables a fine-grained analysis of cognitive load and allows identifying the parts challenging the comprehension of source code. Such an approach provides the means to test new hypotheses addressing the characteristics of specific parts within the source code and paves the road for novel techniques for code review and adaptive e-learning.

CCS CONCEPTS

• **Software and its engineering**; • **Social and professional topics**; • **Human-centered computing**;

KEYWORDS

Program comprehension, source code, cognitive load, eye-tracking, machine learning

*The first author is supported by the International Postdoctoral Fellowship (IPF) Grant (Number: 1031574) from the University of St. Gallen, Switzerland

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
ICPC '22, May 16–17, 2022, Virtual Event, USA
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9298-3/22/05…$15.00
https://doi.org/10.1145/3524610.3527890

ACM Reference Format:
Amine Abbad-Andaloussi, Thierry Sorg, and Barbara Weber. 2022. Estimating Developers' Cognitive Load at a Fine-grained Level Using Eye-tracking Measures. In *30th International Conference on Program Comprehension (ICPC '22), May 16–17, 2022, Virtual Event, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3524610.3527890

1 INTRODUCTION

Software is everywhere! All over the world, millions of lines of code are running in our computers, devices and vehicles to support our daily activities and facilitate almost all aspects of our lives. The development of software systems is prone to faults, which can have significant impacts on humans' life and economy (e.g., [2, 8, 28–30, 41]). With the widespread use of software systems and the speed at which software requirements change and evolve overtime, there is an increasing need to maintain these systems while ensuring that the applied changes are free of faults.

Code comprehension plays an important role in this vein as typical software maintenance activities like code change, code review and debugging depend heavily on developers' understanding of the code. This ability is threatened when the mental effort required to comprehend the code exceeds developers' mental capacity [6, 46]. As a result, developers experience very high levels of cognitive load [38], which, in turn, manifests in increased difficulty, poor understanding of the source code and higher risk of errors.

There is a considerable body of literature that has made attempts to analyze developers' cognitive load when engaging with source code during code comprehension (overview in [15, 16, 34, 36, 42, 43, 50]). However, in the majority of these studies, cognitive load was investigated only at a coarse-grained task level, providing global estimates of the difficulty of the source code as a whole, without pinpointing the mentally demanding parts of it.

In this paper, we propose an approach to pinpoint the mentally demanding parts of code using eye-tracking when reading source code. We answer the following research questions: ***RQ1. Can measures based on eye-tracking be used in estimating the mentally demanding lines and fragments of source code?*** and ***RQ2. Which classes of eye-tracking measures contribute the most to the estimation of the mentally demanding parts of code?***. We base our approach on machine learning (ML) models trained with features derived from *fixation*, *saccade*, *pupil*, *scan-path* and *cluster-based-AOI* characteristics measured at a fine-grained level. We evaluate the performance of our ML models when estimating the mentally demanding parts of code at both line and fragment levels. Therein, we follow a cross-validation approach estimating the mentally demanding parts of code for new participants, new tasks and new participants conducting new tasks. The results

demonstrate high performance in identifying the difficult parts of code in all these scenarios. Notably, we could identify the mentally demanding fragments with a reliability of up to **F1: 85.65%, recall: 84.25%, accuracy 86.24%, precision: 88.61%**. Moreover, we investigate the importance of each class of features and demonstrate that pupil and saccadic features are the most important when estimating the mentally demanding lines and fragments of code respectively.

The outcome of this work is expected to have implications on research, practice and education. Notably, as our approach allows collecting and analyzing cognitive load at a fine-grained level, it can be applied in the context of experiments where researchers are interested in investigating the impact of different parts within the same source code on readers' cognitive load. Moreover, with further development, our approach can be adapted in practical settings to infer the parts of code susceptible to contain quality issues and guide reviewers towards them during code review tasks. Furthermore, the approach can be embedded in adaptive e-learning systems to identify the challenging parts in learning materials and adjust to the learning pace of each individual student. Sect. 2 elaborates the requirements for a fine-grained estimation of the mentally demanding parts of code and provides an overview on how these requirements can be fulfilled. Sect. 3 presents the background and related work. Sect. 4 describes our research method. Sect. 5 presents and discusses the findings. Sect. 6 describes the threats to validity. Finally, Sect. 7 concludes the paper and gives an overview of the future work.

2 APPROACH REQUIREMENTS AND OVERVIEW

To be able to pinpoint the mentally demanding parts of code at a fine-grained level (code lines or fragments), a series of requirements should be fulfilled. First and foremost, (R1) to infer developers' mental effort while reading source code, it is necessary to have continuous measurements of cognitive load, which can be derived from behavioral measures capturing changes in humans voluntary behavior [6] (e.g., eye-movements captured with eye-tracking) and physiological measures capturing variations in the human autonomic nervous system [6, 50] (e.g., pupillary reactions from eye-tracking, galvanic skin response (GSR), heart-rate variability (HRV), electroencephalogram (EEG), functional magnetic resonance imaging (fMRI) and functional near-infrared spectroscopy (fNIRS)) (*overview in* [15, 16, 34, 36, 42, 43, 50]). Secondly, since our goal is to pinpoint difficult parts of code, it is not sufficient to conduct these measurements at a task level (e.g., for an entire source code snippet). Instead, (R2) an approach is required to scale down the continuous measurement of cognitive load to capture it at a fine-grained (spatial) level (i.e., lines and fragments). Third, (R3) the continuous cognitive load measurements should be based on lightweight, easy to carry and use sensors which can be adopted in day-to-day settings. Fourth, (R4) the approach should be applicable to (large) software systems and not only small source code snippets or static stimuli (with a fixed position on the screen). This entails that the approach should deal with scrolling and switching between different source code files. Fifth, (R5) the approach should be generalizable and work as well across different developers and/or source code reading tasks.

These requirements form the basis for our approach (explained in Sect. 4). This approach is based on eye-tracking (using a non-intrusive eye tracker) and provides a continuous measure of cognitive load (addressing R1). Moreover, it is lightweight and thus easier to use and more likely to be accepted in practical settings than EEG, fMRI, fNIRS which rely on sensors requiring a close contact with the human body (addressing R3). With respect to R2, we leverage eye-tracking to establish a mapping between the fixated lines and columns of code with developers' behavioral and physiological measures. Following the eye-mind hypothesis [23], we use (eye-tracking) fixations as an intermediary step in this mapping (see Sect. 4.1.4), associating fixations with internal mental processing and cognitive load. In a second step, we contextualize fixations with respect to the behavioral and physiological measures captured in their temporal vicinity (see Sect. 4.2.1). When it comes to R4, to develop an approach that can operate on both small and large source code, we automate the mapping of fixations' coordinates with the underlying lines and columns of code, allowing, in turn, to present source code (of any size) within the integrated development environments (IDE) and to support scrolling and file navigation, while maintaining the mapping between fixation coordinates and source code lines and columns consistent. Finally, for R5, we validate our approach with respect to its ability to operate across developers and different tasks.

3 BACKGROUND AND RELATED WORK

This section introduces the related concepts and literature. Sect. 3.1 presents the cognitive theory. Sect. 3.2 provides a background on the behavioral and physiological measures based on eye-tracking. These measures were used as a basis to derive the features introduced in Sect. 4.2.1. Sect. 3.3 gives an overview of the state-of-art literature investigating cognitive load at a fine-grained level.

3.1 Cognitive Load Theory

The *cognitive load theory* (CLT) investigates the characteristics of the human's *working memory* (i.e., a buffer storing and processes information for short time spans [45, 55]). The theory defines the concept of cognitive load as a *"a multi-dimensional construct representing the load imposed on the working memory during* [the] *performance of a cognitive task"*[6]. In addition, the theory emphasizes the limitation of the working memory, positing that it can hold a small number of items at a time (7±2items) [35]. When the limits of the working memory are approached, users are likely to experience *cognitive overload* which can negatively affect their performance and lead them to wrong decisions [6].

