Kauno technologijos universitetas

Informatikos fakultetas

Mobiliųjų programėlių lokalizacijos defektų aptikimo sistema

Baigiamasis magistro studijų projektas

Projektą parengė

Gytis Šneideraitis

Projektui vadovavo

doc. Šarūnas Packevičius

Kaunas, 2026

Kauno technologijos universitetas

Informatikos fakultetas

Mobiliųjų programėlių lokalizacijos defektų aptikimo sistema

Baigiamasis magistro studijų projektas

Programų sistemų inžinerija (6211BX011)

Projektą parengė

Gytis Šneideraitis

Projektui vadovavo

doc. Šarūnas Packevičius

Projektą recenzavo

Doc. dr. Dominykas Barisas

Kaunas, 2026

Kauno technologijos universitetas

Informatikos fakultetas

Gytis Šneideraitis

Mobiliųjų programėlių lokalizacijos defektų aptikimo sistema

Akademinio sąžiningumo deklaracija

Patvirtinu, kad:

1. baigiamąjį projektą parengiau savarankiškai ir sąžiningai, nepažeisdamas (-a) kitų asmenų autoriaus ar kitų teisių, laikydamasis (-i) Lietuvos Respublikos autorių teisių ir gretutinių teisių įstatymo, Kauno technologijos universiteto (toliau – Universitetas) intelektinės nuosavybės valdymo ir perdavimo nuostatų bei Universiteto akademinės etikos kodekse nustatytų etikos reikalavimų;

2. visi baigiamajame projekte pateikti duomenys ir tyrimų rezultatai yra teisingi ir gauti teisėtai, nei viena projekto dalis nėra plagijuota nuo spausdintinių ar elektroninių šaltinių, o visos baigiamojo projekto tekste pateiktos citatos ir nuorodos yra nurodytos literatūros sąraše;

3. baigiamajame projekte tinkamai laikiausi asmens duomenų apsaugos reikalavimų, nenaudojau neskelbtinų ar konfidencialių duomenų be teisėto pagrindo, o jei juos naudojau, jie yra tinkamai nuasmeninti;

4. jei rengiant baigiamąjį projektą naudojausi dirbtinio intelekto (toliau – DI) ar kitais automatizuotais įrankiais, juos taikiau pagal Universitete nustatytą tvarką, nepažeisdamas (-a) akademinio sąžiningumo principų;

5. nesumokėjau ir nesu įsipareigojęs (-usi) mokėti jokių įstatymų nenumatytų piniginių sumų už baigiamąjį projektą ar jo dalis jokiam fiziniam ar juridiniam asmeniui;

6. suprantu, kad išaiškėjus akademinio nesąžiningumo ar kitų asmenų teisių pažeidimo faktui, man bus taikoma atsakomybė pagal Universitete nustatytą tvarką ir galiu būti pašalintas (-a) iš Universiteto; akademinio nesąžiningumo atvejis gali būti nagrinėjamas ir po studijų baigimo, inicijuojant kvalifikacinio laipsnio atšaukimo procedūrą.

Gytis Šneideraitis. Mobiliųjų programėlių lokalizacijos defektų aptikimo sistema. Magistro studijų baigiamasis projektas, vadovas doc. Šarūnas Packevičius; Kauno technologijos universitetas, Informatikos fakultetas.

Studijų kryptis ir studijų krypčių grupė: Programų sistemų inžinerija.

Reikšminiai žodžiai: lokalizacijos defektai, kompiuterinė rega, OCR, objektų aptikimas, dirbtinis intelektas, gilusis mokymasis, naudotojo sąsaja, vaizdo apdorojimas, automatinis testavimas, neuroniniai tinklai.

Kaunas, 2026. 72 p.

Santrauka

Mobiliųjų programėlių rinka yra milžiniška ir dažniausiai palaiko įvairias kalbas. Tačiau kiekvienos naujos kalbos palaikymas ar įdiegimas reikalauja daug tiek finansinių, tiek žmogiškųjų išteklių. Kiekviena kalba privalo būti tinkamai ištestuota, kad programėlė atitiktų keliamus reikalavimus bei užtikrintų naudotojų pasitenkinimą. Todėl testavimas yra itin svarbus siekiant užtikrinti, kad viskas būtų kokybiškai išversta į įvairias kalbas ir nebūtų padaryta klaidų. Vis dėlto šis procesas dažniausiai atliekamas rankiniu būdu, kuris užima daug laiko, o žmogiškasis faktorius dažnai lemia nepastebėtas klaidas.

Šiame darbe daugiausia dėmesio skiriama trijų technologijų ir jų galimybių analizei, siekiant testuoti įvairius programėlių lokalizacijos defektus bei pasiūlyti galimus jų pataisymus. Darbe analizuojamas optinis ženklų atpažinimas, objektų aptikimo modeliai bei dirbtinio intelekto daugiafunkciai modeliai. Prieš eksperimentinę dalį bus atlikta apklausa, skirta išsiaiškinti dabartinę situaciją ir problemas, su kuriomis susiduria įvairios sistemų kūrėjų grupės. Galiausiai bus atliktas visų minėtų technologijų eksperimentinis tyrimas, įvertinti jų gebėjimai, trūkumai bei galimi patobulinimai, kurie leistų pasiekti geresnių rezultatų. Remiantis gautais rezultatais bus galima nuspręsti, ar lokalizacijos defektų aptikimas yra realiai pasiekiamas rezultatas.

Gytis Šneideraitis. Mobile Application Localization Defect Detection System. Master's Final Degree Project, supervisor prof. Šarūnas Packevičius Faculty of Informatics, Kaunas University of Technology.

Study field and area (study field group): Software Engineering.

Keywords: localization defects, computer vision, OCR, object detection, artificial intelligence, deep learning, user interface, image processing, automated testing, neural networks.

Kaunas, 2026. 72 p.

Summary

The mobile application market is enormous and most applications support multiple languages. However, supporting or implementing each new language requires significant financial and human resources. Every language must be thoroughly tested to ensure that the application meets the required standards and guarantees user satisfaction. Therefore, testing is extremely important in ensuring that everything is translated correctly into different languages and that no errors are made. Nevertheless, this process is usually performed manually, which is time-consuming, and human error often results in unnoticed defects.

This thesis primarily focuses on the analysis of three technologies and their capabilities in testing various mobile application localization defects and proposing possible fixes for them. The work examines optical character recognition, object detection models, and multimodal artificial intelligence models. Before the experimental part, a survey will be conducted to identify the current situation and the problems faced by different software development teams. Finally, an experimental study of all the mentioned technologies will be carried out, evaluating their capabilities, limitations, and possible improvements that could lead to better results. Based on the obtained results, it will be possible to determine whether localization defect detection is a realistically achievable outcome

Turinys

Lentelių sąrašas

Paveikslų sąrašas

Įvadas

Šiuo metu mobiliųjų programėlių rinka yra milžiniška, o dauguma populiacijos naudojasi išmaniaisiais įrenginiais. Kiekvienas iš šių naudotojų naudojasi įvairiomis programėlėmis – tiek pramogoms, tiek darbui. Tai rodo, kad įvairių programėlių paklausa yra didžiulė, ką pastebi tiek finansuotojai, tiek sistemų kūrėjai. Sistemų kūrimas yra sudėtingas, ilgas ir brangus procesas, reikalaujantis daug žmogiškųjų ir kitų ribotų išteklių.

Todėl, siekiant, kad sukurtos programėlės būtų kokybiškos ir atitiktų naudotojų lūkesčius, sistemų kūrėjai privalo jas kuo išsamiau testuoti. Kiekvieno naujo funkcionalumo įvedimas į sistemą kelia riziką, kad sistema veiks ne taip, kaip numatyta, o tai gali lemti naudotojų nepasitenkinimą. Yra daug testavimo būdų: scenarijų rašymas, statinė kodo analizė ir tiesioginis sistemos testavimas vertinant jos veikimą bei nustatant galimas klaidas. Tačiau toks testavimas reikalauja daug laiko, žmogiškųjų išteklių ir atidumo, siekiant išvengti didžiosios dalies klaidų. Vis dėlto dažnai nepavyksta jų visų pašalinti, todėl sistemoje lieka tam tikrų defektų. Dėl šios priežasties būtina ieškoti alternatyvų – naujų testavimo metodų ar įrankių, galinčių tai užtikrinti.

