Diagnosing Medical Score Calculator Apps

SYDUR RAHAMAN, New Jersey Institute of Technology, USA
RAINA SAMUEL, New Jersey Institute of Technology, USA
IULIAN NEAMTIU, New Jersey Institute of Technology, USA

Mobile medical score calculator apps are widely used among practitioners to help make decisions regarding patient treatment and diagnosis. Errors in score definition, input, or calculations can result in severe and potentially life-threatening situations. Despite these high stakes, there has been no systematic or rigorous effort to examine and verify score calculator apps. We address these issues via a novel, interval-based score checking approach. Based on our observation that medical reference tables themselves may contain errors (which can propagate to apps) we first introduce automated correctness checking of reference tables. Specifically, we reduce score correctness checking to partition checking (coverage and non-overlap) over score parameters' ranges. We checked 12 scoring systems used in emergency, intensive, and acute care. Surprisingly, though some of these scores have been used for decades, we found errors in 5 score specifications: 8 coverage violations and 3 non-overlap violations. Second, we design and implement an automatic, dynamic analysis-based approach for verifying score correctness in a given Android app; the approach combines efficient, automatic GUI extraction and app exploration with partition/consistency checking to expose app errors. We applied the approach to 90 Android apps that implement medical score calculators. We found 23 coverage violations in 11 apps; 32 non-overlap violations in 12 apps, and 16 incorrect score calculations in 16 apps. We reported all findings to developers, which so far has led to fixes in 6 apps.

CCS Concepts: • **Human-centered computing → Smartphones**; • **Software and its engineering → Software testing and debugging**.

Additional Key Words and Phrases: Medical mobile apps, medical score calculators, Android analysis, dynamic analysis

ACM Reference Format:
Sydur Rahaman, Raina Samuel, and Iulian Neamtiu. 2023. Diagnosing Medical Score Calculator Apps. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 7, 3, Article 118 (September 2023), 27 pages. https://doi.org/10.1145/3610912

1 INTRODUCTION

Mobile medical apps see substantial and growing use in clinical settings, with junior personnel being more likely to use apps (Section 2.1). For emergency personnel, medical score calculators are among the most used apps 66.7% [33]. While medical scores and score calculator apps are appealing, helping practitioners quickly assess patient condition, the reliability of such scores and score calculator apps have received little attention. Reliability is particularly important in acute care, e.g., emergency or ICU, where accuracy can decisively influence outcomes.

Therefore, our focus is the correctness of medical score calculators, i.e., apps that compute a medical score based on supplied parameters, as well as the underlying score reference table. Such scores are ubiquitous in triage, ICU, and early warning, e.g., the Sepsis-related Organ Failure Assessment (SOFA) is used in the ICU to determine the rate of organ failure, onset of sepsis, etc. Traditionally, scores were calculated manually from reference tables

Authors' addresses: Sydur Rahaman, sr939@njit.edu, New Jersey Institute of Technology, Newark, USA; Raina Samuel, res9@njit.edu, New Jersey Institute of Technology, Newark, USA; Iulian Neamtiu, ineamtiu@njit.edu, New Jersey Institute of Technology, Newark, USA.

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 the author(s) 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.
© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM.
2474-9567/2023/9-ART118 $15.00
https://doi.org/10.1145/3610912

SOFA score	1	2	3	4
Respiration				
PaO_2/FiO_2, mmHg	<400	<300	<200	<100
			with respiratory support	
Coagulation				
Platelets $\times 10^3/mm^3$	<150	<100	<50	<20
Liver				
Bilirubin, mg/dl	1.2–1.9	2.0–5.9	6.0–11.9	>12.0
(μmol/l)	(20–32)	(33–101)	(102–204)	(<204)
Cardiovascular				
Hypotension	MAP < 70 mmHg	Dopamine ≤5 or dobutamine (any dose)[a]	Dopamine >5 or epinephrine ≤0.1 or norepinephrine ≤0.1	Dopamine >15 or epinephrine >0.1 or norepinephrine >0.1
Central nervous system				
Glasgow Coma Score	13–14	10–12	6–9	<6
Renal				
Creatinine, mg/dl	1.2–1.9	2.0–3.4	3.5–4.9	>5.0
(μmol/l) or urine output	(110–170)	(171–299)	(300–440) or <500 ml/day	(>440) or <200 ml/day

[a] Adrenergic agents administered for at least 1 h (doses given are in μg/kg·min)

Missing value 12.0

Should be > 204

Missing value 5.0

Fig. 1. SOFA Score (from Vincent et al. [53]).

which map parameter values to several score components; these components are added to obtain an overall score, which is then checked against an action threshold.

Reference tables are usually defined in medical research papers or regulatory documents. According to our observations, two main issues impact score calculator apps' correctness. First, the reference tables can be inconsistent, which can lead to erroneous scores even when the score is computed manually. Second, the implementation of these scoring systems in apps can be incorrect: either due to developer error, or due to developer confusion induced by attempting to implement an inconsistently-defined score, i.e., an incorrect specification.

Incorrect scores can have dire consequences. For example, the Modified Early Warning Score (MEWS) [26] determines whether a patient should be moved to the ICU, based on the score value being ≥ 4. Thus, when a score calculator produces an incorrect low score, the condition's severity is underestimated, which can result in misdiagnosis and other negative patient outcomes.

This area continues to be under-scrutinized and under-regulated. We are not aware of any prior attempts to rigorously verify reference scores, or to verify/validate score calculator apps automatically. Other prior efforts, whether targeted studies on apps implementing opioid [28] or insulin [34] calculators, as well as medical app meta-studies [11] have revealed significantly issues: different results across apps for the same calculation, incorrect dosage, potential harmful recommendations, and a lack of medical professional involvement in app creation. Hence there is an impetus for medical score calculators (and medical apps in general) to be highly scrutinized, but in practice they are not, as they fall out of the scope of many regulatory bodies. For instance, in the US, the FTC (Federal Trade Commission) currently only regulates mobile medical apps that are used as a device or connect with a device, e.g., an insulin pump [25].

To this end, in this paper we introduce the first *automated*, *rigorous* approach for verifying reference scores and their implementations in different apps. Specifically, we aim to answer the following research questions:
RQ1) Can our approach extract and verify medical score specifications?
RQ2) Is our approach effective at analyzing and finding errors in real-world apps?

In Section 2 we define the properties we check, i.e., two partition conditions, and motivate our approach by identifying and exemplifying three sources of errors in score calculations: inconsistent reference table, inconsistent GUI, and incorrect score calculation.

We describe our approach and methodology in Section 3: given a medical reference table defined in a simple tabular format, we automatically generate code to check the table for partition condition violations with the help of the Z3 theorem prover [21]. Next, based on this reference specification, we developed a novel dynamic (runtime) approach that automatically verifies the app for GUI consistency, compliance with the reference table, and correct score calculation (Section 3.3). Automatic verification required addressing several challenges. First, we mapped heterogeneous, ad-hoc GUI information to reference table entries (intervals) via semi-supervised clustering. Second, we combined Depth-first Search with UI automation to systematically and automatically explore GUIs, as well as extract score-relevant information.

We hold 12 long-established, widely-used medical scores, up to scrutiny; the evaluation of our score extraction and verification approach is presented in Section 4. We were able to automatically expose errors in five score reference tables, defined in medical research articles. Note that these scores have been used for decades: they were introduced as early as 1985, and no later than 2008; and cited heavily (between 157 and 21,863 citations, respectively).

In Section 5 we evaluate our app checking approach on all 90 Google Play apps implementing these scores. The approach attained 100% precision and 80% recall. We found dozens of errors that lead to inconsistent GUIs; these confuse the user, invite input errors, and can even prohibit legitimate values from being introduced (Section 5.2). Finally, we found score calculation errors, including around threshold values at which emergency intervention is necessary (Section 5.3).

To summarize, we make the following contributions:

- An approach for extracting and checking medical reference table specifications, applied to 12 scores.
- An approach for checking GUI instantiations of reference tables, and apps' calculation w.r.t. a reference score, in Android apps.
- A classification of errors in reference scores and incorrect score implementations in Android apps.

Responsible Disclosure. We have disclosed the errors we found to app developers. As of Feb. 7, 2023, developers of 6 apps have responded to our bug reporting. All confirmed the errors, fixed the apps, and released app updates (details in Table 5). Two respondents had medical (MD) degrees: one MD was the app company's president; the other, a developer, had MD and CS degrees. Confirmation from MDs underscores the relevance and importance of our findings.

Artifacts. All our data (apps, reference scoring systems, errors in scores and apps, etc.) is accessible at https://doi.org/10.17605/OSF.IO/XQEJ5.

2 MOTIVATION

To motivate our approach, we first quantify the adoption of mobile apps in clinical care; next, we define the key terms and concepts used throughout the paper, and then provide examples of errors in actual reference tables and apps.

2.1 Mobile Medical Apps Usage

2.1.1 Medical Apps and Calculators in Acute Care. The adoption of mobile medical apps in a clinical setting in general is already strong, with clinical smartphone use among physicians being reported at 70% and above as early as 2012 [44, 54]. Mobile medical apps and smartphone-based reference material are widely used in emergency/acute care. For example, Hitti et al.'s 2021 study [33] regarding emergency department personnel has revealed that 91.8% of those surveyed used medical apps on their devices during their shifts, amidst heavy workloads and a stressful environment. Flynn et al.'s 2018 study [24] showed that 98% of acute care nurses used a smartphone in acute settings "to access information on medications, procedures, and diseases". Green et al.'s 2019

study [27] revealed that "60% of users indicated that they are somewhat or very likely to use newly published medical calculators".

2.1.2 Higher App Usage for Inexperienced Personnel. Another impetus for studying and improving medical app reliability is that less-experienced personnel might rely more on apps. First, there is evidence that users of medical calculator apps are clinicians and nurses, especially *inexperienced and younger doctors*, according to Hitti et al. [33]. A 2015 study among surgeons found that "Junior doctors were more likely to use medical apps over their senior colleagues (p = 0.001) as well as access the Internet on their smartphone for medical information (p <0.001)" [46]. Additionally, per Green et al.'s survey [27], clinicians with less experience are more likely to use medical calculator software; conversely, experienced clinicians had doubts on the credibility of medical calculators.

2.1.3 Medical Score Calculator Accuracy. Pelletier et al.'s 2022 study [47] on both online and mobile bleeding risk calculators, such as HAS-BLED, has found inconsistencies in calculated risk estimates which can result in harmful clinical decisions. The study has shown that such imprecise results found in apps are due to incorrect calculations, using alternative validation studies, and inaccurate translations of risk factors to risk elements. Fajardo et al.'s 2019 study [22] of online type 2 diabetes risk calculators has revealed that while calculator results are generally understandable, such calculators may not be suited for patients who lack general health literacy; the study also found that these calculators have high variability in terms of determining estimated risk.

Therefore, the reliability of scores and score calculator apps is important in general due to heavy reliance on under-regulated mobile apps, and particularly important in acute care settings where *rapid* and *correct* decisions are key for achieving positive patient outcomes. However, there is a lack of a regulatory framework and enforcement regarding medical apps. For example, the US Food and Drug Administration (FDA) has jurisdiction over medical apps. However, the FDA does not regulate apps that "automate clinical calculations and basic tasks for health professionals" [6]. Additionally, apps that are FDA-approved went through an approval process initiated by the developers themselves.

2.2 Definitions

2.2.1 Reference Table. We use the term *reference table* for the table in the form it was first introduced, e.g., in a medical research article or a regulatory agency document. For example the Sequential Organ Failure Assessment (SOFA) score, shown in Figure 1 and discussed shortly, was introduced by Vincent et al. [53] in the *Intensive Care Medicine* research journal in 1996. The NEWS score, another score we consider, was introduced by the UK's National Health Service (NHS) in 2012 and later updated in 2017 to NEWS2 [2]. Score tables are structured as follows: most commonly, each cell in the table contains intervals for one physiological parameter, while the row or column header contains a numeric value, typically 0–4. For example, in SOFA's reference table (Figure 1) the third row shows intervals 1.2–1.9, 2.0–5.9, and so on, for parameter *Bilirubin*. The header row in the table shows numeric values, in SOFA's case 1 through 4, which correspond to individual scores for the intervals in that column. Occasionally, a table entry contains just a threshold value, e.g., *MAP* < 70 *mmHg* in SOFA's fourth row. Finally, a cell can contain intervals or thresholds for more than one parameter, e.g., *Dopamine* > 15 *or norepinephrine* > 0.1 in SOFA's fourth row. Next we describe score computation.