3.2 Behavioral and Physiological Measures Based on Eye-tracking

3.2.1 Fixation Measures. A fixation refers to the time the eyes stay fixed at a position of the stimulus [21]. Based on the eye-mind hypothesis [23], measures such as *fixation duration* and *fixation count* have been used to capture long and recurrent fixations as indicators of cognitive load [21]. In software engineering, fixation measures were adopted in a number of studies investigating developers' visual effort when engaging with source code (overview in [15, 16, 34, 36, 42]). In past studies, fixations have also been used

to discern shallow information scanning (i.e., effortless) from deep mental processing (i.e., mentally demanding). According to [14, 48], the former is exhibited in short fixations with a duration under 250 ms, while the latter manifests in long fixations exceeding 500ms.

3.2.2 Saccade Measures. A saccade refers to the rapid eye-movements occurring between consecutive fixations [21]. The amplitude (i.e., distance) of saccades tends to decrease as a response to high levels of cognitive load [24, 32]. This phenomenon is referred to in the literature as the tunnel vision effect, which manifests to prevent memory overload when the amount of information (to be processed) within the visual field exceeds the limits of the working memory [31, 32]. Consequently, saccades occur within a narrowed visual field which in turn is reflected in a short amplitude. Following the literature review presented in [42], *saccadic amplitude* was not yet used in a software engineering context. The authors of the review have, nevertheless, introduced the measure and highlighted its possible applications in empirical software engineering studies. *Saccadic direction* is another relevant measure used to infer the predominant direction of saccades within a particular area of the artifact [21]. This measure is typically used in scene viewing studies [21], but has been also recently referred to in empirical software engineering studies. In [7], the authors suggested that the navigation between parts of code (namely, "re-visits of code sections") depicts users' strategies of reading and may relate to cognitive load, which supports the relationship with the saccadic direction.

3.2.3 Pupil Measures. Pupil dilation is a physiological response to increased cognitive load caused by the activation of the sympathetic division of the human autonomic system [47, 50]. Increases in *pupil size* [19, 27], *pupil peak count* [44], and *pupil peak amplitude* [3] have been associated with task difficulty and cognitive load in a number of studies [3, 19, 27, 47]. In software engineering, pupil-based measures have been used in several studies (overview in [15, 16, 34, 50]). Notably, in [13], pupil size was used as a feature of an ML model providing estimates of task difficulty.

3.2.4 Scan-path Measures. A scan-path denotes a sequence of consecutive fixations (or visits to areas of interest – AOIs [21]) and saccades (or transitions between AOIs) recorded during a certain period of time [21]. The analysis of scan-paths reflects the complexity of the visual search, i.e., the process of identifying a target item among a set of distractors [21, 52]. A short and simple scan-path indicates an easy and directed search, while a long and complex one reflects a rather challenging search [21]. The complexity of visual search can be associated with the CLT. During a task involving visual search (e.g., source code comprehension), humans' working memory is used to store the intermediate information leading to a particular target item [40]. When this item is difficult to identify and extract, readers are required to store an increased amount of information in their working memory to keep track of the search progress and the potentially relevant information allowing to locate the target item. Due to the limited capacity of the human working memory, this increase of information intake can raise readers' cognitive load. In the literature, several measures have been proposed to evaluate the complexity of visual search through scan-path properties such as *length*, *density* and *entropy* [21]. These measures are either derived from the scan-path itself, which can be seen as a graph, or the underlying transition matrix [21]. Typically they are extracted at the task level [21], but some studies have also extracted these measures at a more fine-grained level (i.e., fixed time windows) [22]. In software engineering, users' scan-paths have been investigated in several studies covering both source code and conceptual models (overview in [36, 42, 42]).

3.2.5 Cluster-based AOI Measures. Cluster-based AOIs are generated using clustering algorithms, dividing a given set of data samples into subsets (i.e., clusters, or AOIs) with similar properties [21]. Spatial proximity is among the common features used to generate cluster-based AOIs [21]. This class of AOIs has been used in several domains [21]. Notably, in software engineering, cluster-based AOIs derived from a set of spatial and temporal (i.e., spatial-temporal) gaze features have been associated with parts of code where users have been challenged [7]. However, they have not yet been emphasized by the existing literature reviews.

3.3 Fine-grained Estimation of Cognitive load

The difficulty of code comprehension tasks has been related to several factors including, for instance, the complexity of the control and data flows encoded in the source code [39], the presence of anti-patterns [10] and the existence of atoms of confusion [17, 53]. Behavioral and physiological measures of cognitive load [6] are among the measures used to investigate such a relationship. These measures have been adopted in a wide array of studies (overview in [15, 16, 34, 36, 42, 43, 50]). Although their results were promising (e.g., [13]), the majority of the used approaches cannot be adopted to pinpoint the mentally demanding parts of code. To address this need, some studies have attempted to deliver fine-grained source code annotations hinting towards the mentally demanding parts of code. However, none of them cover all the requirements for developing such an approach (cf. Sect. 2). These studies are discussed in this section.

The majority of the empirical studies using behavioral measures based on eye-tracking (overview in [34, 36, 42, 43]) represents source code as a static stimulus (i.e., *not satisfying* R4). This setting does not support scrolling and switching between source code files, which does not reflect the real-world use of source code as developers usually engage with large code bases within IDEs supporting these basic features among a wide set of others. "iTrace" was proposed to address this limitation by automating the mapping between developers' fixations and the underlying lines and columns of the source code when eye-tracking data is being collected [18]. This in turn, allows to present source code within the IDE and supports scrolling and file navigation. However, the aims of the studies using iTrace to analyze users' cognitive load using behavioral measures at a fine-grained level [10, 25] differ from ours. For instance, in [10], the authors compared fixation duration on source code identifiers in the presence and absence of lexical anti-patterns to infer the cognitive load associated with each of these two conditions, while in [26], fixation-based and AOI-based measures [21] were collected a fine-grained level, but their interpretation was used to estimate developers' challenges at the task level.

Physiological measures have been used to estimate cognitive load at a fine-grained level in a wider array of studies (cf. overview in [15, 16, 50]). The authors in [7], adopted a clustering approach

based on spatial-temporal features to identify cluser-based AOIs that are likely to reflect cognitive load. The regions identified by the clusters were visually compared with plots of HRV and pupillography to derive qualitative insights suggesting a relationship between these measures. Being exploratory, the generalizability of the reported insights remains to be demonstrated (i.e., *not satisfying* R5). Our approach differs from [7] as it follows a quantitative analysis based on ML and uses a cross-validation method demonstrating its generalizability across developers and different tasks. The authors in [20] investigated the ability of pupillography, HRV and eye-tracking to infer the challenging part of code, while in [10], the authors adopted fNIRS and eye-tracking for the same purpose. Similar to our setting (cf. Sect. 4.1.2), these two studies collect highlights of the mentally demanding parts of code from the participants and use them as a ground truth against which the performance of ML and statistical models is compared. Likewise, in [33], the authors investigated whether EEG and eye-tracking can differentiate the mentally demanding parts of code. The approaches in [10] and [33] use fNIRS and EEG measures that are challenging to collect in workplaces (i.e., *not satisfying* R3). As for the approach in [20], it relies on HRV features requiring long time windows (above a minute) to make reliable estimates of cognitive load [4]. With such an approach, it is not possible to scale down the measurement of cognitive load to a fine-grained (spatial) level. This is because in a time interval of minute, the user would have visited too many lines and fragments of code, making it difficult to contextualize the cognitive associated with each of them (i.e., *not satisfying* R2).

4 RESEARCH METHOD

This section describes the method used to collect and analyze the data. To enable a continuous measurement of cognitive load (i.e., R1, cf. Sect. 2) using devices that can be adopted in day-to-day settings (i.e., R3) we have based our method on eye-tracking. Section 4.1 describes the data collection procedure, while Section 4.2 presents the data analysis approach.