Sistemos dažnai pateikiamos įvairiomis kalbomis, o kiekvienos kalbos testavimas yra sudėtingas ir nepatogus procesas. Toks testavimas dažniausiai reikalauja peržiūrėti visą sistemą ir vertinti kiekvieną jos dalį. Lokalizacija gali turėti įtakos grafinės sąsajos elementų išdėstymui, gali atsirasti persidengimų ir kitų nenumatytų atvejų, nes sistema dažniausiai kuriama vienai kalbai, tačiau tikimasi, kad veiks visomis. Todėl, siekiant sumažinti poreikį kiekvieną kalbą testuoti rankiniu būdu, kad nereiktų eiti per visus sistemos langus, būtina sukurti automatizuotą sistemą, gebančią aptikti defektus, juos identifikuoti ir pasiūlyti taisymo būdus.

Šio darbo tikslas – naudojant programėlės ekranus automatiškai identifikuoti lokalizacijos defektus, pateikti juos sistemos testuotojui su defektų sąrašu ir siūlomais taisymo būdais, kad sistemų kūrėjai galėtų juos ištaisyti savarankiškai, neieškodami jų rankiniu būdu.
Šiam tikslui pasiekti keliami šie uždaviniai:

- Atlikti išsamią esamų sprendimų ir galimų šios problemos sprendimo būdų analizę, juos identifikuoti ir įvertinti.
- Remiantis atlikta analize, suformuluoti reikalavimus naujam sprendimui ir pasirinkti tinkamus metodus.
- Sukurti sistemą, gebančią automatizuotai aptikti lokalizacijos defektus.
- Atlikti tyrimą, vertinant sukurtos sistemos galimybes padėti testuotojams, bei nustatyti, ar toks sprendimas yra patrauklus ir efektyvus užtikrinant programėlių kokybę lokalizacijos aspektu.

1. Analitinė dalis

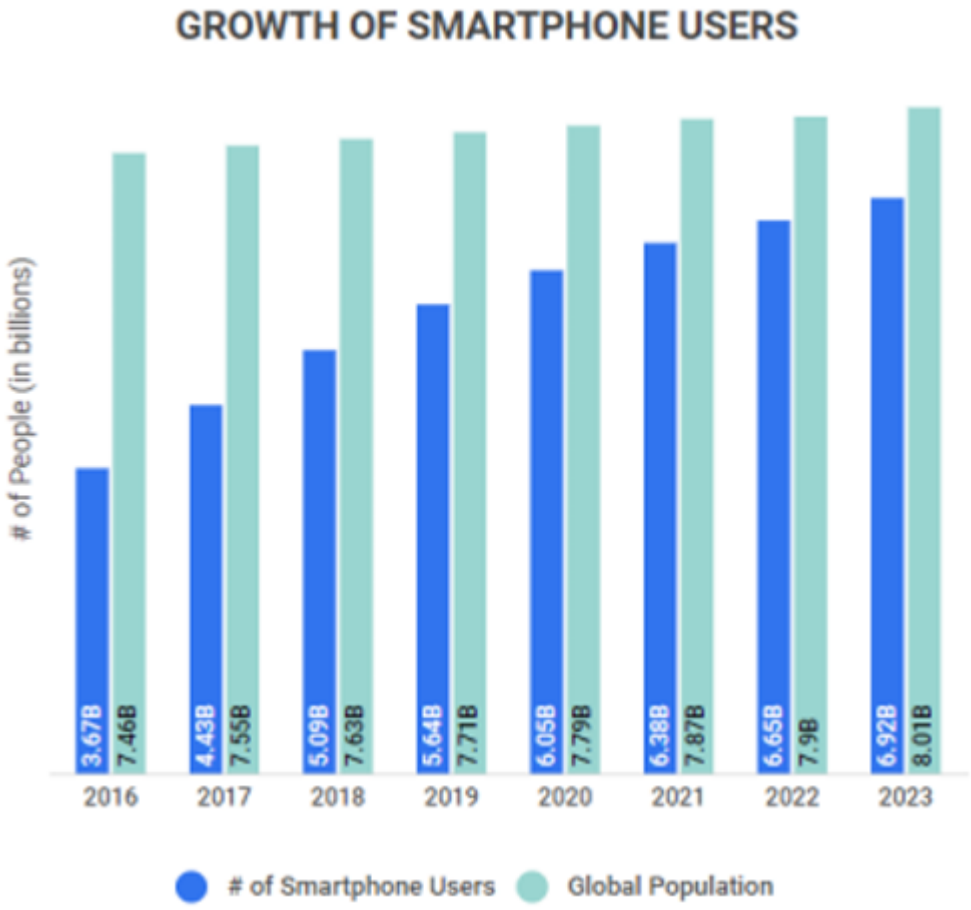
1.1. Įvadas

Šiais laikais sukuriama daug programėlių, skirtų pramogoms, darbui, verslui ir kt. Kiekviena sukuriama sistema turi būti pilnai parengta, atitikti naudotojų lūkesčius, veikti be trukdžių ir, aišku, turėti kuo mažiau klaidų. Programėlių testavimui dažnai yra įsteigiamos atskiros komandos įmonėse, kurios atlieka kartais nuobodų, bet reikšmingą darbą – testuoja programėles rankiniu būdu arba rašydamos testus. Šis darbas reikalauja didelio sutelkto dėmesio, kad nebūtų praleista nei menkiausia klaida. Tačiau klaidų išvengti visiškai dažnai neįmanoma, nes sunku pilnai ištestuoti viską. Viena iš dažniausiai pasitaikančių klaidų testuojant rankiniu būdu yra lokalizacijos defektai. Jie susiję su teksto įvedimu, jo kokybe ir tinkamu vertimu. Būtent šiam darbui, reikalaujančiam rankinio testavimo, buvo atlikta analizė ir vėliau sukurtas įrankis, leidžiantis automatizuotai ištestuoti ir rasti lokalizacijos defektus bei pateikti tikslią ataskaitą testuotojams. Tokių įrankių naudojimas leidžia greičiau ištestuoti programėles, mažinant laiko sąnaudas ir projekto išlaidas. Be to, reikia paminėti, kad šiuolaikiniame pasaulyje žmonės naudoja įvairiausius įrenginius. Programėlių ištestavimas visuose įrenginiuose yra sudėtingas, todėl sukurtas įrankis spręs šią problemą, leisdamas testuoti programėles daugybėje įrenginių vienu metu.

Galiausiai, dėl šio tyrimo bei sukurtos sistemos, programų sistemų vystymo komandos galės skirti daugiau laiko funkcionalumo plėtrai, o mažiau – testavimui, kuris yra laiko reikalaujantis procesas. Tyrimas padės geriau suprasti, kaip kurti tokius sprendimus, kurie leistų efektyviai ir automatizuotai nustatyti sistemos trūkumus, juos identifikuoti, ir perduoti šią informaciją sistemų analitikams. Tai leis analitikams greičiau ir efektyviau šalinti klaidas, tobulinti sistemos veikimą ir užtikrinti aukštesnį kokybės lygį. Todėl šis projektas ne tik suteiks komandoms galimybę efektyviau paskirstyti resursus tarp kūrimo ir testavimo procesų, bet ir padidins bendrą darbo našumą bei kokybę. Tai mažins klaidų pasikartojimą ir pagerins naudotojų patirtį.

1.2. Galimi lokalizacijos defektai

Pirmiausia svarbu aptarti galimus lokalizacijos defektus, su kuriais sistemų kūrėjai susiduria. Pastebima, kad mobiliosios programėlės yra labiausiai imlios naudotojo sąsajos testavimui, ypač laiko atžvilgiu. Taip pat svarbu paminėti, kad mobiliųjų programėlių rinka yra labai didelė, o mobiliuosius telefonus turi beveik kiekvienas – apie 80 proc. žmonių (žr. 1 pav.) [1].