2.2.2 Score. A score is computed by adding the individual scores corresponding to each cell. For example, a patient with *Respiration=350* (score=1), *Coagulation=90* (score=2), *Liver=7.0* (score=3), *Central nervous system=13* (score=1), *Renal=1.5* (score=1) would have an overall SOFA score:

$$SOFA\ score = 1 + 2 + 3 + 1 + 1 = 8$$

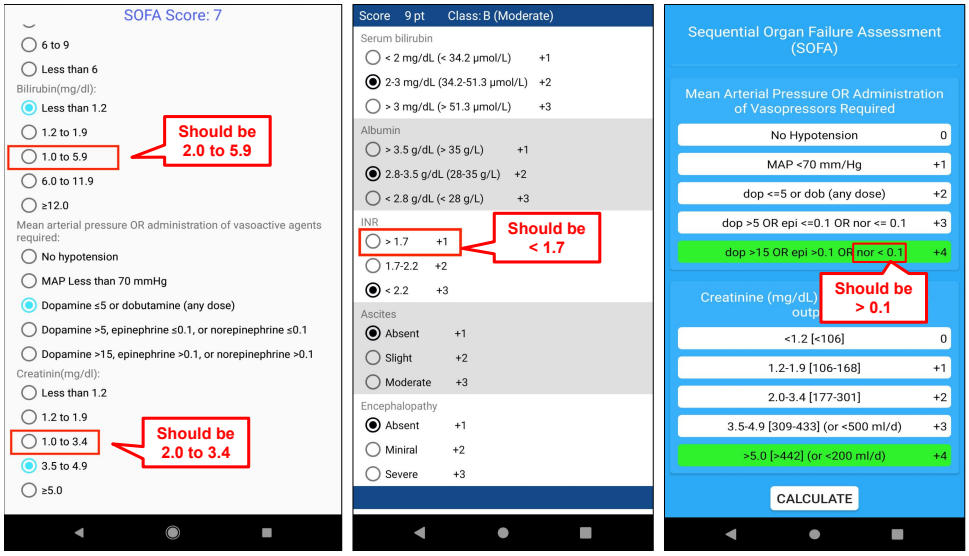


Fig. 2. Inconsistent GUI errors in three apps: Nursing Calculator (left), Child-Pugh Score (center), SOFA 1.2.0 (right).

The overall value determines the course of action. In Table 1 (discussed at length later) we show action thresholds, e.g., "aggressive treatment if HEART score ≥ 7"; hence an accurate value is critical for patients' health outcomes.

2.2.3 Partition. Let $min \in \mathbb{R}$ and $max \in \mathbb{R}$ be the minimum and maximum values for a parameter, respectively. Let $min < p_1 < p_2 < ... < p_n < max$ be ordered values. Based on the p_i's we can define intervals (e.g., closed, $I_i = [p_i, p_{i+1}]$, open, $I'_i = (p_i, p_{i+1})$, or combinations thereof). Let $I_1, I_2, \ldots, I_i, \ldots, I_n$ be nonempty ($I_i \neq \emptyset$) intervals on $[min, max]$. Then $I_1, I_2, \ldots, I_i, \ldots, I_n$ form a *partition* of $[min, max]$ if two conditions are met:

(1) Coverage (exhaustion):
$$I_1 \cup I_2 \cup \ldots \cup I_i, \ldots \cup I_n = [min, max]$$
(2) Non-overlap (disjointness):
$$\forall i, j,\ 1 \leq i < j \leq n \longrightarrow I_i \cap I_j = \emptyset$$

As we will illustrate shortly, many errors, in reference tables themselves or the apps implementing the tables, stem from violations of the aforementioned partition conditions.

2.3 Error Source #1: Inconsistent Reference Table

Errors such as inconsistent definitions in reference tables are the most concerning kinds of issues we found, because, unlike apps, tables are hard to update or fix. Moreover, as our evaluation shows, an incorrect reference table is likely to lead to incorrect implementations in apps, because developers tend to implement tables *ad literam*. Finally, an inconsistent table will lead to an inconsistent GUI that confuses app users and invites score calculation errors.

We illustrate several such inconsistencies on the *SOFA Score* reference table. The SOFA score predicts ICU mortality as follows: it evaluates the dysfunction of six systems by scoring each organ from 0, which is considered normal functionality, to 4, the most abnormal [53]. Thus the highest possible score to obtain would be 24, indicating severe morbidity, and the lowest would be 0.

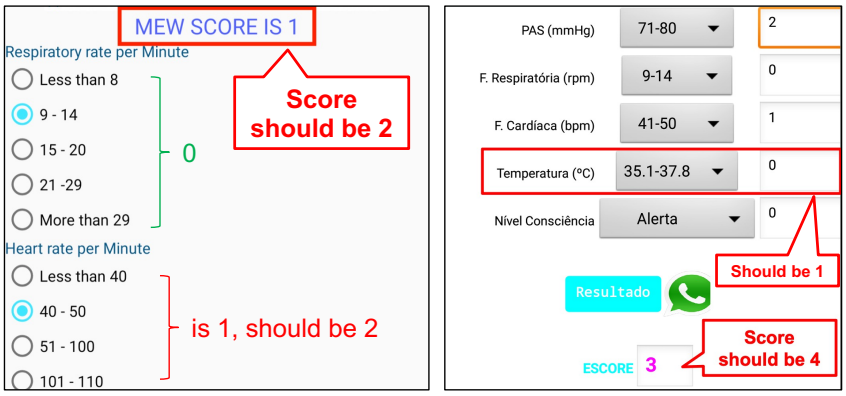


Fig. 3. Nursing Calculator incorrect score (left); MEWS Brasil incorrect scores for *Temperature* and overall (right).

The reference table for the SOFA score is shown in Figure 1. Notice how for Liver (Bilirubin), the second-to-last interval is defined as 6.0–11.9. As the parameter is a real number, the actual interval specification is $[6.0, 12.0)$. That is, a value such as 11.95 would still be in the interval because only the first decimal is specified. The last interval for bilirubin is > 12.0. Hence the interval-based specification for these two entries is: $[6.0, 12.0)$ and $(12.0, max)$. *This squarely violates the coverage property of the partition, because value 12.0 is not covered by any interval.* The same issue is present for parameter Renal (Creatinine), where value 5.0 is not covered. It is unclear how developers are supposed to cope with this incorrect specification, e.g., the SOFA score of a patient with Bilirubin=12 and Creatinine=5 can be *off by as much as 2 points*, depending on how the table is interpreted.

Finally, when bilirubin is specified in μmol/l, the table's last column shows '(< 204)' which is incorrect: the entry should be '(> 204)' (note how values < 204 are already covered in the preceding intervals). If the developer implements the table ad literam and offers '(< 204)' as a GUI option, the SOFA score of a patient can be *off by as much as 3 points*.

Note that even a off-by-one error can affect patients' condition classification, e.g., between "patient should be monitored" and "urgently inform a clinician". Section 4.2 discusses these issues at length.

2.4 Error Source #2: Inconsistent GUI

We now turn to the first kind of implementation errors, where the GUI is inconsistent; our approach detects two kinds of errors. In the first kind, the user can input the same parameter value into *two different GUI boxes*, which impacts the score; in the second kind there is no input box for a certain value. Essentially, these embody violations of the coverage and non-overlap conditions, respectively; in Section 3.3 we discuss how we check GUIs for such errors automatically via dynamic analysis and constraint solving.

Example 1: SOFA score in app Nursing Calculator. The app Nursing Calculator,[1] with over 50,000 installs, provides a variety of medical calculators, including the SOFA score. The app's GUI has two inconsistency errors (first kind), as highlighted in Figure 2 (left), and described next. The option for Bilirubin shows a range of 1.0–5.9 when it is supposed to be 2.0–5.9. Moreover, for Creatinine the range in the app is 1.0–3.4, when it should be 2.0–3.4. Due to these errors, a patient's score can be *off by as much as 2 points*.

[1]https://play.google.com/store/apps/details?id=com.niya.lijo.nursingcalculators

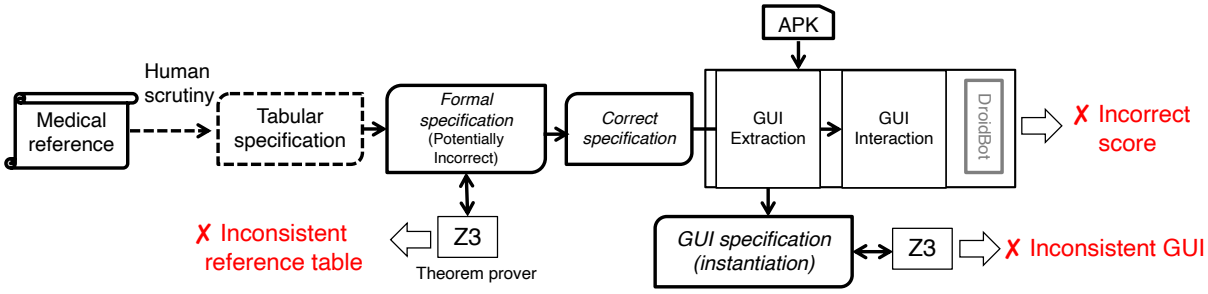


Fig. 4. Overview of our approach and toolchain.

Example 2: Child-Pugh score in app Child-Pugh Score. The Child-Pugh score is generally used to assess the potential for liver diseases, mainly cirrhosis. The app Child-Pugh Score[2] has an inconsistency error (first kind) regarding values for INR, as highlighted in Figure 2 (center): the first option should be '< 1.7' instead of '> 1.7'. Due to this error, a patient's score can be *off by as much as 2 points*.

Example 3: SOFA score in app SOFA 1.2.0. This app exemplifies the second kind of error. The app SOFA 1.2.0,[3] removed from Google Play in the course of our research, exhibited a GUI inconsistency error as highlighted in Figure 2 (right). Note that the reference table's last column in the *Cardiovascular* row specifies the score for $\ldots norepinephrine > 0.1$; the app however incorrectly lists '$norepinephrine < 0.1$'. Basically, the app offers no option where users can indicate the $norepinephrine > 0.1$ condition; this error can alter the score by 1 point.

2.5 Error Source #3: Incorrect Score Calculation

Even with a consistent table and consistent GUI, apps can still be prone to errors in score calculation, e.g., per the table the score is 4, but the app displays 6. These calculation errors are silent, hence particularly pernicious (the user does not have any indication that the calculation has gone awry).

We now present several examples, based on the Modified Early Warning Score (MEWS), used by professionals to determine whether or not a surgical in-patient requires intensive care [26].

Nursing Calculator. When the app starts, parameter values are in their default settings, i.e., individual scores are 0, hence the MEWS score is 0. After the user changes the Heart rate to 40, the output score is 1 instead of the expected value of 2 (screenshot in Figure 3 left). Note that a higher MEWS value indicates a more severe situation.

Another example is the MEWS Brasil app [5], Figure 3 (right). Suppose the user inputs a 41–50 Heart rate and BP between 71–80; cumulatively, the score is 3. The error manifests when the temperature is in the interval 35.1–36; per the table, the individual temperature score value is 1. In the app however, the value is 0, hence the displayed overall MEWS score, 3, is incorrect (correct value: 4). This error is particularly problematic, because for MEWS, 4 is a threshold value: "[at $\geq$ 4] surgical team should be informed immediately" [26].

These incorrect implementations can lead to differences in assessment, and subsequently outcome, making verification of scoring systems crucial.