4.1 Data Collection

4.1.1 Subjects. The data collection covered 16 participants. Following our demographic survey (cf. Sect. 4.1.3), the recruited participants were 6 researchers, 6 PhD students, 3 Bachelor/Master students and 1 IT administrator. All the participants had a background in programming and software engineering (13 intermediates, 2 experts, 1 novice). Moreover, all the participants knew object-oriented programming and thus had the necessary knowledge to take part in the experiment. Although we collected data from all the 16 participants, the data for one participant was excluded during the analysis due to a poor calibration of the eye-tracking device.

4.1.2 Experiment Tasks. The experiment included 9 comprehension tasks, each comprising a source code in Java and a comprehension question. The tasks were inspired by those in [13, 39]. However, since the authors in [13, 39] investigated users' cognitive load at a task level, they used snippets that were either simple or complex as a whole. In our study, we investigate cognitive load at a fine-grained level. To collect data supporting such an analysis, we had to adapt the tasks in the literature [13, 39] to comprise both easy and difficult parts. The simple parts contained variable declarations and simple variable assignments, while the complex ones covered loops (normal and recursive) together with if-else conditions and several arithmetic operations. The used source code snippets had a size varying between 25 and 53 lines of code (i.e., average 34 lines of code). Scrolling was needed in the IDE to read the entire code (i.e., R4). As for the questions, the participants were asked to mentally execute the given source code and derive the final output. In some tasks, the participants were given multiple-choice questions (i.e., selecting a correct execution trace among a set of traces), while in other tasks, they were asked inference questions (i.e., inferring the full execution trace from the source code).

The experiment included also a set of code-highlighting tasks. Therein, the participants were asked to highlight the mentally demanding parts of code (e.g., tokens, lines, fragments of code). These tasks were conducted both during the experiment (i.e., following each comprehension task) and by the end of the experiment as part of a retrospective think-aloud.

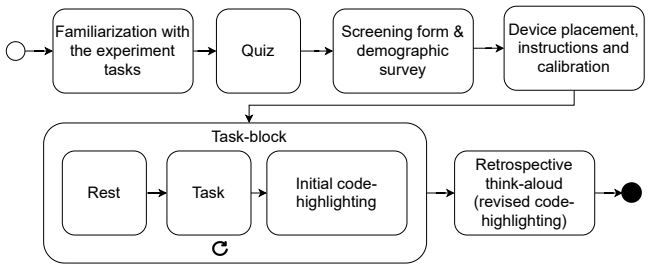

Figure 1: Experiment procedure

4.1.3 Experiment Procedure. The experiment was conducted in a controlled lab environment. All data was collected with informed consent and participants were allowed to withdraw at any stage of the data collection. Figure 1 illustrates the procedure followed in the individual sessions. Each participant was invited to a tutorial session where he/she was familiarized with the patterns used in the java source code (e.g., control-flow structures such as normal/recursive loops, if-else statement and arithmetic operations such as modulo) and the types of experiment questions (multiple-choice, inference questions). Following that, the participant was given a quiz comprising two tasks sharing the same characteristics as those used in the experiment. Afterward, screening and demographic forms were administered to collect general information about his/her experience in programming and software engineering. The quiz and the forms served as basis to assess the participant's ability to take part in the experiment.

Prior to the (main) data collection, the participant was seated in front of the eye-tracker and instructed following existing guidelines to minimize head movements and avoid any kind of external distractions [21]. Moreover, the illumination in the lab was controlled to ensure no optical artifacts or pupillary reactions due to variations in the light intensity [21]. As a last step, the eye-tracking device was calibrated and the mapping between gaze points and the screen coordinates was verified. During the data collection, the participant was exposed to a series of task-blocks displayed in a randomized order. Each block comprised a rest screen, a source code

comprehension task and a code-highlighting task (cf. Sect. 4.1.2). By the end of the data collection, the participant was invited to a retrospective think-aloud session. Therein, for each comprehension task, the participant was shown a (1) heatmap of his/her own gazes on the underlying source code and (2) a copy of his/her *initial* code-highlights. Based on the eye-mind hypothesis (cf. Sect. 3.2.1), the "dark" areas of heatmaps are likely to reflect parts of code where the participants have been challenged. Accordingly, the participant was asked to compare the gaze heatmaps with the initial highlights and double-check that the highlighted parts of code correspond to those that have been mentally demanding during the experiment. The participant was allowed to update the initial highlights. However, the *revised* highlights were saved in a different file, allowing us to use both initial and revised highlights in our analysis (cf. Sect. 4.2.2). The use of heatmaps in retrospective think-aloud is a common practice allowing participants to recall their behavior during the task and reflect on it [10, 21]. During this procedure, the participant was also asked to verbalize his/her thoughts and explain the reason for highlighting different parts of code.

4.1.4 Instrumentation. To support a continuous measurement of cognitive load at a fine-grained (spatial) level (i.e., R2, cf. Sect. 2), we had to collect eye-tracking data where fixations coordinates are mapped with the fixated lines and columns of code. As mentioned in Sect. 2, this constitutes the first step of our mapping between the fixated lines and columns of code and developers' behavioral and physiological measures through fixations. Moreover, to ensure the applicability of our approach to both small and large code bases (i.e., R4), such a mapping had to be automated. We incorporated these needs in the instrumentation of our experiment as follows. We used a Tobii X3-120 eye tracker to capture participants' gazes on the screen. The gaze data was forwarded to two devices: iTrace [18] and iMotions (Version 8.2[1]). We used both iTrace and iMotions to leverage their strengths. In particular, iTrace was used to automatically map the forwarded gaze data with the lines and columns gazed by the participant when reading the source code in Eclipse IDE. iMotions, in turn, was used to design and run the protocol of the experiment as well as to record pupil data and derive fixations (using the I-VT algorithm [37]) from the forwarded gaze data. Moreover, the tool allowed us to monitor developers' eye-movements and keep a constant eye on the data quality during the data collection. Furthermore, since we have collected Galvanic Skin Response (GSR) data (which will be analyzed in a follow-up study), we used iMotions to handle the synchronization between the GSR and the eye-tracking data. The data collected by iTrace and iMotions was linked using a common identifier attributed to gaze points in both iTrace and iMotions logs. As a result, we derived an enriched dataset joining the data from both logs. In this dataset, the fixations (derived from iMotions) were mapped to the fixated lines and columns of code (derived from iTrace).

4.2 Data Analysis

We followed a predictive analysis approach based on supervised ML [1]. Sect. 4.2.1 introduces the derived features (i.e., independent variables). Sect. 4.2.2 describes the labels used as outcome measures (i.e., dependent variables). Finally, Sect. 4.2.3 explains the adopted model training and validation methods.

4.2.1 Feature Extraction. Our feature extraction approach was based on a set of measures derived from fixation, saccade, pupil, scan-path and cluster-based AOI characteristics (overview in Tables 1 and 2). At first, we *contextualized each fixation* with respect to these measures (cf. Sect. 4.2.1.1), then we derived a set of features at the *fixation*, *line* and *fragment* levels of granularity (cf. Sect. 4.2.1.2).

4.2.1.1 Contextualization of Fixations. Supporting a measurement of cognitive load at a fine-grained (spatial) level (i.e., R2, cf. Sect. 2) can be addressed using the mapping approach of which the first step consists of mapping fixations coordinates with the underlying lines and columns of code (cf. Sect. 4.1.4). The second step, in turn, consists of contextualizing fixations with respect to the behavioral and physiological measures captured in their temporal vicinity. As mentioned in Sect. 2, this contextualization is motivated by the eye-mind hypothesis associating fixations with mental processing and cognitive load [23].

Figure 2 illustrates the proposed contextualization approach. The first contextualization of fixations was with respect to the preceding saccade (cf. Fig. 2 (a)). As mentioned in Sect. 3.2.2, saccadic properties associated with their amplitude and direction can provide insights about users' cognitive load when engaging with source code. Accordingly, each fixation was decorated with a number of additional attributes capturing such properties from the saccade preceding it (cf. Table 1).