1 pav. Mobiliųjų telefonų naudotojų skaičius

Iš visų žmonių, turinčių šiuos įrenginius, apie 90 proc. praleidžia laiką įvairiose programėlėse [1]. Tai parodo, kad mobiliųjų programėlių rinka yra milžiniška. Todėl daugelis siekia patekti į šią rinką, sukurti naują programėlę ir iš jos išgauti pelną. Tačiau programėlių kūrimas yra sudėtingas ir ilgas procesas, o viena iš jo dalių – programėlių testavimas. Lokalizacijos defektai yra vieni iš dažniausiai pasitaikančių programėlių defektų, kurių ištaisymas ypač svarbus, kai yra kuriamos programėlės, skirtos plačiajai rinkai ir verčiamos į kelias kalbas. Šie defektai yra skirstomi į keletą pogrupių, kuriuose yra tam tikri testavimo tipai ir jų metu identifikuojami defektai [2]:

1. Funkcinis ir naudojimo sąsajos lokalizavimo testavimas
    a. Naudotojo sąsajos lokalizacija, įtraukiant problemas, tokias kaip elementų persidengimas arba teksto apkarpymas.
    b. Programėlės funkcionalumo testavimas, pvz., vertimų atitikimas naudotojo sąsajos funkcijoms.
    c. „Įprasto“ dydžio mygtukų patikrinimas.
    d. Neišverstas tekstas
    e. Netinkamai įtraukti tekstai.
2. Kalbos testavimas ir atvaizdavimas:
    a. Gramatikos, rašybos ir skyrybos klaidos.
    b. Valiutos, simbolių, piktogramų, kainodaros ir mokėjimų atvaizdavimas.
    c. Iš dešinės į kairę rašomų kalbų (RTL) vertinimas.

    d. Kliento specifinės programėlės terminologijos testavimas.
3. Lingvistinis ir vertimo testavimas
    a. Nenatūralūs arba netikslūs vertimai.
    b. Kalbos nuoseklumo visoje programėlėje užtikrinimas.

Kiekvieną iš šių testavimų turėtų atlikti testuotojų grupė, kuri dažnai jau turi kitų testavimo užduočių, sunaudojančių didelę laiko dalį. Kiekvienas iš šių paminėtų defektų gali būti pilnai ištestuotas tiek rankiniu, tiek automatiniais būdais. Būtent tai ir bus analizuojama šio tyrimo metu ir kaip šiuos defektus būtų galima aptikti.

1.3. Lokalizacijos defektų įtaka sistemų populiarumui ir ekonomiškumui

Programinės įrangos lokalizacijos defektai turi tiesioginę įtaką sistemos kokybei ir vartotojų pasitenkinimui, o tai galiausiai veikia programėlės populiarumą bei ekonominius rezultatus. Bet koks sistemos defektas, įskaitant ir lokalizacijos klaidas, gali sukelti neigiamą naudotojų patirtį bei paveikti bendrą produkto patikimumą.

Pirmiausia aptarsime, kokį poveikį lokalizacijos defektai turi ar gali turėti naudotojų pasitenkinimui. Kiekvienas vartotojas yra svarbus, todėl svarbi ir jo nuomonė. Tokios problemos kaip netikslūs, sunkiai suprantami ar gramatinių klaidų turintys tekstai neigiamai veikia vartotojų požiūrį į sistemą. Yra atlikta daug tyrimų šia tema. Kai kurie iš jų teigia, kad teikiamų paslaugų kokybė – į kurią įeina ir lokalizacijos aspektai – turi reikšmingą tiesioginį poveikį naudotojų lojalumui ir pasitenkinimui [3].

Jeigu susitelksime ties mobiliųjų programėlių kūrimu ir jų testavimu, būtina užtikrinti jų kokybę, atsižvelgiant į didelę mobiliųjų programėlių rinkos konkurenciją. Jei nebus užtikrintas aukštas kokybės lygis, programėlė gali patirti nesėkmę, o ją kūrusiai įmonei gali tekti grįžti prie sistemos perdarymo. Dėl šios priežasties automatizuoti testai leidžia daugeliui įmonių efektyviai ištestuoti savo programėles. Automatizuoti testai gali aptikti ir ištaisyti apie 91 % defektų, kas yra itin svarbu siekiant išlaikyti tiek esamų, tiek būsimų klientų pasitikėjimą. Lokalizacijos defektai šiuo atžvilgiu yra ypač svarbūs, nes per tekstą vyksta tiesioginė komunikacija su naudotoju – jei tekstas klaidingas, nepatogus ar sunkiai suprantamas, atitinkamai didėja klientų nepasitenkinimas [4].

Žvelgiant iš ekonominės pusės, programėlių testavimas – įskaitant ir lokalizacijos defektų paiešką – tiesiogiai veikia investicinę grąžą. Pastebima, kad gerai ištestuotų sistemų investicinė grąža yra ženkliai didesnė nei prastai ištestuotų. Tai patvirtina ir įvairūs šaltiniai, teigiantys, kad programėlių testavimas, atliekamas naudojant įvairius įrankius ir dirbtinio intelekto modelius, leidžia sumažinti duomenų nutekėjimo riziką, padidina programėlių greitaveiką bei patogumą, taip pat išplečia jų pritaikomumo galimybes. Visa tai daro tiesioginę įtaką programėlių finansinei sėkmei ir konkurencingumui sparčiai augančioje mobiliųjų programėlių rinkoje [5, 6].

Testavimo kokybė tiesiogiai lemia galutinio produkto sėkmę rinkoje. Atliekami įvairūs testavimai, nuo kurių priklauso sistemos veikimas – įskaitant ir lokalizacijos defektų identifikavimą bei šalinimą. Šis procesas yra vienas svarbiausių siekiant užtikrinti, kad sukurtas produktas būtų prieinamas įvairių šalių naudotojams ir kad jų patirtis naudojantis

produktu būtų teigiama. Visa tai padeda užtikrinti programėlės finansinę sėkmę ir jos konkurencingumą.

1.4. Defektų aptikimas naudojant kompiuterinę regą

Vienas iš būdų aptikti defektus ekrano nuotraukose yra naudoti kompiuterinės regos modelius. Toks modelis yra PID (angl. Perceptual Image Differencing) [7], kuris naudoja ekrano kopijas ir palygina jas su nurodytomis kitų puslapių ekrano nuotraukomis. Ši technika leidžia aptikti vizualinius skirtumus, pvz., VIs (vizualinės nesuderinamumo problemos), kurie gali turėti įtakos naudotojo sąsajos patogumui ir funkcionalumui.

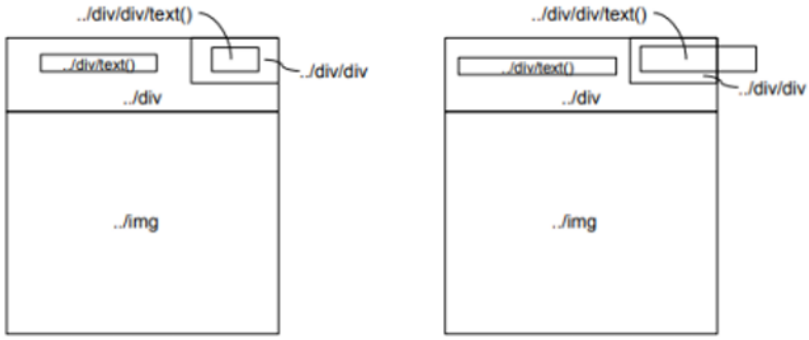
PID metodas yra ypač efektyvus, nes jis orientuojasi į žmogaus suvokiamus vaizdo skirtumus, nekreipdamas dėmesio į mažus, nežymius pikselių lygio skirtumus, kurie dažniausiai nėra svarbūs galutiniam naudotojui. Taikant šį metodą, pirmiausia reikia užfiksuoti naršyklėje atvaizduoto pradinio - gero puslapio ir bandymo tinklalapio ekranų nuotraukas. Vėliau šios nuotraukos apkerpamos iki iš anksto pažymėtų sričių, ir naudojant WebSee sistemos PID funkciją, analizuojamos apkarpytos nuotraukos, siekiant nustatyti vizualinius skirtumus pikselių lygmenyje (žr. 2 pav.). Tai užtikrina tikslesnį ir žmogui pritaikytą defektų aptikimą [7].