3 APPROACH

We now describe our approach to finding errors in reference tables, app GUIs, and app score calculations. An overview is shown in Figure 4. In the first stage for each scoring system, we extract a *specification* for the reference table; our toolchain then checks the specification for consistency, i.e., for coverage and non-overlap

[2]https://play.google.com/store/apps/details?id=br.child_pugh
[3]https://apksos.com/app/com.varendrasoft.sofascore

violations using Z3. We then fix the inconsistency found in the reference table before using it as a reference. Next, for a given app (APK) implementing that score, our toolchain first performs a dynamic analysis to extract a *GUI specification* (aka the GUI instantiation of specification) using the DroidBot automator [38]. The GUI specification is (a) validated against the correct reference table specification, and (b) verified for consistency, using Z3. Finally, our approach drives app execution (GUI interaction) automatically, according to specific input parameter combinations, to produce the app's output score, and verifies this score against the reference score for that parameter combination. We now discuss each phase, including a brief introduction to the underlying tools.

3.1 Partition Checking via Satisfiability

3.1.1 The Z3 Theorem Prover. Z3 [21] is a widely-used automated theorem prover for solving logical formulas in various domains. Z3 is based on Satisfiability Modulo Theories (SMT) techniques, which allow it to handle a wide range of logical theories, including arithmetic over integers or reals, bit-vectors, or arrays. Given an input formula, Z3 returns "`Sat`" or "`Unsat`". "`Sat`", short for "satisfiable," indicates that there exists at least one assignment of values to the variables in the formula that makes the formula true. For example, assuming A and B are integers, the formula:

$$A > 20 \wedge B > A$$

is satisfiable, with Z3 returning `Sat`, and the model $A = 21, B = 22$.

"`Unsat`", short for "unsatisfiable," indicates that there is no assignment of values to the variables in the formula that makes the formula true. For example, formula:

$$A > 20 \wedge B > A \wedge B < 19$$

is unsatisfiable, hence Z3 returns `Unsat`.

We use Z3 to check the satisfiability of logical formulas composed of numerical ranges/intervals. Note that most score calculations involve parameters whose values span a set of ranges, or intervals. In order to be defined as a valid set of input ranges, the ranges should meet the coverage and non-overlap conditions, as per Section 2.2. Hence we encode parameter ranges into a Z3 specification (sets of intervals over integer or real numbers, as appropriate) and then use Z3 to check whether the set of intervals meets the partition conditions.

3.1.2 Coverage (exhaustion) Checking. We encode coverage checking into Z3 by requiring that the union of a given set of intervals cover a range, defined by its minimum and maximum values. When Z3 finds a counterexample, it proves that there exists a "gap" in coverage. For example, the intervals [≥10, 6-9, ≤5], encoded into Z3 as:

$$\neg(\vee(X \geq 10, \wedge(X \geq 6, X \leq 9), X \leq 5)))$$

cover the range, hence Z3 will not find a counterexample. However, the intervals [≥10, 6-9, <5] with encoding:

$$\neg(\vee(X \geq 10, \wedge(X \geq 6, X \leq 9), X < 5)))$$

do not cover the range, and Z3 will successfully find the counterexample *X=5*, i.e., a non-covered value. Note that our implementation automatically generates a Python program that invokes Z3 via its Python API.

3.1.3 Non-overlap (disjointness) Checking. In this case, we use the equation solver feature of Z3. Given a set of variables and corresponding constraints, Z3 generates a solution (if a solution exists) that satisfies the constraints. Hence, to check for overlap between two intervals, we add them as constraints composing two sets, and as an additional constraint we check for overlap between the sets. For example, given an interval with incorrect partitioning [≥65, 45-65], which we encode as:

$$(X \geq 65, \wedge(Y \geq 45, Y \leq 65), X == Y)$$

Z3 finds an overlapping value $X = 65, Y = 65$. If this set of intervals were correctly partitioned ([>65, 45-65]), Z3 would return `Unsat`.

3.2 Reference Table Validation

We check the validity of reference tables in two steps. First, we extract the reference table from the source PDF file into a *tabular specification* in CSV format. Though we employed a PDF→CSV conversion tool, the resulting CSV is still subject to human scrutiny to ensure accurate conversion (this is one of the only two manual steps of our approach; the accuracy of this extraction step is discussed in Section 4.3).

The tabular specification is then automatically encoded into a Python program that invokes Z3 to verify that each parameter of a reference table meets the partition conditions. Note that some of the scores we considered contain real numbers, specified to one or two decimal places (e.g., 5.9, 5.32). We convert such decimals to integers, multiplying by 10 or 100, respectively (5.9 becomes 59; 5.32 becomes 532). Section 4.4 discusses the reference table errors we found.

3.3 App Verification and Validation

App score verification and validation presents several challenges: a) mapping heterogeneous GUI elements to reference table cells, which we solve via semi-supervised clustering (Section 3.3.1); b) systematically and automatically exercising the GUI, which we address by coupling Depth-first Search with DroidBot-based static and dynamic GUI information extraction (Section 3.3.3, Section 3.3.4); and c) identifying app score-related heterogeneous GUI elements and extracting the corresponding score value (Section 3.3.5). We now present our approach to solving these challenges in detail.

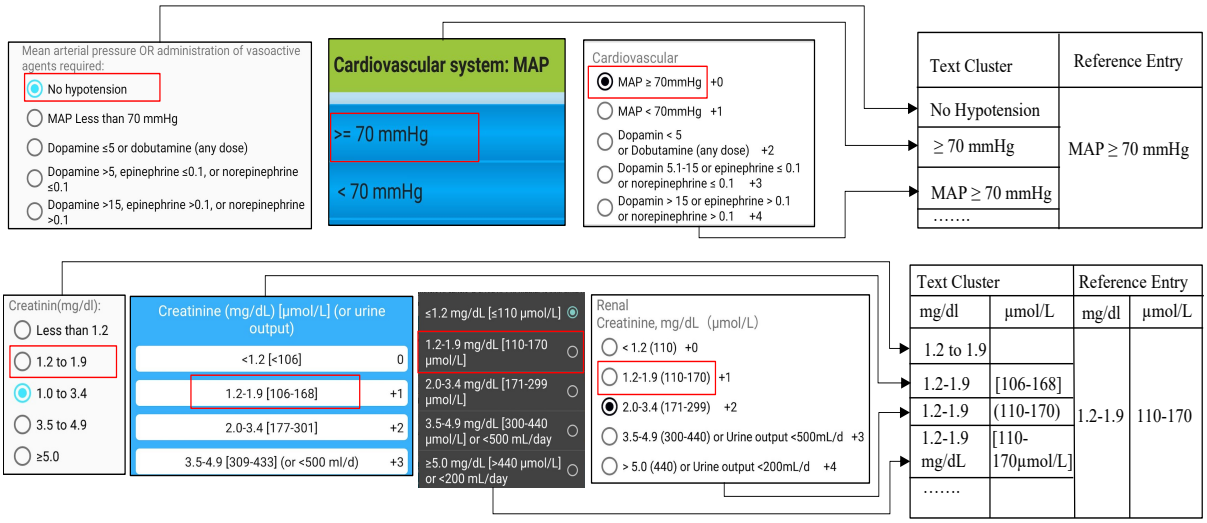


Fig. 5. Heterogeneous GUI mapping example for *Cardiovascular Mean arterial pressure* attribute and *Renal function Creatinine* attribute in apps Nursing, Nursing Calculator, SOFA score, SOFA score, and SOFA Score.

3.3.1 Mapping Heterogeneous GUIs to Specifications. We need to handle heterogeneous GUIs to create the correct specifications set for different parameters of a specific scoring system. In Figure 5, we provide two sets of examples of heterogeneous GUI elements that must be translated to SOFA score parameters. The *Cardiovascular Mean*

arterial pressure parameter (top) has the score value *0* which appears as a "*No Hypotension*" Radio Button in the Nursing app (top left) whereas the Nursing Calculator app (top center) uses a different GUI element, Spinner, labeled "≥ *70 mmHg*", while the SOFA score app (top center) uses a different label, "*MAP* ≥ *70 mmHg*"; all for the same parameter value. To address this challenge, we first extract all the GUI information from the apps automatically using DroidBot (to be introduced later) and then we create a mapping from the heterogeneous, free-text GUI to the reference table entry. We use semi-supervised clustering, i.e., we manually labeled the first points, and when analyzing a new app, the new text is binned into the cluster containing the most similar text. Each of these clusters' title represents the reference table entry – typically an interval or parameter. We defined reference table entries manually (right side of Figure 5); text clusters (shown to the left of reference entries) are populated automatically using edit distance as a similarity metric. In Figure 5 (top right), we show how different phrasings of *Cardiovascular Mean arterial pressure* parameter are mapped. First, the text clusters are formed from the phrasings (*No Hypotension*, ≥ *70 mmHg*, *MAP* ≥ *70 mmHg*), which are then mapped to the same cluster, whose title is '*MAP* ≥ *70 mmHg*'. Similarly, the *Creatinine* parameter (bottom) poses the challenge of heterogeneous GUIs: note how the same score value, *1.2-1.9*, is expressed in four different ways in four different apps. In this case, the text cluster entries are mapped to the reference entry '*1.2-1.9 (110-170)*'. We separate reference table entries by units, e.g., mg/dl were not mixed with μmol/l; this was necessary only for the SOFA score.

3.3.2 Background (DroidBot). DroidBot [38] is a tool that facilitates test automation for Android apps. DroidBot allows users to customize the app testing/exploration strategy by specifying input events for certain app states. DroidBot models the app as a finite state machine and is driven by a specification consisting of States (e.g., a particular screen), Views (specific GUI elements), and Operations (action to perform on a View, e.g., click, swipe, or enter text). A JSON script specifies the input events to generate for each state transition. For example, the script might specify that when the app is in a particular state, DroidBot should generate a specific sequence of taps, swipes, or other input events to simulate a user interacting with the app. DroidBot can also monitor app behavior. We use DroidBot to extract the GUI and automate the exploration, e.g., systematically exploring Radio Button elements to select an interval, clicking the 'Calculate Score' button, and retrieving the resulting score.

3.3.3 Leveraging DroidBot. Our approach uses DroidBot to collect score-relevant GUI information dynamically, i.e., while the app runs. Note that, for practical reasons, dynamic analysis is required for GUI extraction, because static GUI information alone (e.g., from app resources) is incomplete: extracting the GUI at runtime, when all the GUI View objects have been instantiated, is more effective. DroidBot's GUI model helps automatically identify various View objects (such as Radio Button or TextView) related to target score calculation. To trigger the necessary events for completing the score calculation procedure (*GUI Interaction* in Figure 4), we wrote custom DroidBot scripts that automatically activate GUI elements. For example, the scripts automatically trigger different user inputs such as choosing from RadioButtons and selecting from Spinner items. Note that score calculators appear in different Android activities in different apps. We manually directed DroidBot to the activity that corresponds to the score of interest (this is the second of the two manual steps of our approach; the manual effort could be avoided with more engineering, which we leave to future work). In the course of dynamic analysis, i.e., when using DroidBot scripting to populate different score calculations automatically, the states and transitions, e.g., events, are recorded into separate JSON files. These JSON files contain the necessary dynamic information required to construct the GUI specification, as explained next.

3.3.4 Constructing GUI Specification and Finding GUI Specification Errors. Algorithm 1 presents our GUI specification construction and verification approach. The FindCalculatorErrorsViaGUISpecification algorithm takes the target score system's correct reference GUI specification, the DroidBot-generated JSON files from the target APK as input, and outputs the GUI specification errors found.