The second contextualization was with respect to the pupil signal concurring with the individual fixations (cf. Fig. 2 (b)). As mentioned in Sect. 3.2.3, this signal can help to infer pupillary reactions associated with cognitive load. We cleaned the pupil signal using the pipeline proposed in [56]. The pipeline detects and removes blink artifacts, removes outliers, interpolates missing values and applies a third-order lowpass Butterworth filter (cf. [56] for a detailed explanation of this procedure). We used a windowing approach to extract the characteristics of the pupil signal. We opted for this approach (instead of directly using the pupil size reading at the fixation timestamp) following the existing literature showing that pupillary reactions in response to cognitive load occur within a certain time interval (i.e., hundreds of milliseconds) preceding or following a stimulating event [9]. After empirically testing different window lengths (up to 2500ms) occurring before or after fixations at different time intervals (from -500ms to 500ms), we opted for a 700ms window capturing the pupil signal occurring 400ms after the fixation onset. Thereafter, fixations were decorated with a set of additional attributes denoting the features of the associated pupil signal recorded within a [400ms, 1100ms] window after each fixation onset (cf. Table 1). These features were derived from the pupil size, the pupil peaks and their amplitude following the insights reported about these measures in Sect. 3.2.3.

The third contextualization was with respect to the (sub) scan-path observed within the fixation vicinity (cf. Fig. 2 (c)). We investigated these sub-scans following the conceptual bases provided in Sect. 3.2.4, linking the complexity of visual search (observed through scan-paths) with users' cognitive load. We started by generating a global scan-path denoting the sequence of lines of code visited during the entire task. This sequence was represented as a

[1]See https://imotions.com/release/

directed graph whose nodes refer to the visited lines of code and whose edges refer to the transitions between these lines. The edges were assigned weights referring to the number transitions between the related lines. Afterward, we divided the global scan-path into a set of sub-scans with equal duration. Here again, we have empirically tested several window durations (up to 15 seconds) and finally opted for a 2-seconds time window. Finally, we decorated each fixation within the aforementioned time window with a set of attributes capturing the characteristics (i.e., length, density, entropy) of the associated sub-scan (cf. Table 1). These characteristics have been associated with the complexity of visual search [21] and thus can tell about users' cognitive load as explained in Sect. 3.2.4

The fourth contextualization was with respect to the position of fixations in space and time (cf. Fig. 2 (d)). As mentioned in Sect. 3.2.5, fixations with a spatial and temporal proximity were used to generate cluster-based AOIs, which in turn, have been associated with mentally demanding parts of code [7]. Accordingly, we have grouped fixations following two clustering approaches. The former was a spatial clustering based on the fixations' line and column coordinates, while the latter was a spatial-temporal clustering based on the fixations' coordinates and timestamp. The clustering approaches were implemented using the density-based clustering algorithm of the sklearn library[2]. Like for the other contextualizations, the parameters were empirically derived. In the spatial clustering, the clusters were constrained to cover only single lines of code, have a maximum distance of 3 columns between two samples, and contain a minimum of 4 samples per cluster. As for the spatial-temporal clustering, the clusters were constrained to cover only single lines of code, be separated with a maximum of 800ms and contain at least 2 samples per cluster. Thereafter, each fixation was decorated with a set of attributes specifying whether it belongs to a spatial or a spatial-temporal cluster (cf. Table 1). These attributes would, in turn, infer whether the fixation is within a part of code that is likely to be mentally demanding.

4.2.1.2 Extraction of Features at Different Levels of Granularity. The aim of this work is to develop an ML-based approach using users' behavioral and physiological features to infer the mentally demanding part of code at two levels of granularity: i.e., line and fragment of code. To this end, following the contextualization of fixations, we have extracted a set of features at the fixation level, then used them as basis to derive other features at the line and fragment levels. At the fixation level, we extracted the fixation, saccadic, pupil, (sub) scan-path and cluster-based AOI features described in Table 1. At the line of code level, we have aggregated the features collected at the fixation level (cf. Table 1) using the count, sum, max, min, mean, median and standard deviation (std) functions. In addition, we have extracted the new features described in Table 2. Lastly, at the fragment of code level, we have aggregated both the fixation-level and line-level features shown in Tables 1 and 2 using the aforementioned aggregation functions.

4.2.2 Labeling. We investigated two types of labels. The first type indicates whether a line of code was mentally demanding, while the second type indicates whether a fragment of code was mentally demanding. The first label type was obtained directly from the participants' highlights (e.g., a specific line within a control-flow structure), while the second label type was derived by inferring the fragment to which the highlighted line belongs (e.g., the control-flow structure). We have additionally considered the latter type of labels as the retrospective think-aloud revealed that some participants highlighted specific tokens (e.g., the "else" keywords in an if-else statement) with the assumption that the entire part of code associated with these tokens was mentally demanding (e.g., the content of the "else" block). The two types of labels were extracted for both the *initial* and the *revised highlights* (cf. Sect. 4.1.3).

As explained in Sect. 4.2.1, a set of features was extracted and aggregated at the line and fragment of code levels. Together with the aforementioned labels, we derived four datasets comprising respectively: features computed at the *line* level and labels obtained from the *initial* highlights *(LI)*, features computed at the *fragment* level and labels derived the *initial* highlights *(FI)*, features computed at the *line* level and labels obtained from the *revised* highlights *(LR)* and features computed at the *fragment* level and labels derived from the *revised* highlights *(FR)*.

4.2.3 Model Training and Validation. The derived datasets (i.e., LI, FI, LR and FR) were used to train a set of ML models following the approach depicted in Figure 3. Since users highlighted only a few parts of code within each source code file, the classes of highlighted and non-highlighted lines/fragments were not equally represented. This problem is referred to in the literature as *imbalanced classification* [11]. To mitigate a potential bias towards the over-represented class (i.e., non-highlighted lines/fragments), we used a *random under sampling* strategy in the training of the ML models. Therein, we selected a random subset from the over-represented class with a size matching the one of the under-represented class (cf. Fig. 3 (a)). Then, we generated a new data-subset where both classes are equally represented. We opted for feature selection (based on the mutual information algorithm[3]) to identify the top 20 most prominent features in the data-subset (cf. Fig. 3 (b)). These features were used in the training of an ML model, i.e, a decision tree classifier (cf. Fig. 3 (c)). Our feature selection threshold (i.e. 20 most prominent features) has been chosen to enable a fast training of the ML model and make the resulting decision tree easier to interpret. After repeating this random under sampling approach over 10 iterations, we embedded the resulting models into an ensemble, i.e., a voting classifier[4] (cf. Fig. 3 (d)). At prediction time, the input data was fed to all the ML models (of the ensemble), each returning its own estimate. The final estimation was then designated as the common estimation among the set of returned estimates.

We have chosen decision trees as they allow to derive insights on the importance of each feature used for the classification. Accordingly, we used the feature importance metric implemented in the sklearn library[5] to assign an importance score to each feature. This metric is based on the Mean Decrease Impurity (MDI) defined as "the total decrease in node impurity [...] averaged over

[2]See https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html

[3]See https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.mutual_info_classif.html#sklearn.feature_selection.mutual_info_classif

[4]See https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html

[5]See https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier.feature_importances_