2 pav. Paveikslėlių lyginimai ir defektų aptikimas

Nors PID metodas yra naudingas, jis turi keletą trūkumų. Kartais, lyginant nuotraukas, gali būti aptinkami tam tikri skirtumai ar defektai, tačiau tai nebūtinai reiškia, kad tai yra realūs ir pagrįsti defektai. Pavyzdžiui, puslapių išdėstymas gali skirtis, bet pateikta lyginamoji nuotrauka vis dėlto gali būti teisinga [8]. Nepaisant to, šis metodas gali padėti išryškinti tam tikras vietas, kuriose gali būti defektų. Tarkime, turint nuotrauką su tinkamu išdėstymu ir numatyta pagrindine kalba, lyginant ją su identiškomis nuotraukomis, bet kitomis kalbomis, gali padėti atskleisti tam tikrus lokalizacijos trūkumus. Skirtingos kalbos gali keisti puslapio struktūrą, o tai kartais net gali sugadinti puslapio išdėstymą, ypač jei tekstas yra ilgas ir keičiant kalbas, jo ilgis gali didėti arba mažėti.

1.5. Teksto dydžio kitimo dėl lokalizacijos problemų sprendimas

Kaip jau buvo minėta, keičiant kalbas sistemoje, dažnai keičiasi ir teksto dydis, kuris turėtų dinamiškai prisitaikyti prie ekrano dydžio. Tačiau visų galimų variantų įvertinimas yra sudėtingas, todėl neretai atsitinka, kad tekstas išlenda iš savo rėmų į kitą komponentą, taip jį uždengiant (žr. 3 pav.), arba tekstas išlenda už ekrano ribų, dėl ko negalima matyti viso teksto [8].



3 pav. Teksto netilpimas į objekto rėmus

Norint spręsti interneto puslapiuose pasitaikančius tekstų išlindimo defektų nustatymo iššūkius, siūlomas metodas apima išsamų procesą, kuris yra aprašytas toliau [8]:

- Maketo grafiko (MG) kūrimas: Pirmasis žingsnis yra modelio, vadinamo maketo grafiku (MG), apibrėžimas. Šis modelis yra svarbus, nes jis fiksuoja vizualinius santykius bei santykinę HTML žymių ir teksto elementų padėtį internetiniame puslapyje ar programėlėje. MG tarnauja kaip įrankis išsamiai suprasti interneto puslapio struktūrinį išdėstymą ir yra būtinas norint nustatyti išdėstymo problemas, kurios gali atsirasti vertimo proceso metu [8].

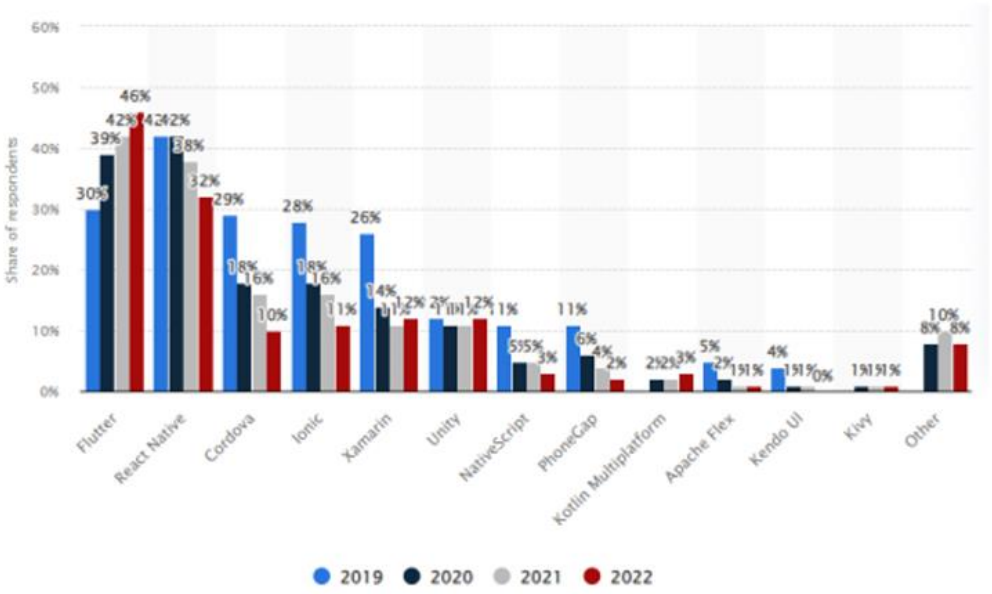
- Įvesties interneto puslapių teikimas: Kad metodas veiktų efektyviai, reikalingos dvi interneto puslapio ar programėlės versijos. Pirmoji yra „Testuojamas puslapis“ (angl. Page Under Test - PUT), tai yra puslapio ar programėlės versija, kuri buvo išversta ir kuri yra tikrinama dėl lokalizacijos defektų. Antrasis elementas yra bazinė versija, idealiu atveju tai yra originali puslapio ar programėlės versija prieš vertimą. Ši bazinė versija yra žinoma kaip turinti teisingą išdėstymą ir tarnauja kaip atskaitos taškas nustatant nuokrypius testuojamame puslapyje [8].

- Kiekvieno puslapio maketo grafikų kūrimas: Nustačius įvesties puslapius, kitas žingsnis yra sukurti maketo grafiką (MG) kiekvienam puslapiui. Tai apima MG kūrimą tiek bazinei versijai, tiek PUT (Testuojamam puslapiui). Šie grafikai atspindi atitinkamų puslapių išdėstymą ir yra esminiai nustatant bet kokius išdėstymo skirtumus, kurie gali kilti dėl vertimo proceso [8].

- Maketo grafikų palyginimas: Sukūrus maketo grafikus (MG), metodas reikalauja jų išsamaus palyginimo. Šio palyginimo tikslas – rasti skirtumus, kurie gali signalizuoti apie galimas klaidas. Analizuojant, kaip elementai Testuojamame puslapyje (PUT) nukrypsta nuo bazinės versijos, tampa įmanoma identifikuoti konkrečias sritis, kuriose išdėstymas galėjo būti neigiamai paveiktas dėl vertimo [8].

- Galimai klaidingų elementų analizavimas ir reitingavimas: Galutinis žingsnis yra išsami anksčiau aptiktų skirtumų analizė. Šiame etape metodas filtruoja šiuos elementus ir kuria jų reitinguotą sąrašą. Šis sąrašas yra ypač naudingas kūrėjams, nes jis pabrėžia tuos elementus, kuriuose labiausiai tikėtina aptikti defektus. Taip yra suteikiamas aiškus prioritetas gedimų taisymui ir tolesnei korekcijai [8].

Šiais žingsniais metodas sistemiškai nustato ir sprendžia susijusius su lokalizacijos defektais, užtikrindamas, kad išversti interneto puslapiai ar programėlės išlaikytų jų numatytą išdėstymą ir palaikytų gerą naudotojo patirtį. Nors šie žingsniai yra aprašyti daugiau kaip žiniatinklių aplikacijų testavimo kontekste, jie yra lengvai pritaikomi ir mobiliųjų programėlių atveju. Daugelis mobiliųjų programėlių yra kurtos naudojant programavimo karkasus (žr. 4 pav.), kurie yra labai panašūs į žiniatinklių aplikacijų karkasus ir programavimo kalbas. Dėl šių panašumų, metodai ir rekomendacijos, taikomi žiniatinklio aplikacijoms, yra taip pat pritaikomi mobiliųjų programėlių srityje [9].