Algorithm 1 Constructing the GUI Specification and Finding GUI Specification Errors

Input: ReferenceGUISpecification (Correct specifications from the score reference table)
JSON files (DroidBot-generated GUI information from the target APK file)
Output: GUI specification errors

```
1:  procedure FindCalculatorErrorsViaGUISpecification(ReferenceGUISpecification, JSON files)
2:      for each JSON file F in JSON files do
3:          ExtractedIntervals ← []
4:          for each view objects V in F do
5:              IsScoreRelevantParameterOptionCheck(V)
6:              RunDFSToGetParameterValueOptions(V)
7:              IntervalText ←ProcessParameterGUIText(V["text"])
8:              MappedInterval ←MapExtractedIntervalToCluster(IntervalText)
9:              ExtractedIntervals.add(MappedInterval)
10:         end for
11:         FindGUISpecificationMismatch(ReferenceGUISpecification, ExtractedIntervals)
12:         CheckCoverageViolation(ExtractedIntervals)
13:         CheckOverlappingViolation(ExtractedIntervals)
14:     end for
15: end procedure
```

For a given app, our toolchain explores its score-relevant screens (aka activities) using DroidBot, generating multiple JSON files. These JSON files are then fed to our validation method to check for GUI specification errors. We perform a DFS search to find and save all the View objects representing score parameters and their values (lines 5,6). The GUI text, e.g., *"dopamine ≥ 5 mg/kg/min"* is extracted (line 7). Then, we map the extracted text to its corresponding reference entry using the semi-supervised clustering approach discussed in Section 3.3.1 (line 8). The mapped values will be used as final extracted intervals (line 9). Finally, GUI specification discrepancies (*Inconsistent GUI errors*) are found by comparing correct reference parameter value options and extracted parameter value options from the app (line 11). The extracted GUI intervals are checked for coverage violations (line 12) and overlapping violations (line 13) via Z3 (Sections 3.1.2 and 3.1.3).

3.3.5 Finding Calculation Errors Via GUI Exploration. We also check the score resulting from GUI interaction against the formal specification; in other words, we check the validity of the app-computed final score, given the selected parameter values, w.r.t. the reference table's score calculation. This step discovers *incorrect score* errors.

Algorithm 2 shows our approach to finding score calculation errors. The FindCalculatorErrorsViaGUIExploration algorithm takes the DroidBot-generated JSON files and the reference GUI specification as input. We first check whether the View object corresponds to a score parameter option or interval (line 5). If the View object is a relevant score parameter option, then all the attributes and values of that parameter are extracted using DFS (lines 6,7). Next, on lines 8 and 9, we find which score parameter option is selected (set to true for spinners) or checked (set to true for radio buttons). By looping through all the score parameters all their selected values are saved and passed as input to GetReferenceScore along with the reference GUI specification (line 12). This method calculates the expected correct score by mapping each selected option to its correct value and then simply performing an addition to get the total score. We extract the app-generated score via GetAppScore method (line 13) using a similar JSON file View object analysis and text processing approach. We show an example of score value extraction in Figure 6. The figure shows three different apps, where the computed score appears in three different forms. Finally, we compare the two scores: the app-calculated score and the expected correct score, to find any existing score-related error (line 14).

Algorithm 2 Finding Calculation errors via Automatic, DroidBot-driven GUI Exploration

Input: ReferenceGUISpecification (Correct specifications from the score reference table)
JSON files (DroidBot-generated GUI information from the target APK file)
Output: Score calculation errors

```
1:  procedure FindCalculatorErrorsViaGUIExploration(ReferenceGUISpecification, JSON files)
2:      for each JSON file F in JSON files do
3:          SelectedParameterValues ← []
4:          for each view objects V in F do
5:              IsScoreRelevantParameterOptionCheck(V)
6:              RunDFSToGetParameterAttributes(V)
7:              ParameterValue ←ProcessParameterGUIText(V["text"])
8:              if V["checked"]==true OR V["selected"]==true then
9:                  SelectedParameterValues.add(ParameterValue)
10:             end if
11:         end for
12:         CorrectScore ←GetReferenceScore(SelectedParameterValues, ReferenceGUISpecification)
13:         AppScore ←GetAppScore(V)
14:         CheckScoreValidity(CorrectScore, AppScore)
15:     end for
16: end procedure
```

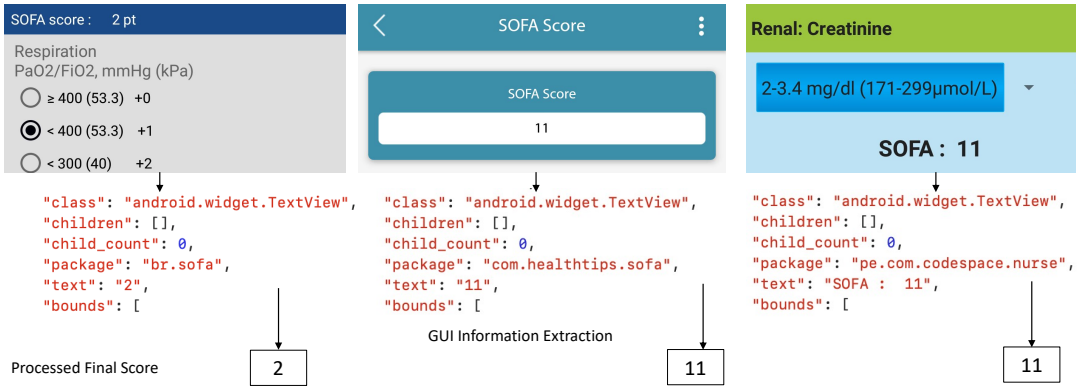


Fig. 6. Score extraction from heterogeneous GUIs: Blue Rock SOFA (left); SOFA Score (center); Nursing (right).

3.3.6 Comparison With Existing Dynamic Analyzers. Our work makes two key advances that permit rigorous, automatic score verification: extracting a formal interval semantics from GUIs, and systematic GUI exploration. We illustrate these advances by comparing with existing dynamic analyses.

GUI Specification Extraction. Although traditional dynamic analyzers [37] can extract GUI information from apps, such as button size and word count, they do not capture semantics. Furthermore, they are not suitable for dealing with heterogeneous GUIs as discussed in section Section 3.3.1. Our approach solves these issues by accurately mapping diverse textual representations of GUI specifications to a reference entry, using interval semantics. These intervals are then used as inputs to Z3 for coverage and overlap error-checking.

Figure 7 shows how our approach maps GUI elements to intervals for the *Creatinine* parameter in the br_SOFA app (top-right part of the figure). Additionally, we separate the extracted numeric intervals based on the different

units present for an attribute (e.g., mg/dL, μmol/L) for error-checking. These interval-based specifications can be encoded into Python and passed to Z3 for error checking (Section 3.2). In contrast, prior GUI extractors do not map GUI elements to an abstract semantics but rather produce lexical or layout metrics, such as the number of words and text spacing, as shown on the bottom-right part of the figure.

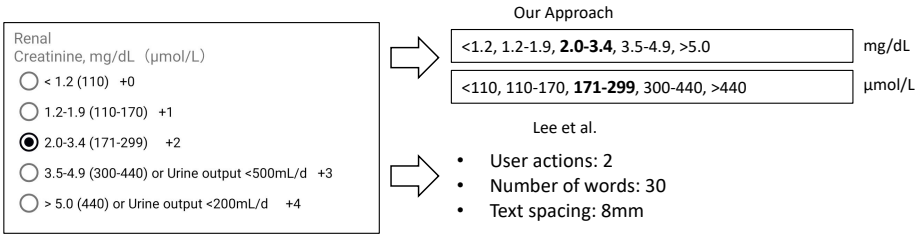
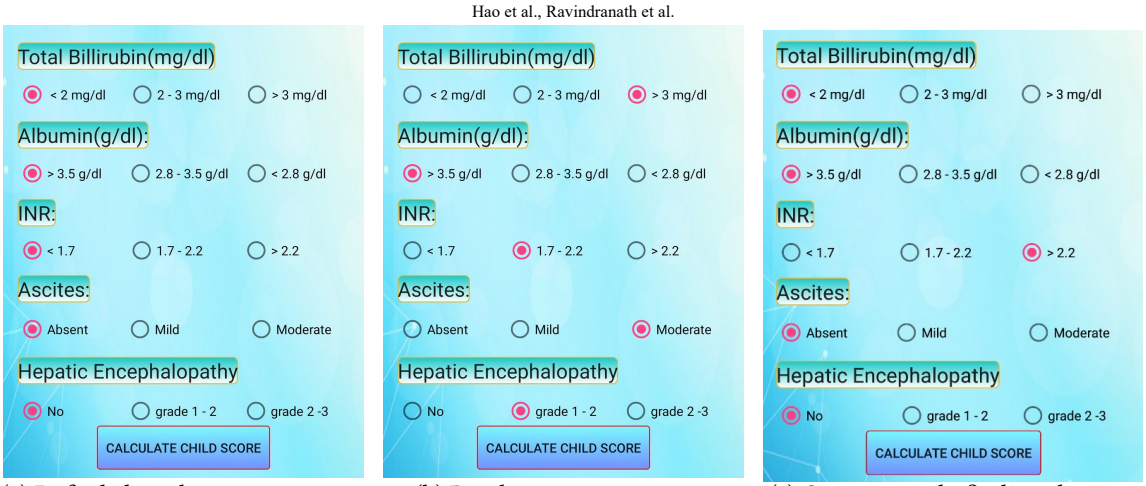


Fig. 7. GUI of br_SOFA app (left); extracting an interval-based semantics (top-right); prior GUI extraction approaches (bottom-right).

GUI Exploration. Traditional dynamic analyzers typically explore the app based on either fixed static GUI states [37] or random GUI states (explored via Android Monkey [29, 48]). However, the lack of systematic exploration makes these approaches inadequate for identifying calculation errors that only manifest in a specific GUI state, that typically differs from the default GUI state.



(a) Default-based strategy (b) Random strategy (c) Our approach, finding the error

Fig. 8. GUI exploration of Child-Pugh Score (KSoft Apps).

In Figure 8 we illustrate the inadequacy of these approaches and how our approach discovers an actual error in the Child-Pugh Score app. The error only manifests when the GUI state is exactly as shown in Figure 8(c), e.g., the attribute *INR* value is "> 2.2".

Defaults-based approaches use the statically-defined default GUI settings, so the GUI setting that induces the error is not explored (Figure 8(a)). Random-based approaches simply conduct a single random sampling of GUI

Table 1. Medical scores analyzed, the year scores were introduced, errors found, score ranges, and action thresholds.

	Score Name	Year	Errors		Range	Thresholds
			Interval/Value Not Covered	Overlap		
Error Found	SOFA *(11,162 citations)*	1996	Bilirubin=12.0 Creatinine=5.0	Bilirubin ([102-204], [<204])	0-24	≥ 11: "higher mortality rate" [23]
	APACHE II *(21,863 citations)*	1985	Age=44		0-71	25: "predicted mortality of 50%" ≥ 35 "predicted mortality of 80%" [16]
	HEART *(582 citations)*	2008		Age ([45-65], [≥ 65]) Troponin (≤normal limit, 1x normal limit)	0-10	4-6:"cannot discharge; admit for clinical observation, noninvasive investigation" ≥ 7: "early aggressive treatment including invasive strategies" [50]
	Pulmonary Asthma Score *(157 citations)*	2002	Resp. rate (<6 yrs)=30 Resp. rate (≥6 yrs)=20		0-9	9 "severe exacerbation" [51]
	RAPS "retold" *(357 citations)*	2004	Respiratory rate=5 Heart rate=39 Mean arterial press.=49		0-20	≥ 7 "increased mortality" [43]
No error found	MEWS	2006			0-18	≥ 4:"surgical team should be informed immediately" [26]
	NEWS	2012			0-18	Any value of 3 in a parameter: "urgent ward based response" 5 or 6: "key threshold for urgent response" ≥ 7: "urgent or emergency response" [17]
	NEWS2	2017			0-21	5 or 6: "patient should be monitored" ≥ 7: "urgently inform a clinician competent in the assessment of acutely ill patients" [42]
	Child Pugh	1973			0-15	8-10: "increased mortality" ≥ 11:"hepatic failure" [55]
	HAS-BLED	2010			0-9	≥ 3 "high risk of bleeding" [39]
	CHA2DS2VASc	2010			0-9	≥ 2 "high risk of stroke and thromboembolism" [40]
	Glasgow Coma Scale	1974			3-15	13-15: "mild neuroemergency " 3-5 "mortality is high and long-term neurological outcomes are generally poor" [12]

settings, as illustrated in Figure 8(b). This has an exponentially low chance of "stumbling" upon the error, as it would require the random configuration to land in the exact GUI state shown in Figure 8(c).