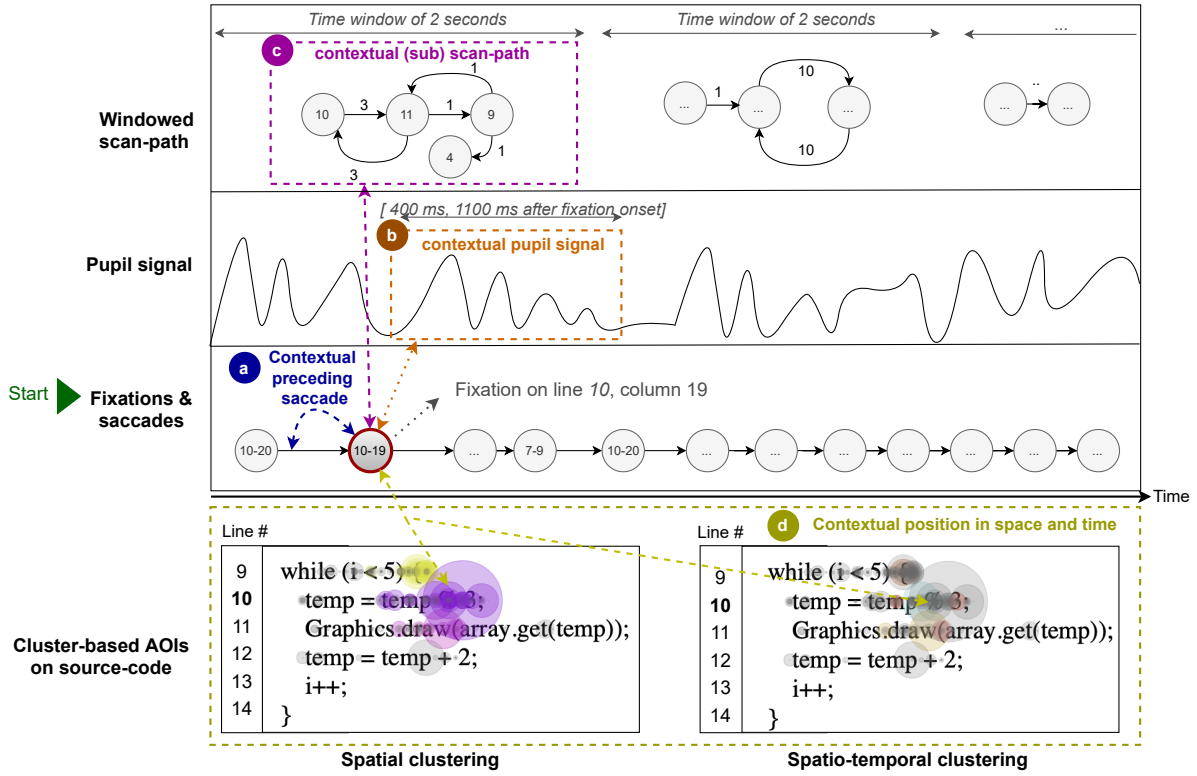

Figure 2: Contextualization of fixations

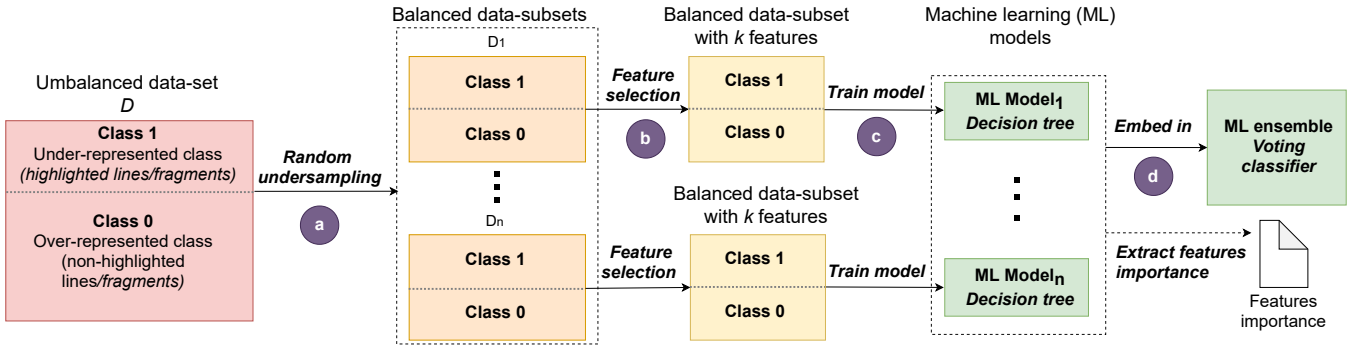

Figure 3: Machine learning approach.

all trees of the ensemble" [49]. Finally, we grouped the features by type (fixation-based, saccade-based, pupil-based, scan-path-based, cluster-based AOI) and aggregated their feature importance scores using the sum function. The final feature importance scores for each feature group are shown in Table 4.

To demonstrate the generalizability of our approach across developers and different source code reading tasks (i.e., R5, cf. Sect. 2), we crossed-validated our ML models using different leave-one-out-strategies (similar to [13]). Herein, three exhaustive sets of test and training folds were respectively generated by (1) leaving one participant out, (2) one task out and (3) one pair of participant-task out. In (1), the data of all participants except one is used in the training of the model, while the data of the remaining participant is used to test the model. A similar approach is used in (2) and (3) while considering the data corresponding to different tasks or different pairs of participants-tasks. This cross-validation was repeated exhaustively to cover all individual participants, tasks and pairs of participants-tasks. The performance of the models trained and tested under these conditions reflect respectively their ability to predict the mentally demanding lines and fragments of code for new participants, new tasks and new participants performing

Category	Feature	Description
Fixation	Fixation duration	Duration of the fixation in milliseconds
	Fixation duration *fix.>= 250ms*	*Expression:* fixation duration **if** fixation duration >= 250*ms* **else** NaN
	Fixation duration *fix.>= 500ms*	*Expression:* fixation duration **if** fixation duration >= 500*ms* **else** NaN
Saccade	Saccadic amplitude	Euclidean distance to the previous fixation in the (code) line and column coordinates
	No saccade	*Expression:* 1 **if** source (i.e., previous) fixation is on the same line and column **else** 0
	Horizontal	*Expression:* 1 **if** source fixation is on the same line and different column **else** 0
	Non-horizontal	*Expression:* 1 **if** source fixation is on a different line **else** 0
Pupil	Pupil size *{aggr.}*	Pupil size in mm collected over the specified time window and aggregated using *{count, sum, max, min, mean, median, std}*
	Number of pupil peaks	Number of pupil peaks observed in the specified time window
	Amplitude of pupil peaks *{aggr.}*	Amplitude of the individual pupil peaks observed in the specified time window and aggregated using *{count, sum, max, min, mean, median, std}*
(sub) Scan-path	Sub-scan length – # transitions	Number of transitions between lines of code in the sub-scan derived over the specified time window. In our graph-based representation, this corresponds to the number of edges multiplied by their weights
	Sub-scan length – # unique visits	Number of unique lines of code visited in the sub-scan derived over the specified time window. In our graph-based representation, this corresponds to the number of nodes in the graph
	Sub-scan density	(# transitions /# unique line visits)
	Sub-scan entropy	Entropy of the sub-scan transition matrix. In our graph-based representation, this corresponds to the entropy of the graph adjacency matrix
Cluster-based AOIs	Is fix. in spatial cluster	*Expression:* 1 **if** fixation is in spatial cluster **else** 0
	Is fix. in spatial-temporal cluster	*Expression:* 1 **if** fixation is in spatial-temporal cluster **else** 0

Table 1: Features extracted at the fixation level. Abbreviations: Fix.: Fixation

Category	Feature	Description
Fixation	Fixation count	Number of fixations on the line of code
	Fixation count *fix.>= 250ms*	Number of fixations with duration >= 250ms on the line of code
	Fixation count *fix.>= 500ms*	Number of fixations with duration >= 500ms on the line of code
Cluster-based AOIs	# Spatial clusters	Number of spatial clusters formed on the line of code
	# Spatial-temporal clusters	Number of spatial-temporal clusters formed on the line of code

Table 2: Additional features extracted at the line level. Abbreviations: Fix.: Fixation

new tasks. This performance was measured using F1, recall, accuracy and precision metrics as shown in Table 3. These performance metrics are typically used to evaluate classification ML models in similar contexts (e.g., [13, 20]).

The replication package including the design and the analysis material of the experiment is available in our online repository[6].