4 pav. Mobilių programėlių karkasų naudojimas

1.6. Dvikryptės kalbos

Kalbant apie lokalizacijas mobiliuosiuose programėlėse, svarbu paminėti vieną išskirtinį atvejį – dvikryptes kalbas. Tai dažniausiai yra arabų ar hebrajų kalbos, kurių rašymo ir skaitymo tvarka skiriasi nuo daugumos žinomų kalbų. Šių kalbų tekstai yra skaitomi iš dešinės į kairę (žr. 5 pav.), dėl ko ir teksto išdėstymas programėlėse yra pritaikytas dešinėje. Be to, šios kalbos pasižymi ganėtinai unikaliu raidynu, kuris dažnai neatitinka įprastų standartų [10].

5 pav. Elementų išdėstymas anglų ir arabų kalbomis

Toks veidrodinis atvaizdavimas yra taikomas įvairiems piktogramoms ir tekstui, esančiam šalia jų. Galima išskirti keletą UI elementų, kuriems būdingas toks veidrodinis atspindys dvikrypčių kalbų atveju [11]:

- Puslapiavimas – skaičiai ir tekstas eina iš dešinės į kairę, o tekstas turi būti tinkamai lygiuojamas prie atitinkamo UI elemento.
- Pagrindiniai mygtukai turėtų būti išdėstyti dešinėje pusėje, o tekstas lygiuojamas iš dešinės į kairę
- Pranešimai – gauti pranešimai turėtų būti atvaizduojami pagal kalbos skaitymo kryptį; pavyzdžiui, arabų kalbos atveju – iš dešinės į kairę.
- Meniu – navigacijos meniu, esantis įvairiose programėlėse ar tinklalapiuose, gali būti pritaikytas veidrodiniam atvaizdavimui, atsižvelgiant į kalbos ypatumus.
- Paieškos laukai turi būti pritaikyti pagal naudojamos kalbos skaitymo kryptį, užtikrinant intuityvų naudojimąsi paieškos funkcija.
- Formų laukai – pvz., registracijos ar prisijungimo formose, tekstas ir įvedimo laukai turėtų atitikti kalbos skaitymo kryptį, suteikiant vartotojui patogesnį naudojimąsi.

Šios UI adaptacijos yra ypač svarbios kurdami daugiakalbes programas ar svetaines, siekiant užtikrinti, kad naudotojų patirtis būtų intuityvi ir patogi, nepriklausomai nuo jų kalbos ypatumų. Visas šis programų kūrimo procesas reikalauja kruopštaus testavimo, atsižvelgiant į visas palaikomas kalbas. Todėl svarbu prieš pradėdami testavimą žinoti, kurias kalbas sistema pilnai palaiko ar planuoja palaikyti, ir pagal tai atitinkamai atlikti testavimą [12].

1.7. OCR technologija ir jos taikymas

Viena iš populiariausių technologijų, naudojamų tekstų išgavimui iš paveikslėlių, yra OCR (angl. Optical Character Recognition). Ši technologija leidžia nuskaityti ir skaitmenizuoti

paveikslėlyje esantį tekstą. Jos naudojimas yra paplitęs tarp didžiųjų kompanijų, tokių kaip Apple, Google, ir Amazon. 2023 m. OCR technologija padarė reikšmingą pažangą, pasiūlydama aukštesnį tikslumo lygį, ypač gerai suformatuotuose dokumentuose. Pavyzdžiui, „Google Docs OCR" ir „ABBYY 13 FineReader" teigia, kad jų charakterių atpažinimo tikslumas siekia 98-99%. Panašaus lygio tikslumą taip pat demonstruoja „Microsoft Azure Computer Vision" ir „Amazon Textract" [13].

Nepaisant didelės pažangos, OCR technologija vis dar susiduria su iššūkiais, ypač neidealiose sąlygose. Tokios problemos kaip prasta vaizdo kokybė, sudėtingi ar painūs šriftai, įvairūs dokumentų išdėstymai ir senų rankraščių atpažinimas yra tik keli iš jų. Be to, OCR sistemoms dažnai sunku tvarkyti ne lotyniškas raštų sistemas, pavyzdžiui, kinų, arabų ar sanskrito kalbas [13].

OCR technologija paverčia tekstinį vaizdą į mašinomis skaitytiną formatą. Ši technologija leidžia dokumentų vaizdų konvertavimą į teksto duomenis, palengvina jų analizę ir integraciją į skaitmeninius darbo procesus, taip padidindama operacinį efektyvumą ir produktyvumą [14].

OCR proceso etapai [14]:

- Vaizdo gavimas: dokumentų skenavimas ir jų konvertavimas į dvejetainius duomenis.
- Apdorojimas prieš analizę: vaizdo valymas, klaidų šalinimas ir paruošimas teksto skaitymui.
- Teksto atpažinimas: naudojant algoritmus tekstui atpažinti remiantis šablonų atitikimu ir savybių išskyrimu.
- Apdorojimas po analizės: konvertuojamas išgautas tekstas į kompiuterio failą, kartais sukuriant anotuotus PDF failus.

Skirtingi OCR technologijų tipai naudojami atsižvelgiant į jų panaudojimą ir poreikius. Tai apima paprastą OCR, intelektualų simbolių atpažinimą (ICR), intelektualų žodžių atpažinimą ir optinį ženklų atpažinimą. Kiekvienas šių tipų skiriasi pagal naudojamus metodus ir sudėtingumą, kai kurie iš jų taip pat naudoja pažangius mašinų mokymosi metodus.

OCR siūlo keletą svarbių privalumų:

- Paieškomas tekstas: konvertuoja dokumentus į lengvai prieinamą ir analizuojamą formatą.
- Operacinis efektyvumas: automatizuoja dokumentų tvarkymą, taupo laiką ir išteklius.
- Dirbtinio intelekto integracija: leidžia pagerinti AI sprendimus, atpažįstant tekstą įvairiose situacijose, pavyzdžiui, autonominiuose automobiliuose ar socialinėje žiniasklaidoje.

OCR yra svarbus įrankis įvairiose pramonės šakose, tokiose kaip bankininkystė, sveikatos priežiūra ir logistika. Jis naudojamas finansinių dokumentų patikrinimui, pacientų įrašų tvarkymui ir logistikos dokumentų valdymui, gerokai pagerinant efektyvumą ir saugumą [14].

Pasitelkus OCR atliktą teksto išgavimą, galima taikyti kitus testavimo būdus. Vienas iš tokių – žodžio rašybos tikslumo patikrinimas.

1.8. Žodžių rašybos tikrinimas

Kaip jau buvo minėta, turint išgautą tekstą, atsiranda galimybė šį tekstą apdoroti. Viena iš dažniausiai pasitaikančių lokalizacijos defektų – klaidos įvairiuose žodžiuose. Šią problemą galima spręsti naudojant apmokytus kalbų tikrinimo modelius, kurie sugeba aptikti rašybos klaidas žodžiuose.

Šių klaidų aptikimui galima kurti savo personalizuotus modelius, kurie leistų aptikti specifines rašybos klaidas, arba naudoti jau sukurtus modelius, kurie supaprastina šį darbą. Vienas iš tokių modelių yra NeuSpell, kuris yra skirtas angliškų žodžių rašybos klaidų aptikimui.

NeuSpell yra neuroninio rašybos taisymo įrankių rinkinys, sukurtas anglų kalbai. Jis apima įvairius modelius ir juos vertina realioje aplinkoje, naudodamas klaidingus žodžius iš įvairių šaltinių. Šio įrankio pagrindinis tikslas – spręsti kitų modelių trūkumus, tokius kaip neefektyvus konteksto, esančio aplink žodį, išnaudojimas, ir taip tobulinti žodžių rašybos tikslumą. Svarbus NeuSpell požiūrio aspektas yra jo priklausomybė nuo gilios konteksto aibės reprezentacijos, kuri suteikia šiam modeliui aukštą tikslumą. Ši strategija žymiai pagerina įrankių rinkinio gebėjimus atpažinti ir ištaisyti klaidingus žodžius, remiantis platesniu teksto kontekstu. Įrankių rinkinys yra išsamus ir apima dešimt skirtingų rašybos taisymo modelių. Šis spektras apima tiek paprastus, tiek neuroninius modelius, kai kurie iš pastarųjų yra papildyti giliais konteksto apmokytais duomenimis. Ši įvairovė užtikrina, kad NeuSpell galėtų patenkinti įvairius poreikius ir suteikti plačias taikymo galimybes [15].