In contrast, our approach (Figure 8(c)) first reveals a specific attribute for which there exists a discrepancy with the reference score (in this example, the *INR* option parameter "> 2.2" mismatches with the reference specification "> 2.3"). Next, our DroidBot-based exploration (described in Section 3.3.5) reaches that specific GUI state (*INR* value "> 2.2") and generates the overall score. Finally, when our approach compares the app-generated score with the expected reference score, the calculation error is revealed.

4 CHECKING REFERENCE SCORES

We evaluated our approach on 12 medical scores. We focused on scores used in critical settings, where errors have serious implications. The scores, ranges, potential errors and action thresholds are shown in Table 1. We first discuss scores' nature, argue why score accuracy is critical, and then present the errors we found.

4.1 Reference Scores

We only provide details and citations numbers (Table 1, first column) for those scores that contain errors.

The *SOFA* (Sequential Organ Failure Assessment) score predicts the mortality rate of an ICU patient based on the functionality of six organ systems. The score is updated and calculated every 24 hours until the patient is discharged [53]. *APACHE II* (Acute Physiology and Chronic Health Enquiry II) is used to provide a general measure of disease severity while taking into account current measurements, age, and health history [35]. The *HEART* (History, EKG, Age, Risk Factors, Troponin) score is used to predict the risks of a major cardiac event while taking into account risk factors from a patient's history or age and other parameters [50]. The *RAPS* (Rapid Acute Physiology Score) predicts patient mortality in critical care transport [49]. The *Pulmonary Asthma Score* was developed to simplify children's asthma severity calculation [51].

The remaining scores do not contain errors (though apps implementing the scores do); the scores' domains are: identifying the severity of patients' conditions in critical care (*MEWS* [26], *NEWS* [17], *NEWS2* [2]); chronic liver disease severity (*Child-Pugh* [55]); risk of bleeding (*HAS-BLED* [39]); stroke risk (*CHA2DS2VASc* [40]); and severity of a brain injury (*Glasgow Coma Scale* [52]).

4.2 Why is Score Accuracy Critical?

We chose these scores because they capture critical conditions, where action is urgently needed. Errors in the app-calculated scores can result in under-estimating the real score, i.e., patient state is more critical than the app indicates, which potentially means that time-critical life-saving actions will not be taken. Conversely, errors that result in the app over-estimating the real score might lead to an overly aggressive, disproportionate intervention, as well as unnecessary use of resources (personnel, ICU beds, etc.).

For each score, Table 1's second-to-last and last columns show the range of possible values and threshold values, respectively; the third and fourth columns show errors (if any) and will be discussed in Section 4.4. The threshold column is particularly revealing, as it indicates the score value(s) at which a more aggressive intervention is warranted, or values where the prognosis turns dim. For example, for the HEART score, a patient who "scores" ≤ 3 can be discharged; a patient who scores 4–6 would be admitted for noninvasive investigation; whereas a patient who scores ≥ 7 will receive "early aggressive treatment including invasive strategies". Hence a score calculation error *at* or *around* the threshold value is particularly concerning.

4.3 Specification Extraction Accuracy

We measured the accuracy of specification extraction in terms of true positives (reference table entries that should be checked for coverage and non-overlap), true negatives (entries that should not be checked), false positives (entries that should not be checked, but our approach does check), and false negatives (entries that should be checked but our approach does not check). Specifically, the results obtained via the automated analysis (A) described in Section 3.2, were checked and cross-referenced by three human analyzers (H1, H2, H3) as follows.

Human analysis. To establish ground truth, all the reference tables were verified manually by three human analyzers, H1, H2, and H3. H1/H2/H3 analyzed each table and individually recorded any observed errors, i.e., violations of coverage or non-overlap. Next, each human analyzer compared their findings against the errors reported by the automatic analysis (A) and recorded TP/FP/TN/FN. Finally, a cross-checking was performed to measure agreement. The observed multiple-rater agreement across H1, H2, and H3's findings was 100%.

Age		Troponin	
	≤65 year		>2x normal limit
	45-65 year		1-2x normal limit
	<45 year		≤normal limit

(a) HEART score has two non-overlap violations: *Age=65* and *Troponin=(1x) normal limit.*

Add 0 points for the age <44.2 points. 45–54 years:

(b) APACHE II has a coverage violation for *Age=44.*

	Respiratory Rate (breaths/min)	
Score	<6 Years	≥6 Years
0	<30	<20
1	31–45	21–35
2	46–60	36–50
3	>60	>50

Missing value 30

Missing value 20

(c) Pulmonary Asthma Score has two coverage violations for *Respiratory Rate.*

Physiological variable	+2	3+	+4
Body temperature	32–33.9	30–31.9	<30
Mean arterial pressure	50–69		<49
Heart rate	55–69	40–54	<39
Respiratory rate	6–9		<5

(d) RAPS "retold" has three coverage violations for *Respiratory rate, Heart rate, and Mean arterial pressure.*

Fig. 9. Reference tables with no straightforward fixes.

Results. Across all tables, there were 418 rows, of which 319 should be checked for. The resulting confusion matrix was:

True Positives: 318	**False Positives**: 0
False Negatives: 1	**True Negatives**: 99

i.e., 100% precision and 99.69% recall. The false negative was in the APACHE II score, where the error was hidden in a supplementary, nonstandard, age adjustment footnote (relevant excerpt shown in Figure 9b).

4.4 Inconsistent Reference Table

We found errors in the original reference tables for 4 of the 11 scores; we also found errors in one score as defined in follow-up work to the original reference table; these errors are shown in the top part of Table 1.

For SOFA, as discussed in Section 2.3, the partition condition (1) coverage, is violated for *Bilirubin=12.0* and *Creatinine=5.0*; these values do not appear in the table though values lower or higher do appear in the table (Figure 1). The second issue for SOFA was a violation of the partition condition (2) non-overlap, where multiple table entries satisfy *Bilirubin* < *204*. While the latter issue might be alleviated if an app/medical system does not use the μmol/l units, it is unclear how an app developer is supposed to deal with the former issue: should the 12.0 and 5.0 values be included into the left or right cells in the table?

The HEART score's reference table (relevant excerpt shown in Figure 9a) violates the non-overlap condition at two points: *Age=65* and *Troponin=normal limit*; possible resolutions include changing *Age*≤*65* to *Age*>*65* and *Troponin:* ≤*normal limit* to *Troponin:* <*normal limit.*

The APACHE II reference table [35], dating back to 1985, violates the coverage condition for *Age=44.* This table would be particularly challenging to verify manually as it has 117 entries (13 rows by 9 columns).

The Pulmonary Asthma Score's reference table (relevant excerpt in Figure 9c) violates coverage at two points: *RespiratoryRate=30* and *RespiratoryRate=20*; it is unclear how an app developer is supposed to cope with these, and whether the scores for those values should be 0 or 1.

The "RAPS retold" score was an interesting find. Note that the original RAPS score, introduced by Rhee et al. [49], does not violate the partition conditions. A new score, REMS, was introduced by Olsson et al. [43] to

improve upon RAPS; the paper presents both scores, but the "retold" RAPS table (Figure 9d) has three coverage violations, as shown in Table 1.

4.5 Correcting the Specification

Although we have found faulty reference tables, they do not introduce false positives, as we first translate such tables into correct ones (partition-wise), then compare apps' GUI specifications against corrected tables. We now discuss approaches for, and challenges associated with, fixing the incorrect tables.

Straightforward case. One of the specification errors found in the SOFA reference table (Figure 1) for Bilirubin, originally *'<204'*, can be easily fixed by changing it to *'> 204'*.

"Reasonable" fix. In addition, the SOFA reference table has no coverage for *Bilirubin=12.0 mg/dl*. However, by observing that $204\mu mol/l = 11.93mg/dl$ we could infer a reasonable fix, that is, to change '> 12.0' to '≥ 12.0'. Similarly, *Creatinine=5.0 mg/dl* is not covered, but since $440\mu mol/l = 4.98mg/dl$, a reasonable fix would be to change '> 5.0' to '≥ 5.0'.

4.5.1 Challenging Cases. In the remaining cases (Figure 9), it is unclear how to "fix" the specification. To fix the HEART reference table (Figure 9a), *Age=65* can be assigned a score of either 2 or 1. In fact the paper itself is ambiguous as it says "one point if the patient was between 45 and 65 years and two points if the patient was 65 years or older" [50]. Similarly, in the case of the Pulmonary Asthma score (Figure 9c), *RespiratoryRate=30* and *RespiratoryRate=20* can be assigned score values of either 0 or 1. For the APACHE II reference table (Figure 9b), *Age=44* can be assigned 0, 1, or 2 points. Finally, for RAPS retold (Figure 9d), different score interpretations can be made for *RespiratoryRate=5*, *HeartRate=39*, and *MAP=49*.

How developers cope with faulty scores. We found 14 instances where developers attempted to correct a faulty score when implementing that score in their apps. For SOFA, we found 4 apps that performed the straightforward *'> 204'* fix, and 4 apps that used the reasonable '$Bilirubin \geq 12.0$' and 'Creatinine ≥ 5.0' fixes. For APACHE II, 4 apps performed an ad-hoc fix (changing $Age < 44$ to $Age \leq 44$). Finally, for the HEART score, which has overlap errors at *Age=65* and $Troponin \leq normal\ limit$, no app has fixed the overlap errors; moreover, three apps have introduced additional overlap errors at *Age=45*.

We can now summarize our RQ1 findings.

RQ1: Can our approach extract and verify medical score specifications?
Answer: Yes. Specification extraction has a 99.5% F1-score and verification has uncovered all 11 violations in the 5 incorrect scores.

5 FINDING ERRORS IN APPS

We have evaluated our approach on a dataset of 90 apps; the selection process is explained next, followed by a discussion of the errors we found, effectiveness, and efficiency.

5.1 App Dataset

We selected our apps from the Medical category on Google Play. We scraped 3,762 apps and their descriptions from Google Play; using ranked retrieval, we identified 556 apps classified as medical calculators. We then focused on apps which computed one or more among the 12 scores we verified, resulting in a total of 90 apps.

5.2 App Errors: Inconsistent GUI

We now present our findings: coverage violations and non-overlap violations.

Table 2. Inconsistent GUI: coverage violations.

App Name	Score	Parameter value(s)
Sepsis Clinical Guide	APACHE II	Hct(%) =60
Child Pugh Calculator	Child Pugh	INR =1.7
Child-Pugh Score (Blue Rock)	Child Pugh	INR < 1.7, INR > 2.2
HAS-BLED Score	HAS-BLED	Age=65
Nursing Calculator	MEWS	Systolic BP=70, Resp=8, Temp=35.0
Quick EM	SOFA	PaO2 ≥ 400, Dopamine=5
Nursing	SOFA	Bilirubin=12.0, Creatinine (mg/dl)=5.0
SOFA Score (widebitsbd)	SOFA	GCS=15, PaO2 ≥ 400, Platelets ≥ 150, Creatinine (mg/dl) <1.2
SOFA Score (Blue Rock)	SOFA	Bilirubin=12.0, Creatinine (mg/dl)=5.0, Dopamine=5
Merck Manual Professional	SOFA	Bilirubin=12.0, Creatinine (mg/dl)=5.0
SOFA	SOFA	Bilirubin=12.0, Creatinine (mg/dl)=5.0

Table 3. Inconsistent GUI: non-overlap violations.