5 FINDINGS AND DISCUSSION

Model Performance. To answer RQ1 (cf. Sect. 1), we evaluated 12 models trained respectively with the 4 datasets presented in Sect. 4.2.2 *(i.e., LI, FI, LR, FR)* and cross-validated following the 3 scenarios introduced in Sect. 4.2.3 (i.e., *one-participant-out, one-task-out, one-participant-task-out*). Table 3 summarizes the performance of our models.

[6]See. https://github.com/aminobest/ICPC2022_fineGrainedCL

Overall, our models reach the best performance when estimating the mentally demanding fragments of code for new participants (F1: 85.65%, recall: 84.25%, accuracy 86.24%, precision: 88.61%). This performance remains stable when testing the model on new tasks and new participants performing new tasks. Also, it does not vary much between models trained with the initial and revised highlights. These insights support the generalizability of our findings across developers and different tasks (i.e., R5, cf. Sect. 2).

Not surprisingly, our models perform better at the fragment of code level compared to the line level (cf. Table 3). The moderate scores at the line level can potentially be attributed to the subjectivity of participants. As mentioned in Sect. 4.2.2, some participants highlighted specific code tokens with the intention to convey that the entire part of code associated with these tokens was mentally demanding.

The performance scores of the ML models under the three cross-validation scenarios shown in Table 3 are similar with a little advantage for the prediction of the mentally demanding parts of code for new participants (i.e., the one-participant-out scenario). Compared with the literature, our performance scores (in terms of recall, accuracy and precision measures) for the ML models predicting the mentally demanding code fragments are in the same range as those in [20] where the authors used HRV and eye-tracking to predict code regions associated with increased difficulty (best prediction in [20]: recall: 79% , accuracy: 83%, precision: 89%). However, our results might not be directly comparable as the authors in [20], used cross-validation approaches (i.e., the conventional leave-one-sample-out validation and the K-fold validation with 5 splits) which cannot clearly demonstrate how their models generalize across developers and different tasks.

Our cross-validation approach was also adopted in [13]. However, the labels used to train the ML models were collected at a (coarse-grained) task level and are therefore not directly comparable. Taking this disparity into account, the performance scores (in terms of F1, recall and precision measures) of our ML models operating at the fragment level (and based on eye-tracking features only) outperform the ML models in [13] trained with EEG, GSR and eye-tracking features combined (best prediction in [13] F1: 73.33%, recall: 68.79%, precision: 84.38%). It is also worthwhile to mention that the results of our three cross-validations are more stable compared to those in [13], suggesting that our ML models perform equally well in all the covered cross-validation scenarios.

The models trained with the initial highlights and those trained with the revised highlights provide similar performance with a little advantage for the latter ones. By comparing the participants' initial and revised highlights, we, indeed, could not perceive big differences between both. Several rounds of highlights were also conducted in [10]. However, the authors did not investigate whether there is a difference between the initial and revised highlights with regards to their ability to provide better estimates of cognitive load. Based on our findings, we postulate that collecting highlights once might be good enough for the training of the ML models.

Features' Importance. To answer RQ2 (cf. Sect. 1), we derived the importance of each feature group (i.e., fixation-based, saccade-based, pupil-based, scan-path-based, cluster-based AOIs) for each of the generated models (i.e., 12 models in total) as shown in Table 4. Overall, pupil-based and saccade-based features have the highest importance scores in the ML models predicting the mentally demanding lines and fragments of code respectively. Compared to the literature, the importance of different classes of eye-tracking features reported in our study delivers more detailed insights than those reported in [13, 20] where features were grouped by modality (EEG, eye-tracking, GSR and HRV) and compared for their ability to estimate cognitive load.

Implications. Our work has implications on research, practice and education. With regards to research, our approach opens for a new class of (laboratory) experiments where users' cognitive load can be measured at a fine-grained level of the source code. This, in turn, can be used to isolate the parts of code susceptible to have quality concerns (e.g., due to the presence of code smells [12, 54]), scrutinize the underlying cognitive load and contrast it with the load associated with the reading of other parts of code with no quality concerns. In addition, our approach is designed with the aim to apply it to large software systems and not only small source code snippets. Therefore, our approach can promote large-scale empirical studies investigating users' cognitive load at a fine-grained level when engaging with large code bases, reflecting the true scale of software projects in the real-world.

For practice, our findings demonstrate that in a controlled laboratory setting, eye-tracking features can estimate users' cognitive load and help to pinpoint the mentally demanding parts of code. However, to use our approach in practical settings, it is important to investigate its applicability in less controlled environments. We advance that different types of eye-tracking features (i.e., based on fixations, saccades, pupillary, scan-path and AOI measures) capture cognitive load from different dimensions (i.e., behavioral, physiological). This, in turn, can provide a multi-modal measurement that mitigates the constraints/limitations of the individual measures (e.g., sensibility of pupillary measures to light variations) when being collected in less controlled environments. Nevertheless, this assumption still needs to be backed up by empirical evidence. If successful, our approach can serve as a basis for generating "code-highlights" indicating the critical parts that have challenged developers' when engaging with the source code and thus are likely to contain quality concerns. These code-highlights can support code review tasks by guiding reviewers' attention towards the critical parts of code and thus helping them to identify the quality concerns that could hinder the interpretation of the source code.

When it comes to education, with further development, our approach can be embedded in adaptive e-learning systems, which given a certain learning material (e.g., an explanation of a design pattern and a source code exemplifying it), can pinpoint the exact parts challenging the comprehension of the students. This information can help to automatically adjust the pace at which the student progresses with the learning material and set the focus on the parts requiring further explanations, examples and exercises. Moreover, such an approach can inform the instructor about the common challenges and potential pitfalls in the learning material.

6 THREATS TO VALIDITY

Internal Validity. Failing to instruct participants about the experiment tasks and lacking control over the external environment can pose significant issues to the internal validity of an empirical study. To mitigate these threats, we have designed a careful protocol to instruct participants uniformly during the experiment. Moreover, to ensure the quality of our data, we have conducted the data collection in a controlled lab environment where the illumination was controlled to ensure no optical artifacts or pupillary reactions due to light variations [21]. The tasks used in the experiment could also affect the internal validity of the study. To mitigate this threat, the design of our tasks was inspired by the source code patterns used in [13, 39] which have been already shown to trigger reactions associated with cognitive load. Another potential internal threat to validity is associated with the learning effect during the experiment. We mitigated this threat by randomizing the order of the tasks to spread out this effect uniformly over the different experiment runs.

Accr.	Level	Highlights	One-participant-out				One-task-out				One-participant-task-out			
			F1	Recall	Acc.	Prec.	F1	Recall	Acc.	Prec.	F1	Recall	Acc.	Prec.
LI	**Line**	**Initial**	67.51%	67.36%	69.94%	71.23%	65.17%	65.98%	67.92%	68.94%	64.74%	70.73%	70.80%	69.51%
FI	**Fragment**	**Initial**	84.98%	85.47%	85.16%	86.25%	80.79%	80.86%	84.21%	85.29%	79.78%	82.84%	84.16%	81.15%
LR	**Line**	**Revised**	68.57%	69.01%	70.75%	71.78%	67.22%	67.43%	70.30%	71.60%	65.96%	73.35%	71.48%	68.96%
FR	**Fragment**	**Revised**	85.65%	84.25%	86.24%	88.61%	81.46%	80.26%	84.47%	85.82%	81.14%	84.32%	85.56%	82.21%

Table 3: Performance of ML models. Abbreviations: Accr.: Accronym, Acc.: Accuracy, Prec.: Precision