Minėtas NeuSpell yra tik vienas iš daugelio galimų variantų, skirtų aptikti kalbų trūkumus. Tačiau svarbu atkreipti dėmesį į tai, kad kalbų testavimui reikėtų rinktis tas kalbas, kurias naudoja daugelis naudotojų. Ne visada yra praktiška testuoti visas kalbas, nes tai gali būti labai brangu. Be to, jeigu reikalingi specifiniai moduliai nėra sukurti, juos kurti patiems gali būti neįmanoma dėl didelių sąnaudų ir ilgo proceso.

Kaip minėta, NeuSpell yra dirbtinio intelekto modelis, specializuotas žodžių klaidų taisymui ir konteksto supratimui, tačiau jis palaiko tik ribotą kalbų rinkinį. Šiais laikais turime daugybę kitų įrankių, kurie tai atlieka dar geriau ir gali dirbti su kur kas įvairesnėmis kalbomis. Tokie įrankiai kaip „Grammarly“, „Google Docs“ ar „Microsoft Editor“ pasitelkia sudėtingus giliojo mokymosi algoritmus ir modelius. Jie yra panašūs į pokalbių modelius, tokius kaip „ChatGPT“, kurie yra apmokyti naudojant milžiniškus tekstų rinkinius iš interneto, knygų ir įvairaus tekstinio turinio. Šie modeliai geba atpažinti bei suprasti gramatikos klaidas, sakinio struktūros netikslumus ir žodžių derinimo klaidas [16].

Tyrimai rodo, kad tokie įrankiai, apmokyti didžiuliais tekstų kiekiais, sugeba aptikti įvairias klaidas su maždaug 90 % patikimumu, įskaitant sakinio struktūros ar logikos klaidas, o gramatikos klaidas, pavyzdžiui, „Google“ modeliai, gali aptikti net iki 98 % tikslumu. Kaip ir NeuSpell, šie modeliai turi gebėjimą išlaikyti tam tikrą kontekstą – tai yra, analizuoti jau parašytą tekstą ir jį panaudoti tikslinant rašybą, stilių ar kitus aspektus. Šie modeliai geba prisitaikyti prie rašymo stiliaus ir pagal tai koreguoti tekstą [16].

1.9. Mašininių algoritmų taikymas objektų aptikimui

Kiekvieną UI nuotrauką galima suskaidyti į mažesnius objektus, tuomet atpažinti šiuos objektus ir galiausiai ieškoti juose būdingų lokalizacijos defektų. Tai apima objektų klasifikavimą, jų išgavimą ir, taikant įvairius algoritmus ar net pačius mašininio mokymosi algoritmus, identifikuoti defektus šiuose objektuose.

Tradicinė objektų aptikimo modelių grandinė apima tris pagrindinius etapus: informatyvių regionų atranką, požymių išskyrimą ir objektų klasifikavimą [17]:

1. Informatyvių Regionų Atranka: Šiame etape identifikuojamos tos vaizdo dalys, kuriose galimai yra objektų. Tradicinės metodikos dažnai naudoja slankiojančius langus (angl. multi-scale sliding window) visam vaizdui nuskaityti. Tačiau analizuojamas metodas yra skaičiavimų požiūriu brangus ir neefektyvus dėl daugybės pasikartojančių kandidatų langų generavimo [17].

2. Požymių išskyrimas: Antrajame etape iš atrinktų regionų išskiriami vizualūs požymiai. Šie požymiai turėtų būti atsparūs ir semantiškai prasmingi, kad būtų galima tiksliai atpažinti įvairius objektus. Tradicinės metodikos naudojo rankiniu būdu sukurtus požymius, tokius kaip SIFT, HOG ir Haar tipo požymiai. Tačiau šie požymiai dažnai susiduria su sunkumais, susijusiais su objektų išvaizda, apšvietimo sąlygomis ir fono įvairove [17].

3. Klasifikavimas: Galutinis etapas apima išskirtų požymių klasifikavimą į skirtingas objektų kategorijas. Tai pasiekiama naudojant klasifikatorius, tokius kaip atraminių vektorių mašina (SVM), AdaBoost arba Deformuojamas Dalinis Modelis (DPM). Pavyzdžiui, DPM metodas derina objektų dalis su deformacijos sąnaudomis ir naudoja diskriminacinį mokymąsi, kad sukurtų aukštos tikslumo dalinius modelius [17].

Suskaidžius nuotrauką į atskirus objektus, atsiveria galimybė lengviau pritaikyti algoritmus, kurie gali būti labai specifiški kiekvienai objektų kategorijai. Toks personalizuotas defektų aptikimo metodas kiekvienam komponentui gali žymiai padidinti defektų aptikimo teisingumo koeficientą, o tai šiam tyrimui yra labai svarbus aspektas.

Svarbu paminėti, kad elementai, turintys teksto, yra sunkiau aptinkami ir klasifikuojami nei tie, kurie neturi teksto. Buvo atlikti tyrimai [18], kurie parodė, jog atskiriant tekstinius elementus nuo netekstinių, objektų aptikimo tikslumas žymiai padidėjo.

Dėl šios priežasties siūlomas naujas požiūris, kuris derina tradicinių metodų paprastumą su pažangiomis giluminio mokymosi modelių galimybėmis pradiniame netekstinių elementų regionų aptikimo etape. Šis požiūris remiasi iš viršaus į apačią strategija, pradedant nuo bendrinių vaizdo savybių ir judant link detalių, taikant GUI specifinius vaizdo apdorojimo algoritmus. Jame taip pat įtrauktas iš anksto apmokytas „ResNet50" vaizdo klasifikatorius, kuris yra smulkiai derinamas regionų klasifikavimui [18].

Svarbu paminėti dar vieną aspektą – vaizdų (nuotraukų) apdorojimą prieš modelio apmokymą. Šie apdorojimo procesai gali apimti įvairius filtravimo metodus, segmentavimo technikas ir požymių išskyrimo algoritmus. Vaizdų apdorojimui dažnai taikomi tradiciniai

metodai, tokie kaip briaunų aptikimas, morfologinės operacijos ar spalvų erdvių transformacijos. Jie padeda lengviau ir efektyviau identifikuoti įvairius defektus, pavyzdžiui, lyginimo netikslumus, persidengimus ar netinkamą kraštų atskyrimą. Kiekvienai pateiktai nuotraukai gali būti taikomi tokie filtrai kaip Gauso žemųjų dažnių filtras, medianinis filtras ar dvipusis (angl. bilateral) filtras. Šie filtrai pašalina įvairius triukšmus iš vaizdų ir leidžia aiškiau išryškinti defektus, padarant juos geriau matomus analizės metu. Be filtravimo, gali būti taikomos ir segmentavimo strategijos. Vieni iš galimų metodų – semantinis segmentavimas bei slenksčių (angl. thresholding) nustatymas. Šie metodai leidžia tiksliau atskirti ir identifikuoti defektus, pavyzdžiui, UI (vartotojo sąsajos) komponentų ribas ar jų pažeidimus. Daugelis tyrimų rodo, kad derinant kelias apdorojimo technikas galima padidinti apmokytų modelių tikslumą maždaug 13–20 %, kas yra reikšmingas pagerėjimas. Tai tiesiogiai prisideda prie modelio patikimumo didinimo [19, 20].