App Name	Score	Overlapping Ranges
Child-Pugh Score	Child-Pugh	INR: [>1.7], [1.7-2.2]
HEART Score	HEART	Age: [≤45], [45-65], [≥ 65] Troponin: ≤normal limit, 1-3x normal limit, ≥3x normal limit
HEART Score Calculator	HEART	Age: [45-65], [≥ 65] Troponin: 1-3x normal limit, ≥3x normal limit
Quick EM	HEART	Age:[45-65], [≥ 65], Troponin: ≤normal limit, 1-3x normal limit
HEART Score Gumption	HEART	Age: [45-65], [≥ 65] Troponin: ≤normal limit, 1-3x normal limit, ≥3x normal limit
REBELEM	HEART	Age: [≤45], [45-65], [≥ 65], Troponin: 2-3x normal limit, ≥3x normal limit
Medical Calculators	HEART	Age: [≤45], [45-65], [≥ 65] Troponin: ≤normal limit, 1-3x normal limit, ≥3x normal limit
MEWS	MEWS	Heart Rate: [51-101], [101-111], Respiratory rate: [15-21], [21-30] Systolic BP: [71-81], [81-101], Temperature: [≤35], [35-38.5],[≥38.5]
Nursing Calculator	SOFA	Bilirubin: [1.2-1.9], [1.0-5.9], Creatinine (mg/dl): [1.2-1.9][1.0-3.4] Platelets: [≥ 150], [100-150]
SOFA Score (Blue Rock)	SOFA	Creatinine (μmol/L): [110], [110-170],[300-440],[440]
SOFA	SOFA	Norepinephrine: [≤0.1], [<0.1]
Sepsis Clinical Guide	SOFA	Creatinine (mg/dl): [≤1.2],[1.2-1.9], Platelets: [≥ 150], [≤ 150]

5.2.1 Coverage violations. Table 2 shows the results; we found 23 coverage errors in 11 apps. The first column shows the official app name on Google Play, the second column shows the affected score calculator, while the third column shows values or ranges that are not covered. Interestingly (though somewhat predictably), the "original sin" in the SOFA reference table (no coverage for *Bilirubin=12.0* and *Creatinine=5.0*) leads to non-coverage issues for those parameter values in four apps.

5.2.2 Non-overlap violations. Table 3 shows the results; we found 32 non-overlap errors in 12 apps. The parameters with overlapping ranges are shown in the third column. In this case, all the *HEART* score apps' errors appear attributable to the error in the original HEART reference table (Table 1).

Table 4. Calculation errors in apps.

App Name	Score	Calculation Errors			Meets or Exceeds Threshold
		Parameter Option	App Score	Ref. Table Score	
Atrial fibrillation risk calc	CHA2DS2VASc	Age=75	1	2	N
Child-Pugh Score (KSoft Apps)	Child Pugh	INR=2.3	11	10	Y
Child-Pugh Score (Blue Rock)	Child Pugh	INR=2.3	11	10	Y
Child-Pugh Score (Liver)	Child Pugh	INR=2.3	11	10	Y
Nursing Calculator	MEWS	Heart rate=40	3	4	N
MEWS	MEWS	Temperature=[35.1-36]	3	4	N
Nursing Calculator	SOFA	Platelets=150	12	11	Y
Sepsis 3	SOFA	Dopamine=5	12	11	Y
Nursing	SOFA	PaO2=300	12	11	Y
MediCalc	SOFA	Dopamine=5	12	11	Y
SOFA Score (widebitsbd)	SOFA	Creatinine (μmol/L)=106	12	11	Y
Merck Manual Professional	SOFA	PaO2=300	12	11	Y
SOFA	SOFA	Creatinine(μmol/L)=106	12	11	Y
SOFA - (Sepsis)	SOFA	Dopamine=5	12	11	Y
Sepsis Clinical Guide	SOFA	Bilirubin=1.2	12	11	Y
Sepsis SOFA Calculator	SOFA	Platelets=20	4	3	Y

Table 5. Apps fixed thanks to our reporting.

App Name	Package Name	#Installs
Nursing	pe.com.codespace.nurse	100,000
MEWS Brasil	appinventor.ai_blinkeado.InformaticasaudeMEWS	500
Atrial fibrillation risk calc	com.gumptionmultimedia.atrialfibrillationriskscore	5,000
Nursing Calculator	com.niya.lijo.nursingcalculators	50,000
Sepsis Clinical Guide	app.escavo.sepsis	100,000
SOFA	gumptionmultimedia.com.sofascore	1,000

5.3 App Errors: Incorrect Score Calculations

Table 4 shows errors in score calculation; we found 16 calculation errors in 16 apps. The *Calculation Errors* grouped columns show the parameter values for which the errors manifest, and the app value vs. reference score value. We make several observations. First, app errors lead to both under-estimating the true score (e.g., apps Atrial fibrillation risk calc, MEWS, Nursing Calculator-MEWS) and over-estimating the true score (e.g., apps Sepsis3 or MediCalc). Both error types are problematic due to potential under-intervention and over-intervention respectively, as explained in Section 4.2. Second, as the last column indicates, certain errors "straddle" the threshold, which, as discussed previously, can put the patient in a different class.

We have reached out to the developers of apps where we found errors. So far, 6 apps (listed in Table 5) have been fixed and updated.

5.4 Effectiveness

We measured the accuracy of our automated app analysis via a process similar to the one described in Section 4.3. Specifically, the results obtained via the automated analysis (A) described in Section 3.3 were checked and cross-referenced by three human analyzers (H1, H2, H3) as follows.

5.4.1 Human Analysis. To establish ground truth, all the apps were first explored manually by three human analyzers (H1, H2, H3). The analyzers performed several tasks: (a) ensure that the app implements a score that was among our examined scores, (b) find the app activity implementing the score (which formed the start of the automated analysis), (c) check for inconsistent GUIs, and (d) exercise all GUI options and check for score calculation errors. Next, each human analyzer compared their findings against the errors reported by the automatic analysis (A) and recorded TP/FP/TN/FN. A cross-checking was performed to measure multiple-rater agreement among the errors discovered. The observed multiple-rater agreement across H1, H2, and H3's findings was 100%. A fourth human H4 (the research lead) performed a final cross-check between the manual and automatic analysis results. Note that the manual app analysis was a considerable task, due the large space induced by the number of apps, number of scores, and manual GUI interaction; overall the task took H1, H2, and H3 about four person-months.

5.4.2 Results. The confusion matrix resulting from comparing the automated analysis findings with ground truth was:

Table 6. Results.

True Positives: 20	**False Positives**: 0
False Negatives: 5	**True Negatives**: 65

The false negatives are due to apps using WebView (Section 6.5). These figures, a 100% precision and 80% recall, are par for the course for a dynamic analysis.

5.5 Efficiency

Table 7 provides details on the efficiency of our approach. The median size of app bytecode alone (`.dex`) was 3.8MB; note that app size (`.apk`) would be much larger as that includes app resources. A typical app takes about 3 seconds to analyze. Reference table verification, including running Z3, took less than 1 second for any table; for any app, GUI extraction followed by verification/validation took at most 2 seconds.

Table 7. Efficiency results.

Analysis time (seconds)			**Bytecode size (MB)**		
min	max	median	min	max	median
2	3	3	0.17	125	3.8

We can now summarize our RQ2 findings.

RQ2: Is our approach effective at analyzing and finding errors in real-world apps?
Answer: Yes. The analysis could tackle all 90 real-world apps, achieving 100% precision and 80% recall, and taking 3 seconds per app on average.

6 DISCUSSION

We now interpret and analyze the findings presented in this study, provide insights into their implications, offer recommendations for regulatory bodies, discuss limitations, and suggest avenues for future research.

6.1 Interpretation of Our Results

6.1.1 Multiple Numeric Ranges in Scoring Systems Invite Errors. Our study demonstrates that medical scores with numeric range-based parameters are more susceptible to coverage and non-overlap issues compared to scores that solely rely on text-based parameters ,such as the GCS score. For instance, the options for the *Motor Response* parameter in the GCS scoring system are text-based, such as *"Makes no movements"*, *"Abnormal extension"*, *"Abnormal flexion"*, *"Flexion"*, hence mutually exclusive by default. This characteristic makes the GCS scoring system immune to coverage and non-overlap violation errors. In contrast, a numeric range-based scoring system like SOFA offers options such as *"< 1.2"* or *"1.2–1.9"* for the *Bilirubin* parameter.

Scores that use at least two numeric ranges in their parameter specifications (e.g., SOFA or APACHE II) invite errors because at least two intervals are needed for coverage or non-overlap issues. In contrast, scores such as HAS-BLED involve zero or one interval, hence have no coverage or non-overlap errors.

However, we found no correlation between the total number of parameters and the probability of error in a scoring system. For example, the HEART score has fewer parameters compared to MEWS, yet the former has a specification error while the latter is error-free.

6.1.2 Slider-based Input UIs can Prevent Errors. To mitigate coverage and non-overlap errors, app developers should consider user input via sliders, instead of numeric range-based CheckBox or RadioButton View objects. For example, Android offers a continuous RangeSlider UI input object [8] that allows developers to create input ranges by only defining valueFrom and valueTo attributes without the need of labeling intermediate intervals or ranges; in other words, users select a single value in the interval [valueFrom,valueTo]. Using such a continuous RangeSlider avoids coverage issues (gaps) thanks to the continuous natures of the UI control. Moreover, sliders prevent the repetition of the same numeric input value, hence avoiding non-overlap errors. For example, the coverage issue (e.g., missing value 12.0) found in SOFA calculator apps can be easily avoided using a continuous RangeSlider.

6.1.3 Lack of Transparency and Documentation. Another critical issue we uncovered is the insufficient transparency and documentation pertaining to the references used by medical score calculator apps. In cases where there were no straightforward fixes for reference table errors (Section 4.5), 8 out of 14 apps failed to provide comprehensive information regarding the data sources underlying their chosen fixes. This lack of transparency poses challenges for (a) healthcare professionals in assessing the validity and reliability of these apps, hindering their ability to make informed judgments, and (b) software verification, because there is no specification to compare the app against.

6.2 Implications

6.2.1 Research Community. As shown in Section 5.2, an incorrect table entry can propagate to apps, as developers are forced to either offer a literal translation of the incorrect score into the GUI (which then confuses users), or provide their own, ad-hoc interpretation of what a "correct" entry should be. The extent of the problem is larger, unfortunately. Errors in the original version of a score have multiple negative ramifications, as these errors will propagate not only to apps, but to reference manuals, medical reference websites, etc. The errors will first confuse developers, then users, and ultimately negatively affect care outcomes. There are preventive and curative approaches for eliminating such errors. First, it is imperative that the research community perform a stricter scrutiny of reference tables at peer review time, to correct any error before publication. Second, along the lines of

Section 4, larger-scope studies should be performed on a broader range of existing, published reference tables, leveraging automatic reasoning to minimize human effort and ensure scalability.

6.2.2 Clinical Practice. The findings of this study have important implications for healthcare professionals relying on medical score calculator apps in their clinical practice. The inaccuracies, inconsistencies, and lack of transparency in these apps underscore the need for caution when utilizing them for clinical decision-making. Healthcare providers should be aware of the limitations and potential risks associated with these apps and consider multiple sources of information, or manual cross-checking, to validate the results obtained. Additionally, healthcare institutions should establish guidelines and standards for the development and evaluation of medical score calculator apps to ensure their accuracy, reliability, and adherence to evidence-based medicine.

6.3 Generalizability of Proposed Solution

We have observed partition violations or incorrect score implementations in web-based medical score calculators [4, 7], and partition violations non-medical Android apps, e.g., IELTS score conversion [3]. The root cause of such issues is that current software development tools – targeting mobile, desktop, or Web platforms – fail to perform a partition check on GUI elements. These unchecked partition errors lead to user confusion and ultimately incorrect outcome. The GUI partition checks could be made mainstream and performed at app compile time: for mobile, in Android Studio or Apple Xcode; for Web, in front-end frameworks; for desktop, in the GUI designer/specification module of desktop IDEs.

6.4 Recommendations for Regulatory Bodies and App Stores

A 2019 literature review reveals that many consumer health apps have false content, poor design, and bad functionality [10]. However, score calculator apps are currently outside the purview of regulatory agencies, as explained next. For example, in the US, the FDA has jurisdiction over certain categories of apps pursuing to the Federal Food, Drug, and Cosmetic (FD&C) Act: apps that turn the phone into a medical device [6], e.g., by connecting to external sensors or medical devices, or apps that provide patient-related analyses for diagnosis and treatment. The US Federal Trade Commission (FTC) is concerned with user privacy and security in medical apps, as well as developers being able to support health claims made in apps with scientific evidence [1]. US HIPAA (Health Insurance Portability and Accountability Act) regulates how "covered entities" (e.g., health plans, providers) handle sensitive private health identifiers (PHI) that could uniquely identify the patient[31, 32]. Hence apps that automate clinical calculations are outside the purview of such regulatory frameworks. We believe that voluntary "self-policing" by app developers is unrealistic. Therefore, there are several possible solutions: (1) regulatory action, e.g., adding score calculator apps and score accuracy to FDA or FTC regulatory frameworks, or (2) app marketplace action – adding score calculator accuracy as a precondition for listing the app on marketplaces such as Google Play or Apple App Store.