Feature gr.	Ac.	Level	Highl.	Feature gr. importance		
				OPO	OTO	OPTO
Fixation	**LI**	**Line**	**Initial**	0.102	0.126	0.132
	FI	**Frag.**	**Initial**	0.118	0.093	0.11
	LR	**Line**	**Revised**	0.155	0.162	0.167
	FR	**Frag.**	**Revised**	0.082	0.094	0.092
Saccade	**LI**	**Line**	**Initial**	0.134	0.151	0.157
	FI	**Frag.**	**Initial**	**0.653**	**0.68**	**0.655**
	LR	**Line**	**Revised**	0.126	0.135	0.115
	FR	**Frag.**	**Revised**	**0.702**	**0.7**	**0.694**
Pupil	**LI**	**Line**	**Initial**	**0.587**	**0.524**	**0.547**
	FI	**Frag.**	**Initial**	0.147	0.159	0.164
	LR	**Line**	**Revised**	**0.539**	**0.563**	**0.553**
	FR	**Frag.**	**Revised**	0.151	0.141	0.147
Scan-path	**LI**	**Line**	**Initial**	0.174	0.198	0.158
	FI	**Frag.**	**Initial**	0.076	0.062	0.064
	LR	**Line**	**Revised**	0.176	0.135	0.158
	FR	**Frag.**	**Revised**	0.059	0.055	0.062
Cluster-based AOI	**LI**	**Line**	**Initial**	0.003	0.002	0.006
	FI	**Frag.**	**Initial**	0.006	0.007	0.007
	LR	**Line**	**Revised**	0.004	0.004	0.007
	FR	**Frag.**	**Revised**	0.005	0.01	0.006

Table 4: Importance of features. Abbreviations: Gr.: Group, Ac.: Acronym, Highl.: Highlights, OPO: One-participant-out, OTO: One-task-out, OPTO: one-participant-task-out, Frag. Fragment

External Validity. To mitigate external validity threats, we have used a cross-validation approach and showed the generalization of our ML models across developers and different tasks (i.e., R5, cf. Sect. 2). Nevertheless, it is worthwhile to mention that all the participants (who served as proxies for developers) have an academic background and therefore to demonstrate the external validity of our work, it is crucial to replicate our study with participants from industry. In this replication, it is also important to use larger source code bases, which our approach supports due to the ability to deal with both small and large code bases (i.e., R4). Furthermore, although eye-tracking can be seen as more lightweight and easy to use than other sensors (e.g., EEG, fMRI, fNIRS; i.e., R3), it is possible that when moving outside of laboratory settings, factors such as changes in lighting conditions and lack of instructions about the use of eye-tracking might affect the quality of the collected data and thus the robustness of our ML models.

Construct Validity. To mitigate construct validity threats, our use of eye-tracking to derive a continuous measurement of cognitive load (i.e., R1) was supported by the existing literature [21, 36, 42, 43]. As for the contextualization of fixations to support the measurement of cognitive load at a fine-grained (spatial) level, it was motivated by the eye-mind hypothesis [23] (i.e., R2). Furthermore, we have used different classes of features (i.e., fixation-based, saccade-based, pupil-based, scan-path-based, cluster-based AOIs) that have been individually associated with cognitive load in a number of studies (cf. Sect. 3.2). Hence, the training of our models mitigates the risk of the mono-method bias [51].

Conclusion Validity. Our sample size could be a threat. However, this sample size is comparable to similar studies [13]. This possible limitation will be addressed in our future work.

7 CONCLUSION

This paper presents a novel approach allowing to estimate developers' cognitive load at a fine-grained level and accordingly indicate the mentally demanding parts of code. This approach was based on a number of features characterizing users' fixations, saccades, pupil reactions, scan-paths and cluster-based AOIs. These features were, in turn, used to develop a set of ML models providing estimates on the mentally demanding parts of code. We have evaluated these models in three scenarios to investigate their ability to identify the mentally demanding lines and fragments of code for new participants, new tasks and new participants performing new tasks. Overall, our findings demonstrate high performance when conducting predictions at the level of code fragments in all these scenarios.

As future work, we are planning to apply our approach to large source code projects and recruit a wider array of participants beyond academia (e.g., developers in the industry). Moreover, we are planning to incorporate source code metrics in the training of our ML models. These metrics will be carefully selected to provide an overarching characterization covering both the essential and accidental complexities [5] of the source code at a fine-grained level. This new approach will provide additional features that will be evaluated with respect to their impact on the performance of our ML models.

ACKNOWLEDGMENT

Special thanks to Hamed Hemati for his feedback on the machine learning analysis presented in this work.

REFERENCES

[1] Charu C Aggarwal. 2015. *Data mining: the textbook*. Springer.
[2] Taylor Armerding. 2018. Hard Questions Raised When A Software 'Glitch' Takes Down An Airliner. (2018). https://www.forbes.com/sites/taylorarmerding/2018/11/20/hard-questions-raised-when-a-software-glitch-takes-down-an-airliner/#5cf4e9907b1d.

[3] Jackson Beatty. 1982. Task-evoked pupillary responses, processing load, and the structure of processing resources. *Psychological bulletin* 91, 2 (1982), 276.

[4] Nicolas Bourdillon, Laurent Schmitt, Sasan Yazdani, Jean-Marc Vesin, and Grégoire P Millet. 2017. Minimal window duration for accurate HRV recording in athletes. *Frontiers in neuroscience* 11 (2017), 456.

[5] F Brooks and H Kugler. 1987. *No silver bullet.* April.

[6] Fang Chen, Jianlong Zhou, Yang Wang, Kun Yu, Syed Z Arshad, Ahmad Khawaji, and Dan Conway. 2016. *Robust multimodal cognitive load measurement.* Springer.

[7] Ricardo Couceiro, Raul Barbosa, João Durães, Gonçalo Duarte, João Castelhano, Catarina Duarte, Cesar Teixeira, Nuno Laranjeiro, Júlio Medeiros, Paulo Carvalho, et al. 2019. Spotting Problematic Code Lines using Nonintrusive Programmers' Biofeedback. In *2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE).* IEEE, 93–103.

[8] Clive Davidson. 2012. A dark knight for algos. *Risk* 25, 9 (2012), 32.

[9] Rachel N Denison, Jacob A Parker, and Marisa Carrasco. 2020. Modeling pupil responses to rapid sequential events. *Behavior research methods* 52, 5 (2020), 1991–2007.

[10] Sarah Fakhoury, Devjeet Roy, Yuzhan Ma, Venera Arnaoudova, and Olusola Adesope. 2020. Measuring the impact of lexical and structural inconsistencies on developers' cognitive load during bug localization. *Empirical Software Engineering* 25, 3 (2020), 2140–2178.

[11] Alberto Fernández, Salvador García, Mikel Galar, Ronaldo C Prati, Bartosz Krawczyk, and Francisco Herrera. 2018. *Learning from imbalanced data sets.* Vol. 10. Springer.

[12] Martin Fowler. 2018. *Refactoring: improving the design of existing code.* Addison-Wesley Professional.

[13] Thomas Fritz, Andrew Begel, Sebastian C Müller, Serap Yigit-Elliott, and Manuela Züger. 2014. Using psycho-physiological measures to assess task difficulty in software development. In *Proceedings of the 36th international conference on software engineering.* 402–413.

[14] Andreas Glöckner and Ann-Katrin Herbold. 2008. Information processing in decisions under risk: Evidence for compensatory strategies based on automatic processes. *MPI collective goods preprint* 2008/42 (2008).

[15] Lucian Gonçales, Kleinner Farias, Bruno da Silva, and Jonathan Fessler. 2019. Measuring the cognitive load of software developers: A systematic mapping study. In *IEEE/ACM 27th International Conference on Program Comprehension.* 42–52.

[16] Lucian Gonçales, Kleinner Farias, and Bruno C da Silva. 2021. Measuring the cognitive load of software developers: An extended Systematic Mapping Study. *Information and Software Technology* (2021), 106563.

[17] Dan Gopstein, Jake Iannacone, Yu Yan, Lois DeLong, Yanyan Zhuang, Martin K-C Yeh, and Justin Cappos. 2017. Understanding misunderstandings in source code. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering.* 129–139.

[18] Drew T Guarnera, Corey A Bryant, Ashwin Mishra, Jonathan I Maletic, and Bonita Sharif. 2018. itrace: Eye tracking infrastructure for development environments. In *Proceedings of the 2018 ACM Symposium on Eye Tracking Research & Applications.* 1–3.