Visi šie metodai gali būti pritaikomi defektų aptikimui, pavyzdžiui, atliekant elementų ribų nustatymą bei jų susikirtimų analizę. Tokiu būdu galima aptikti defektus, kai tekstas išeina už numatytų ribų, iškraipo elementų proporcijas ir pažeidžia bendrą sąsajos struktūrą. Naudojant mašininio mokymosi metodus objektų klasifikavimui, galima ne tik identifikuoti objektus, bet ir suskirstyti juos į mažesnes dalis, taikyti joms specifinius algoritmus ar net adaptuoti pačius mašininio mokymosi modelius tikslumo gerinimui.

1.10. Pokalbių daugiafunkcinių modelių taikymas

Dirbtinio intelekto pokalbių modeliai, sukurti šių dienų technologijų pagrindu, jau yra taip išpopuliarėję ir plačiai naudojami, kad beveik kiekvienas žmogus žino, kas tai yra, o daugelis jų jau yra naudojęsi šiomis sistemomis. Tokios technologijos kaip OpenAI GPT serija leido pokalbių modeliams generuoti itin žmogiškus atsakymus, geriau suprasti kontekstą ir ženkliai pagerino jų universalumą, tikslumą bei personalizavimą [21].

Šiuo metu pokalbių modeliai yra tokie plačiai taikomi, kad jie nebeapsiriboja vien atsakymais į žmogaus klausimus. Jie jau pritaikomi tokiose srityse kaip medicina, švietimas ir net vyriausybinės įstaigos, kur šios technologijos plačiai naudojamos siekiant palengvinti žmogaus darbą [22, 23].

Remiantis naudojimo statistika, pokalbių modelių populiarumas yra įspūdingas – net 87 % vartotojų juos vertina teigiamai ir yra patenkinti jų teikiama informacija [24].

Šiuolaikiniai pokalbių modeliai geba suprasti ne tik vartotojo įvestą tekstą, bet ir paveikslėlius, garso įrašus bei kitus informacijos tipus, juos apdoroti ir pateikti atitinkamą informaciją. Jų veikimo principas pagrįstas neuroninių tinklų architektūromis, leidžiančiomis apjungti ir vienu metu apdoroti kelis informacijos šaltinius – tekstinius, vizualinius ar garso duomenis. Tokie modeliai kaip Google Gemini, ChatGPT ir kiti panašūs [25] šiuo metu yra itin aktualūs ir svarbūs kuriamai sistemai, kadangi jie sugeba interpretuoti pateiktas nuotraukas ir grąžinti tikslius rezultatus.

Toliau apžvelgiamos šių modelių galimybės ir jų kokybė interpretuojant nuotraukas bei suvokiant jų turinį, nes tai – svarbiausias aspektas kuriamai sistemai. Šiuo metu stipriausias modelis, gebantis tiksliausiai interpretuoti vaizdus, yra GPT-o3 modelis (2025 m.), kuris

sugeba suprasti sudėtingus paveikslėlius ir net lenkia vidutinio žmogaus specialisto lygį standartizuotose vizualinėse užduotyse [21].

Kiekvienas modelis yra lyginamas pagal tai, kiek tiksliai jis interpretuoja paveikslėlius, nes tai svarbu siekiant nustatyti, kuris modelis geriausiai tiktų kuriamai sistemai (žr. 1 lentelę). Kaip jau minėta, geriausias rezultatas priklauso o3 modeliui, kuris pasižymi aukščiausiu tikslumu ir gebėjimu tiksliai interpretuoti vaizdus. Toliau seka Claude Opus 4 su panašiu tikslumo ir vizualinės interpretacijos lygiu, bei Gemini Pro 2.5, kurio rodikliai yra artimi pastarajam modeliu [21].

1 lentelė. DI pokalbių modelių palyginimai

Modelis	**Tikslumas (%)**	**Vizualinė interpretacija**
o3 (GPT)	99	Labai aukšta
Claude Opus 4	95	Aukšta
Gemini 2.5 Pro	95	Aukšta
Grok3	95	Aukšta
GPT-4	~79	Vidutinė-aukšta
Bard (Google)	~62	Vidutinė

Galima teigti, kad dirbtinio intelekto (DI) pokalbių modeliai, gebantys nuskaityti ir analizuoti paveikslėliuose esančią informaciją, yra itin naudingi ir veiksmingi klaidų bei defektų aptikimo procese. Šie modeliai ne tik gali identifikuoti lokalizacijos defektus, bet ir pasiūlyti galimus jų pataisymus, kuriuos būtų galima įgyvendinti automatiškai arba su minimaliu žmogaus įsikišimu. Šiandien tokių technologijų taikymo sritis yra labai plati, todėl ir kuriamai sistemai būtų naudinga išnaudoti šią technologiją visu savo potencialu.

1.11. Statinis testavimas

Iš jau minėtų lokalizacijos defektų aptikimo būdų svarbu paminėti ir statinį testavimą. Šis darbas gali būti atliekamas rankiniu būdu arba pasitelkiant tam skirtus įrankius, kurie šį procesą reikšmingai palengvina. Nors statinė analizė dažniausiai atliekama iš kodo pusės, prie kurios prieigą turi tik sistemos kūrėjai, vis dėlto ji yra itin svarbi, nes suteikia galimybę testuoti ir aptikti lokalizacijos defektus dar ankstyvame kūrimo etape.

Statinė analizė ir jai skirti įrankiai leidžia aptikti įvairius sistemos defektus dar prieš ją pilnai įgyvendinant. Šie įrankiai gali analizuoti kodą ir identifikuoti defektus, tarp jų – ir lokalizacijos klaidas, pavyzdžiui, į kodo struktūrą tiesiogiai įrašytas teksto reikšmes, kurios dažnai tampa lokalizacijos problemų šaltiniu. Taip pat galima aptikti struktūrinius neatitikimus, kylančius dėl netinkamai išverstų žodžių ar nevienodo jų ilgio, kas gali sukelti sąsajos išdėstymo problemų. Vienas populiariausių statinės analizės įrankių yra SonarQube, gebantis atlikti gilią kodo analizę ir palaikantis daugelį programavimo kalbų. Šis įrankis pasižymi dideliu tikslumu bei patikimumu aptinkant defektus, įskaitant lokalizacijos klaidas, tarptautinimo (angl. internationalization) problemas ir perteklinius kodo fragmentus. SonarQube analizė paremta sintaksinio medžio analizavimu bei valdymo eigos nagrinėjimu, siekiant aptikti tam tikrus pasikartojančius kodų šablonus, kurie gali būti susiję su lokalizacijos defektais [26].

Kalbant apie statinį testavimą, svarbu paminėti ir žmogiškąjį veiksnį. Rankinis testavimas yra paprasčiausias, tačiau tuo pačiu ir daug laiko reikalaujantis procesas, o jo kokybė priklauso nuo testuotojų kompetencijos. Nepaisant to, kad šiais laikais yra daug automatinių įrankių, rankinis testavimas išlieka svarbus tiek kodo peržiūros, tiek galutinio produkto vertinimo etapuose. Lokalizacijos defektų aptikimo kontekste rankinis testavimas yra itin reikšmingas, nes žmogus, dirbantis su sistema, turi platesnį kontekstinį supratimą – jis geba įvertinti kultūrinius niuansus, netinkamus vertimus ar turinį, kuris gali būti įžeidžiantis kitų šalių vartotojams [27]. Rankinė kodo peržiūra taip pat laikoma efektyvia, ypač kai ją atlieka patyrę specialistai, turintys aukštą kompetencijos lygį savo srityje. Tokie specialistai geba įvertinti, kokią įtaką tam tikri sprendimai gali turėti sistemai – ne tik techniniu, bet ir lokalizacijos požiūriu. Net ir mažos klaidos, susijusios su tekstų įterpimu ar jų išdėstymu, gali paveikti programos išvaizdą bei naudotojo patirtį [28, 29].

Statinė analizė yra tokia pat svarbi kaip ir automatinis testavimas. Ji leidžia išvengti klaidų ankstyvose sistemos kūrimo stadijose, o žmogiškasis testavimas bei kodo peržiūra suteikia galimybę aptikti subtilius, automatizuotoms sistemoms sunkiau atpažįstamus defektus. Šios žinios svarbios norint suprasti, kaip žmonės testuoja kuriamas sistemas, kokie yra jų lūkesčiai, bei kaip galima sumažinti rankinio testavimo apimtį, taip optimizuojant laiką ir išlaidas.