6.5 Limitations

Our approach has two small limitations which can be addressed with more engineering. First, DroidBot failed to produce proper JSON GUI data in 5 cases when apps used WebView (i.e., rendering Web content instead of using Android UI elements). Second, we had to correct occasional minor errors in PDF-to-CSV conversion, e.g., SOFA mixes numbers with text for Dopamine, while other scores are purely numeric.

6.6 Future Research

6.6.1 Expanding to iOS and WebView-based Calculators. While this paper focused solely on Android apps, we acknowledge that the iOS versions of our examined apps may contain errors as well. Therefore, our future research aims to extend the applicability of our toolchain to the iOS platform. Additionally, we intend to streamline the

analysis process for apps utilizing WebView (apps where DroidBot encountered difficulties) by reducing the manual effort involved and automating the analysis of such apps.

6.6.2 Evolution of Errors. As shown in Table 1, the papers introducing the reference scores have been cited substantially (from 157 to 21,863 times depending on the score) so their impact is wide-reaching and long-lasting. To gain a comprehensive understanding of the propagation of errors, we plan to conduct a study examining how reference scores evolve from the original, flawed reference. This investigation will offer an evolutionary perspective: how certain errors are corrected, which errors are still preserved, and what kinds of new errors are introduced.

6.6.3 Deploying a Tool for Clinicians. Our automated toolchain holds the potential for practical deployment in at least two settings: by app stores for regulatory checks, and by clinicians for verification purposes. Clinicians can utilize our toolchain to review the correctness of medical score calculators before using them for diagnostic purposes, enhancing the reliability of these applications, and ultimately the safety of clinical care that involves automated score calculators.

7 RELATED WORK

Related work falls into four categories: medical research studies, checking tabular specifications, GUI extraction, automated testing and analysis.

Medical research studies. Bierbrier et al.'s 2014 study [15] tested medical scores (including Child-Pugh and HAS-BLED) and calculations, e.g., BMI. The authors asked 5 physicians to identify relevant scores. Two of their analyzed scores were on our list as well: 5 physicians selected Child-Pugh and 4 physicians selected HAS-BLED as scores of interest. The authors then tested each of the 14 apps (2 Android and 12 iOS) with 10 values: 2 extreme values and 8 middle values. There were errors in two Child-Pugh apps, though they were at the low score ranges hence would not place the patient over the threshold or in a different class. No issues were found with HAS-BLED implementations. Instead of random testing, our approach verifies reference tables and apps automatically, which, aside from rigor, makes the approach more scalable. As Table 2, Table 3, and Table 4 show, we found issues with both these scores, including issues at the threshold. Haffey et al. [28] performed a study on 23 Android opioid conversion apps. Their study revealed two main issues. First, 11 out of 23 apps failed to identify the sources related to their calculations; 12 apps failed to state whether any medical professionals were involved in the app creation. More importantly, these apps' calculations resulted in highly variable results and significantly different outputs across apps. Huckvale et al. [34] studied 46 insulin dose calculator apps. They found that 31 of the apps pose a risk to users due to incorrect dosage calculation. Further issues included lack of disclaimers, lack of input validation, no updates in response to changes in input, etc. Hers et al.'s 2021 study [30] has revealed an inaccurate risk assessment in an aortic surgery risk calculator. They found that the calculator underestimated complications consistently when taking into account patient demographics and data. While the study focused on the overall accuracy of the calculator in a clinical setting, it shows general reliability issues with medical calculators which we also manage to uncover in our study. Akbar et al.'s [11] meta-analysis on 74 app studies (none of which have looked at score calculators, however) has revealed numerous safety concerns, including calculation errors, potentially harmful recommendations, etc. Though somewhat complementary to our work, all these efforts underline the importance of verification and tighter scrutiny in the medical app domain.

Checking tabular specifications. Bultan and Heitmeyer [18], and Lawford et al. [36] also performed disjointness and coverage checks on tabular specifications. Their specifications were written manually, whereas we extract the specification automatically. Our scope, end-to-end automatic reference table and app checking, is completely different from theirs.

GUI extraction. Extracting semantic information from Android app GUIs is notoriously difficult, especially when it has to be done automatically and at scale (Choudhary et al. [19] discuss challenges and some approaches). Abbas et al. [9] extracted text data from medical text images using Optical Character Recognition (OCR). The extracted unstructured text data is then processed to produce relevant medical terms. Guigle [14] builds a searchable index of GUI elements in Android apps. Liu et al. [41] discuss a method for automatically annotating mobile UIs using a lexical database that contains design semantics. The approach involves identifying different components and concepts in UIs by leveraging the vocabulary provided by the database and a set of labeled examples. The method can automatically identify 25 UI component categories, 197 text button concepts, and 99 classes of icons in Android UIs using code-based properties and a neural network combined with anomaly detection. This approach is applicable to example-based UI search that identifies visually similar UI screens. While these OCR, search-engine, and lexical-database approaches provide detailed information about the location and functionality of GUI elements, they cannot be applied to find medical calculator apps' GUI errors. Given our need to map GUI information to specific medical score parameters and interval-based semantics, we decided to build our own clustering and DroidBot-based approach; we are not aware of related work that would subsume our approach (Section 3.3).

Automated testing and analysis. The purpose of AMC [37] is to automatically examine user interface designs of vehicular applications and ensure they meet safety and consistency standards. This tool conducts automated dynamic exploration of mobile applications similar to ours, but it verifies apps based on UI properties such as the number of actions per task (should be less than 10 per screen), text word count (not more than 100 words per screen), text contrast (3:1 contrast ratio between foreground and background recommended), and button size (must be greater than $80mm^2$) and spacing (at least 15mm). On the other hand, our focus is on extracting UI elements involving numeric ranges or intervals, and verifying them for partition conditions. VanarSena [48] uses a dynamic exploration approach to test different fault induction modules on the app, including user input, network, and system state faults. They used "many randomized concurrent monkeys" approach that generates input events to test the apps and a hit testing mechanism to ensure that input events are sent to valid UI elements. The hit testing mechanism is implemented as a tree search algorithm that searches for UI elements at a given position, starting from the top-level UI element. They also used fault injection modules to simulate various user scenarios and test the robustness of the apps. The approach was effective in finding numerous crashes and bugs in apps that are already in the marketplace. In contrast, our approach is targeted at finding incorrect definitions of numeric ranges or intervals. PUMA [29] is a dynamic exploration tool similar to DroidBot, designed for automated testing of mobile applications; it uses Android Monkey-based testing techniques [13] and is programmable to explore the application UI and report any issues or bugs found during the testing process. PUMA requires the application to be instrumented before running the tests. PUMA scripts and apps are input into the tool, and the interpreter instruments the apps to trigger app-specific events. However, random dynamic exploration does not check for, and does not detect, calculation errors as our analysis does. FlowCog[45] uses a combination of static and dynamic analyses to detect information leaks that are caused by improper handling of sensitive data. FlowCog first extracts a context-aware semantics from the Android app's source code and resource files, then uses this information to create a precise model of app behavior, including how it handles sensitive data. Finally, FlowCog performs dynamic analysis to observe the app's actual behavior and compares it against the expected behavior predicted by the model. If the runtime flow deviates from the expected behavior based on the extracted semantics and view dependencies, FlowCog reports it as a potential data leak. In contrast, we extract and verify the GUI specification and report potential errors if the GUI or score result differ from the correct reference specification.

For Android developers creating personal data-handling apps that require accessing sets of interrelated personal data, the Epistenet [20] tool can assist in addressing many of the associated challenges. The use of Epistenet

involves storing personal data in a knowledge graph with a semantic structure, exposing the relationships between the data. As a result, developers only have to interact with a single API. Without Epistenet, personal data is stored in silos, and developers must manually interface with each data provider; in addition, the relationships between data are not evident. Epistenet generates a knowledge graph of personal data, categorizing it using ontologies to establish relationships. Each data piece is represented as an object with attributes and meta-attributes linked to ontology classes. This enables developers to retrieve interconnected personal data easily. In comparison, our clustering approach uses distance metrics to map GUI texts to reference entries, whereas Epistenet relies on manual relationship mapping.

8 CONCLUSIONS

Mobile health apps are seeing increasing adoption in acute care settings, and mobile app developers, including developers *who are not medically qualified*, are eager to capitalize on this growing demand. Though errors in medical scores and score calculator apps can have severe negative consequences, at this point, such scores and apps are subject to no scrutiny. We tackle this issue via rigorous, automated approaches: (1) extracting reference tables into interval-based specifications and checking them for partition violations, and (2) validating apps against the aforementioned specification, as well as verifying app GUIs. We have uncovered errors in long-standing medical reference articles. We found that incorrect specifications translate to incorrect app implementations, and that even correct specifications can be implemented incorrectly, affecting the resulting scores. Our findings indicate a need for tighter scrutiny of reference scores themselves, as well as apps implementing these scores.

ACKNOWLEDGMENTS

We thank Jessica Jeshalkumar Parekh for help with initial app data collection, and the anonymous reviewers for their valuable feedback. This material is based upon work supported by the National Science Foundation under Grant No. CCF-2106710.

REFERENCES

[1] 2021. FTC cracks down on marketers of "melanoma detection" apps. https://www.ftc.gov/news-events/news/press-releases/2015/02/ftc-cracks-down-marketers-melanoma-detection-apps.
[2] 2021. National Early Warning Score (NEWS) 2. https://www.rcplondon.ac.uk/projects/outputs/national-early-warning-score-news-2
[3] 2022. IELTS Band Score. https://play.google.com/store/apps/details?id=com.samir.ieltsmarkstoband. Accessed: 2022-09-01.
[4] 2022. MDApp SOFA Score. https://www.mdapp.co/sequential-organ-failure-assessment-sofa-score-calculator-184/. Accessed: 2022-09-01.
[5] 2022. MEWS Brasil. https://play.google.com/store/apps/details?id=appinventor.ai_blinkeado.InformaticasaudeMEWS.
[6] 2022. Policy for device software functions and mobile medical applications guidance for industry and food and drug administration staff.
[7] 2022. SOFA. https://www.rccc.eu/ppc/indicadores/sofa.html. Accessed: 2022-09-01.
[8] 2023. RangeSlider. https://developer.android.com/reference/com/google/android/material/slider/RangeSlider. Accessed: 2023-07-01.
[9] Asim Abbas, Muhammad Zaki Ansaar, and Sungyoung Lee. 2019. Medical Concept Extraction Using Smartphone and Natural Language Processing Techniques (Poster). In *Proceedings of the 17th Annual International Conference on Mobile Systems, Applications, and Services* (Seoul, Republic of Korea) *(MobiSys '19)*. Association for Computing Machinery, New York, NY, USA, 630–631. https://doi.org/10.1145/3307334.3328661
[10] Saba Akbar, Enrico Coiera, and Farah Magrabi. 2019. Safety concerns with consumer-facing mobile health applications and their consequences: A scoping review. *Journal of the American Medical Informatics Association* 27, 2 (2019), 330–340. https://doi.org/10.1093/jamia/ocz175
[11] Saba Akbar, Enrico Coiera, and Farah Magrabi. 2020. Safety concerns with consumer-facing mobile health applications and their consequences: a scoping review. *J. Am. Med. Inform. Assoc.* 27, 2 (Feb. 2020), 330–340.
[12] Wayne M. Alves, Brett E. Skolnick, Brett E. Skolnick, and Shamik Chakraborty. 2018. *Chapter 5 - Traumatic Brain Injury.* Academic Press.
[13] Android Developers. 2023. UI/Application Exerciser Monkey. https://developer.android.com/studio/test/other-testing-tools/monkey.