[19] Eija Haapalainen, SeungJun Kim, Jodi F Forlizzi, and Anind K Dey. 2010. Psycho-physiological measures for assessing cognitive load. In *Proceedings of the 12th ACM international conference on Ubiquitous computing.* 301–310.

[20] Haytham Hijazi, Ricardo Couceiro, João Castelhano, Paulo De Carvalho, Miguel Castelo-Branco, and Henrique Madeira. 2021. Intelligent Biofeedback Augmented Content Comprehension (TellBack). *IEEE Access* 9 (2021), 28393–28406.

[21] K. Holmqvist, M. Nyström, R. Andersson, R. Dewhurst, H. Jarodzka, and J. van de Weijer. 2011. *Eye Tracking: A comprehensive guide to methods and measures.* OUP Oxford.

[22] Joel Jordan and Mel Slater. 2009. An analysis of eye scanpath entropy in a progressively forming virtual environment. *Presence* 18, 3 (2009), 185–199.

[23] Marcel A Just and Patricia A Carpenter. 1980. A theory of reading: From eye fixations to comprehension. *Psychological review* 87, 4 (1980), 329.

[24] Merve Keskin, Kristien Ooms, Ahmet Ozgur Dogru, and Philippe De Maeyer. 2020. Exploring the Cognitive Load of Expert and Novice Map Users Using EEG and Eye Tracking. *ISPRS International Journal of Geo-Information* 9, 7 (2020).

[25] Katja Kevic, Braden M Walters, Timothy R Shaffer, Bonita Sharif, David C Shepherd, and Thomas Fritz. 2015. Tracing software developers' eyes and interactions for change tasks. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering.* 202–213.

[26] Katja Kevic, Braden M Walters, Timothy R Shaffer, Bonita Sharif, David C Shepherd, and Thomas Fritz. 2017. Eye gaze and interaction contexts for change tasks–Observations and potential. *Journal of Systems and Software* 128 (2017), 252–266.

[27] Jeff Klingner. 2010. Fixation-aligned pupillary response averaging. In *Proceedings of the 2010 symposium on eye-tracking research & applications.* 275–282.

[28] Timothy B. LEE. 2018. Report: Software bug led to death in Uber's self-driving crash. (2018). https://arstechnica.com/tech-policy/2018/05/report-software-bug-led-to-death-in-ubers-self-driving-crash/.

[29] Nancy G Leveson and Clark S Turner. 1993. An investigation of the Therac-25 accidents. *Computer* 26, 7 (1993), 18–41.

[30] Jacques-Louis Lions, Lennart Luebeck, Jean-Luc Fauquembergue, Gilles Kahn, Wolfgang Kubbat, Stefan Levedag, Leonardo Mazzini, Didier Merle, and Colin O'Halloran. 1996. Ariane 5 flight 501 failure report by the inquiry board.

[31] Norman H Mackworth. 1965. Visual noise causes tunnel vision. *Psychonomic science* 3, 1 (1965), 67–68.

[32] James G May, Robert S Kennedy, Mary C Williams, William P Dunlap, and Julie R Brannan. 1990. Eye movement indices of mental workload. *Acta psychologica* 75, 1 (1990), 75–89.

[33] Júlio Medeiros, Ricardo Couceiro, Gonçalo Duarte, João Durães, João Castelhano, Catarina Duarte, Miguel Castelo-Branco, Henrique Madeira, Paulo de Carvalho, and César Teixeira. 2021. Can EEG Be Adopted as a Neuroscience Reference for Assessing Software Programmers' Cognitive Load? *Sensors* 21, 7 (2021), 2338.

[34] Juliano Paulo Menzen, Kleinner Farias, and Vinicius Bischoff. 2021. Using biometric data in software engineering: a systematic mapping study. *Behaviour & Information Technology* 40, 9 (2021), 880–902.

[35] George A Miller. 1956. The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological review* 63, 2 (1956), 81.

[36] Unaizah Obaidellah, Mohammed Al Haek, and Peter C-H Cheng. 2018. A survey on the usage of eye-tracking in computer programming. *ACM Computing Surveys (CSUR)* 51, 1 (2018), 1–58.

[37] Anneli Olsen. 2012. The Tobii I-VT Fixation Filter. (2012).

[38] Fred Paas, Juhani E Tuovinen, Huib Tabbers, and Pascal WM Van Gerven. 2003. Cognitive load measurement as a means to advance cognitive load theory. *Educational psychologist* 38, 1 (2003), 63–71.

[39] Norman Peitek, Sven Apel, Chris Parnin, André Brechmann, and Janet Siegmund. 2021. Program comprehension and code complexity metrics: An fmri study. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE).* IEEE, 524–536.

[40] Gillian Porter, Tom Troscianko, and Iain D Gilchrist. 2007. Effort during visual search and counting: Insights from pupillometry. *Quarterly journal of experimental psychology* 60, 2 (2007), 211–229.

[41] Christos Saltapidas and Ramin Maghsood. 2018. Financial Risk The fall of Knight Capital Group. (2018).

[42] Zohreh Sharafi, Timothy Shaffer, Bonita Sharif, and Yann-Gaël Guéhéneuc. 2015. Eye-tracking metrics in software engineering. In *2015 Asia-Pacific Software Engineering Conference (APSEC).* IEEE, 96–103.

[43] Zohreh Sharafi, Zéphyrin Soh, and Yann-Gaël Guéhéneuc. 2015. A systematic literature review on the usage of eye-tracking in software engineering. *Information and Software Technology* 67 (2015), 79–107.

[44] Stuart R Steinhauer, Greg J Siegle, Ruth Condray, and Misha Pless. 2004. Sympathetic and parasympathetic innervation of pupillary dilation during sustained processing. *International journal of psychophysiology* 52, 1 (2004), 77–86.

[45] Robert J Sternberg and Karin Sternberg. 2016. *Cognitive psychology.* Nelson Education.

[46] John Sweller. 2011. Cognitive load theory. In *Psychology of learning and motivation.* Vol. 55. Elsevier, 37–76.

[47] Pauline van der Wel and Henk van Steenbergen. 2018. Pupil dilation as an index of effort in cognitive control tasks: A review. *Psychonomic bulletin & review* 25, 6 (2018), 2005–2015.

[48] Boris M Velichkovsky. 1999. From levels of processing to stratification of cognition Converging evidence from three domains of research. *Stratification in cognition and consciousness* 15 (1999), 203.

[49] Yuanchao Wang, Zhichen Pan, Jianhua Zheng, Lei Qian, and Mingtao Li. 2019. A hybrid ensemble method for pulsar candidate classification. *Astrophysics and Space Science* 364, 8 (2019), 1–13.

[50] Barbara Weber, Thomas Fischer, and René Riedl. 2021. Brain and autonomic nervous system activity measurement in software engineering: A systematic literature review. *Journal of Systems and Software* 178 (2021).

[51] Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, and Anders Wesslén. 2012. *Experimentation in software engineering.* Springer Science & Business Media.

[52] Jeremy M Wolfe. 1994. Guided search 2.0 a revised model of visual search. *Psychonomic bulletin & review* 1, 2 (1994), 202–238.

[53] Martin K-C Yeh, Dan Gopstein, Yu Yan, and Yanyan Zhuang. 2017. Detecting and comparing brain activity in short program comprehension using EEG. In *2017 IEEE Frontiers in Education Conference (FIE).* IEEE, 1–5.

[54] Min Zhang, Tracy Hall, and Nathan Baddoo. 2011. Code bad smells: a review of current knowledge. *Journal of Software Maintenance and Evolution: research and practice* 23, 3 (2011), 179–202.

[55] Robert Z Zheng. 2017. *Cognitive load measurement and application: a theoretical framework for meaningful research and practice.* Routledge.

[56] Stefan Zugal, Jakob Pinggera, Manuel Neurauter, Thomas Maran, and Barbara Weber. 2017. Cheetah experimental platform web 1.0: cleaning pupillary data. *arXiv preprint arXiv:1703.09468* (2017).