[14] Carlos Bernal-Cárdenas, Kevin Moran, Michele Tufano, Zichang Liu, Linyong Nan, Zhehan Shi, and Denys Poshyvanyk. 2019. Guigle: A GUI Search Engine for Android Apps. *CoRR* abs/1901.00891 (2019). arXiv:1901.00891 http://arxiv.org/abs/1901.00891

[15] Rachel Bierbrier, Vivian Lo, and Robert C Wu. 2014. Evaluation of the accuracy of smartphone medical calculation apps. *J. Med. Internet Res.* 16, 2 (Feb. 2014), e32. https://doi.org/10.1007/s40264-013-0015-0

[16] Michael J. Breslow and Omar Badawi. 2012. Severity scoring in the critically ill. *Chest* 141, 1 (2012), 245–252. https://doi.org/10.1378/chest.11-0330

[17] Amy Brown, Apoorva Ballal, and Mo Al-Haddad. 2018. Recognition of the critically ill patient and escalation of therapy. *Anaesthesia & Intensive Care Medicine* 20 (12 2018). https://doi.org/10.1016/j.mpaic.2018.11.011

[18] Tevfik Bultan and Constance Heitmeyer. 2006. Analyzing tabular requirements specifications using infinite state model checking. *Formal Methods and Models for Co-Design, ACM/IEEE International Conference on* 0 (07 2006), 7–16. https://doi.org/10.1109/MEMCOD.2006.1695895

[19] Shauvik Roy Choudhary, Alessandra Gorla, and Alessandro Orso. 2015. Automated Test Input Generation for Android: Are We There Yet? (E). In *Proceedings of the 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE) (ASE '15)*. IEEE Computer Society, Washington, DC, USA, 429–440. https://doi.org/10.1109/ASE.2015.89

[20] Sauvik Das, Jason Wiese, and Jason I. Hong. 2016. Epistenet: Facilitating Programmatic Access Processing of Semantically Related Mobile Personal Data. In *Proceedings of the 18th International Conference on Human-Computer Interaction with Mobile Devices and Services* (Florence, Italy) *(MobileHCI '16)*. Association for Computing Machinery, New York, NY, USA, 244–253. https://doi.org/10.1145/2935334.2935349

[21] Leonardo De Moura and Nikolaj Bjørner. 2008. Z3: An Efficient SMT Solver. In *Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems* (Budapest, Hungary) *(TACAS'08/ETAPS'08)*. Springer-Verlag, Berlin, Heidelberg, 337–340.

[22] Michael Anthony Fajardo, Guy Balthazaar, Alexandra Zalums, Lyndal Trevena, and Carissa Bonner. 2019. Favourable understandability, but poor actionability: An evaluation of online type 2 diabetes risk calculators. *Patient Education and Counseling* 102, 3 (2019), 467–473. https://doi.org/10.1016/j.pec.2018.10.014

[23] Flavio Lopes Ferreira. 2001. Serial evaluation of the SOFA score to predict outcome in critically ill patients. *JAMA* 286, 14 (2001), 1754. https://doi.org/10.1001/jama.286.14.1754

[24] Greir Ander Huck Flynn, Barbara Polivka, and Jodi Herron Behr. 2018. Smartphone use by nurses in acute care settings. *Comput. Inform. Nurs.* 36, 3 (March 2018), 120–126.

[25] FTC. 2019. MOBILE HEALTH APPS INTERACTIVE TOOL. https://www.ftc.gov/tips-advice/business-center/guidance/mobile-health-apps-interactive-tool

[26] J Gardner-Thorpe, N Love, J Wrightson, S Walsh, and N Keeling. 2006. The Value of Modified Early Warning Score (MEWS) in Surgical In-Patients: A Prospective Observational Study. *The Annals of The Royal College of Surgeons of England* 88, 6 (2006), 571–575. https://doi.org/10.1308/003588406x130615

[27] Tim A. Green, Stevan Whitt, Jeffery L. Belden, Sanda Erdelez, and Chi-Ren Shyu. 2019. Medical calculators: Prevalence, and barriers to use. *Computer Methods and Programs in Biomedicine* 179 (2019), 105002. https://doi.org/10.1016/j.cmpb.2019.105002

[28] Faye Haffey, Richard R. Brady, and Simon Maxwell. 2013. A comparison of the reliability of smartphone apps for opioid conversion. *Drug Safety* 36, 2 (2013), 111–117. https://doi.org/10.1007/s40264-013-0015-0

[29] Shuai Hao, Bin Liu, Suman Nath, William G.J. Halfond, and Ramesh Govindan. 2014. PUMA: Programmable UI-Automation for Large-Scale Dynamic Analysis of Mobile Apps. In *Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services* (Bretton Woods, New Hampshire, USA) *(MobiSys '14)*. Association for Computing Machinery, New York, NY, USA, 204–217. https://doi.org/10.1145/2594368.2594390

[30] Tessa M. Hers, Jan Van Schaik, Niels Keekstra, Hein Putter, Jaap F. Hamming, and Joost R. Van Der Vorst. 2021. Inaccurate risk assessment by the ACS NSQIP Risk Calculator in aortic surgery. *Journal of Clinical Medicine* 10, 22 (2021), 5426. https://doi.org/10.3390/jcm10225426

[31] HHS. 2019. Health Information Laws. https://www.hhs.gov/hipaa/for-professionals/security/index.htmlas of December 2019.

[32] HHS. 2019. Summary of Privacy Rule. https://www.hhs.gov/hipaa/for-professionals/privacy/laws-regulations/index.htmlas of December 2019.

[33] Eveline Hitti, Dima Hadid, Jad Melki, Rima Kaddoura, and Mohamad Alameddine. 2021. Mobile device use among emergency department healthcare professionals: Prevalence, utilization and attitudes. *Scientific Reports* 11, 1 (2021). https://doi.org/10.1038/s41598-021-81278-5

[34] Kit Huckvale, Samanta Adomaviciute, José Tomás Prieto, Melvin Khee-Shing Leow, and Josip Car. 2015. Smartphone apps for calculating insulin dose: a systematic assessment. *BMC Med.* 13, 1 (May 2015), 106.

[35] W A Knaus, E A Draper, D P Wagner, and J E Zimmerman. 1985. APACHE II: a severity of disease classification system. *Crit. Care Med.* 13, 10 (Oct. 1985), 818–829.

[36] Mark Lawford and P Froebel. 2001. Application of tabular methods to the specification and verification of a nuclear reactor shutdown system. *Formal Methods in System Design - FMSD* (09 2001).

[37] Kyungmin Lee, Jason Flinn, T.J. Giuli, Brian Noble, and Christopher Peplin. 2013. AMC: Verifying User Interface Properties for Vehicular Applications. In *Proceeding of the 11th Annual International Conference on Mobile Systems, Applications, and Services* (Taipei, Taiwan) *(MobiSys '13)*. Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/2462456.2464459

[38] Yuanchun Li, Ziyue Yang, Yao Guo, and Xiangqun Chen. 2017. DroidBot: A Lightweight UI-guided Test Input Generator for Android. In *Proceedings of the 39th International Conference on Software Engineering Companion* (Buenos Aires, Argentina) *(ICSE-C '17)*. IEEE Press, Piscataway, NJ, USA, 23–26. https://doi.org/10.1109/ICSE-C.2017.8

[39] Gregory Y.H. Lip, Lars Frison, Jonathan L. Halperin, and Deirdre A. Lane. 2011. Comparative Validation of a Novel Risk Score for Predicting Bleeding Risk in Anticoagulated Patients With Atrial Fibrillation: The HAS-BLED (Hypertension, Abnormal Renal/Liver Function, Stroke, Bleeding History or Predisposition, Labile INR, Elderly, Drugs/Alcohol Concomitantly) Score. *Journal of the American College of Cardiology* 57, 2 (2011), 173–180. https://doi.org/10.1016/j.jacc.2010.09.024

[40] Gregory Y. Lip and Deirdre A. Lane. 2015. Stroke prevention in atrial fibrillation. *JAMA* 313, 19 (2015), 1950. https://doi.org/10.1001/jama.2015.4369

[41] Thomas F. Liu, Mark Craft, Jason Situ, Ersin Yumer, Radomir Mech, and Ranjitha Kumar. 2018. Learning Design Semantics for Mobile Apps. In *Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology* (Berlin, Germany) *(UIST '18)*. Association for Computing Machinery, New York, NY, USA, 569–579. https://doi.org/10.1145/3242587.3242650

[42] Royal College of Physicians. 2021. National Early Warning Score (NEWS) 2: standardising the assessment of acute-illness severity in the NHS. Updated report of a working party 2017. (2021).

[43] T. Olsson, A. Terent, and L. Lind. 2004. Rapid emergency medicine score: A new prognostic tool for in-hospital mortality in nonsurgical emergency department patients. *Journal of Internal Medicine* 255, 5 (2004), 579–587. https://doi.org/10.1111/j.1365-2796.2004.01321.x

[44] Ortholive. 2019. 32 Statistics on mHealth. https://www.ortholive.com/blog/32-statistics-on-mhealth/.

[45] Xiang Pan, Yinzhi Cao, Xuechao Du, Boyuan He, Gan Fang, and Yan Chen. 2018. FlowCog: Context-Aware Semantics Extraction and Analysis of Information Flow Leaks in Android Apps. In *Proceedings of the 27th USENIX Conference on Security Symposium* (Baltimore, MD, USA) *(SEC'18)*. USENIX Association, USA, 1669–1685.

[46] Rikesh K Patel, Adele E Sayers, Nina L Patrick, Kaylie Hughes, Jonathan Armitage, and Iain Andrew Hunter. 2015. A UK perspective on smartphone use amongst doctors within the surgical profession. *Ann. Med. Surg. (Lond.)* 4, 2 (June 2015), 107–112.

[47] Ryan Pelletier, Jeff Nagge, and John-Michael Gamble. 2022. Variation in bleeding risk estimates among online calculators. *Canadian Family Physician* 68, 4 (2022). https://doi.org/10.46747/cfp.6804e127

[48] Lenin Ravindranath, Suman Nath, Jitendra Padhye, and Hari Balakrishnan. 2014. Automatic and Scalable Fault Detection for Mobile Applications. In *Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services* (Bretton Woods, New Hampshire, USA) *(MobiSys '14)*. Association for Computing Machinery, New York, NY, USA, 190–203. https://doi.org/10.1145/2594368.2594377

[49] K J Rhee, C J Fisher, Jr, and N H Willitis. 1987. The Rapid Acute Physiology Score. *Am. J. Emerg. Med.* 5, 4 (July 1987), 278–282.

[50] A. J. Six, B. E. Backus, and J. C. Kelder. 2008. Chest pain in the emergency room: Value of the heart score. *Netherlands Heart Journal* 16, 6 (2008), 191–196. https://doi.org/10.1007/bf03086144

[51] Sharon R. Smith, Jack D. Baty, and Dee Hodge. 2002. Validation of the pulmonary score: An asthma severity score for children. *Academic Emergency Medicine* 9, 2 (2002), 99–104. https://doi.org/10.1197/aemj.9.2.99

[52] G Teasdale and B Jennett. 1974. Assessment of coma and impaired consciousness. A practical scale. *Lancet* 2, 7872 (July 1974), 81–84.

[53] Jean-Louis Vincent, Rui Moreno, Jukka Takala, S Willatts, A Mendonça, H Bruining, C Reinhart, Peter Suter, and L Thijs. 1996. The SOFA (Sepsis-related Organ Failure Assessment) score to describe organ dysfunction/failure. On behalf of the Working Group on Sepsis-Related Problems of the European Society of Intensive Care Medicine. *Intensive care medicine* 22 (08 1996), 707–10. https://doi.org/10.1007/BF01709751

[54] Sean Wallace, Marcia Clark, and Jonathan White. 2012. 'It's on my iPhone': attitudes to the use of mobile computing devices in medical education, a mixed-methods study. *BMJ Open* 2, 4 (Aug. 2012), e001099.

[55] Heinz Zimmermann and Jürg Reichen. 1998. Hepatectomy: Preoperative analysis of hepatic function and postoperative liver failure. *Digestive Surgery* 15, 1 (1998), 1–11. https://doi.org/10.1159/000018578