1.12. Panašių sprendimų analizė

Šiame tyrime buvo analizuoti panašūs sprendimai, kurie iš dalies arba pilnai įgyvendina mūsų užsibrėžtą tikslą. Buvo atrinkti du panašūs įrankiai: „Android Studio“ ir „Firebase Test Lab“.

Pradedant nuo „Android Studio“, svarbu paminėti, kad ši platforma turi „Layout Inspector" funkciją, kuri leidžia testuoti programėlės UI sąsają [30]. Šis įrankis suteikia galimybę simuliuoti įvairius įrenginius su skirtingais ekrano dydžiais, leidžiant lengvai pastebėti trūkumus keičiant ekrano dydžius ar pačius įrenginius. Galima atlikti paveikslėlių lyginimą – pavyzdžiui, lyginant tinkamą paveikslėlį, kuriame viskas atvaizduota teisingai, su kitu ekranu po kalbos pakeitimo. Taip galima pastebėti skirtumus, tokius kaip objektų ištempimas ar netilpimas į savo rėmus [30, 31]. Tačiau verta paminėti, jog šis tikrinimas yra atliekamas rankiniu būdu, o ne automatiškai, todėl tai reikalauja papildomo darbo.

Be „Layout Inspector“ funkcijos, „Android Studio“ siūlo dar vieną įrankį, vadinamą „UI Automator“. Naudojant šią funkciją, galima kurti vienetų testus, kurie aprašo tam tikrus scenarijus, kaip programa turėtų veikti ir kokio rezultato tikėtis [32]. Tai suteikia galimybę ištestuoti su lokalizacija susietas funkcijas, pavyzdžiui, patikrinti, ar tekstas tinkamai išverstas ir ar funkcijos veikia teisingai po kalbos pakeitimo.

Kitas svarbus testavimo įrankis yra „Firebase Test Lab" [33]. Skirtingai nuo kitų, šis įrankis atlieka testavimą debesyse, o ne naudotojo kompiuteryje, kas ženkliai pagreitina testavimo procesą. Panašiai kaip ir „Android Studio“, „Firebase Test Lab“ suteikia galimybę testuoti programėles įvairiuose įrenginiuose ir su skirtingais nustatymais. Klaidos, aptiktos testavimo metu, yra išsamiai dokumentuojamos ir pranešamos naudotojui.

Viena iš įdomiausių „Firebase Test Lab“ funkcijų yra „Robo script“. Naudojant šią funkciją, galima sukurti detalius scenarijus, kurie aprašo, kaip programėlė turėtų būti naudojama. Sukūrus tokį scenarijų, jį galima automatiškai paleisti skirtinguose įrenginiuose, turinčiuose įvairias savybes, ir stebėti, ar scenarijus veikia kaip numatyta [34]. Ši funkcija yra itin naudinga automatiniam testavimui, kai reikia tikrinti programėlės veikimą pagal nustatytas specifikacijas įvairiuose įrangose.

Be to, buvo išanalizuotos bendros „Android Studio“ ir „Firebase Test Lab“ įrankių specifikacijos, kurios padeda aiškiau suvokti, kokias funkcijas ir paslaugas šie įrankiai teikia [30, 31, 32, 33] (žr. 2 lentelė).

2 lentelė. Įrankių lyginimo kriterijai

Lyginamieji kriterijai	**Android studio**	**Firebase Test lab**
Testavimas naudojant nuotraukas	-	-
Automatizuoti Testai ir Automatinių Testų Kūrimas	-	+
Tekstinių Defektų Paieška	-	-
Testavimai Naudojant UI, Imituojant naudotoją	+	+
Palaikomos platformos	Android	iOS, Android

Tuo tarpu, skirtingai nei anksčiau minėti sprendimai, labiau priklausantys tradiciniam testavimui, kuris remiasi pačiu programos kodu, egzistuoja ir kitas įrankis – „Mobile-UI-Repair“ (MUI-R). Šis įrankis yra pažangus, paremtas giliojo mokymosi sprendimais, ir skirtas aptikti tam tikrus lokalizacijos vizualinius defektus mobiliųjų programėlių naudotojo sąsajose. Įrankis naudoja objektų aptikimo (angl. object detection) metodus. Jis paremtas RCNN architektūra, kuri leidžia ne tik identifikuoti matomas klaidas, bet ir tiksliai nurodyti jų vietą ekrane. Tai iš esmės leidžia programuotojams lengviau aptikti ir ištaisyti nustatytas klaidas. Buvo atlikti tyrimai, siekiant įvertinti šio įrankio tikslumą, ir pasiekti gana geri rezultatai – tikslumas siekė 87,7 %, o atkūrimo rodiklis (angl. recall) – 86,5 %. Šis pavyzdys puikiai demonstruoja giliojo mokymosi potencialą automatizuojant defektų aptikimą, kas yra tiesiogiai aktualu kuriant tokio tipo sistemą – lokalizacijos defektų aptikimo sistemą, paremtą ekrano nuotraukų analize, kuriose ir identifikuojami defektai [35].

Kaip matome, tiek „Android Studio“, tiek „Firebase Test Lab“ yra orientuoti į programėlių testavimą, prieinamą per jų programinį kodą arba tiesioginį naudojimą, o ne per jų nuotraukas. Tuo tarpu M-UI-R sprendimas, grįstas kompiuterine rega, yra tiesioginis pavyzdys to, ko siekia mūsų projektas – vizualinio defekto aptikimo iš ekrano nuotraukos. Šis palyginimas yra labai naudingas mūsų atliekamam tyrimui, nes suteikia supratimą apie tai, kokie artimi sprendimai jau yra sukurti ir plačiai naudojami šiuolaikinėje programėlių testavimo srityje.

2. Projektinė dalis

2.1. Veiklos sudėtis

2.1.1. Esama padėtis

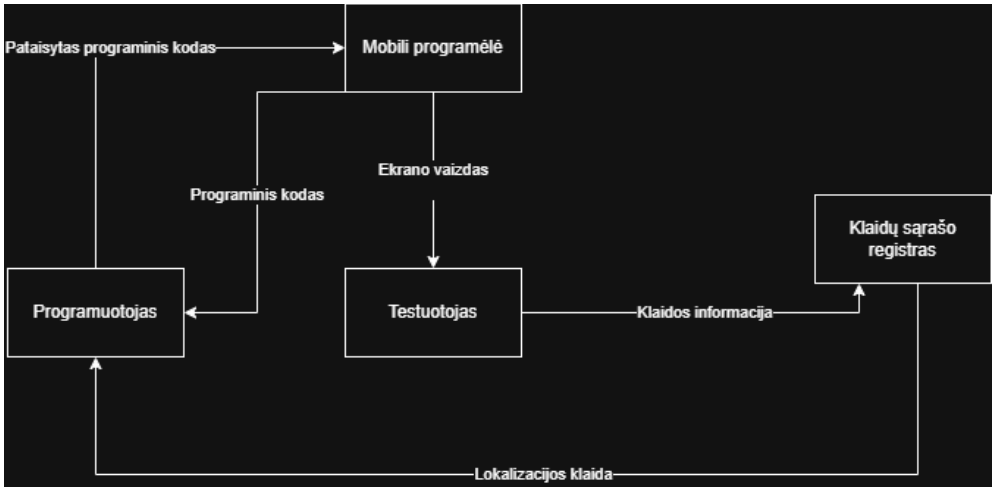
Šiuo metu situacija yra tokia, kad lokalizacijos testavimas dažniausiai atliekamas rankiniu būdu. Tai reiškia, kad asmuo paleidžia programėlę savo įrenginyje arba simuliuojamoje aplinkoje (angl. emulation), testavimo metu peržiūri kiekvieną sistemos langą ir tikrina, ar kalba neiškraipo vaizdo, nesugadina esamo išdėstymo, ar tekstas yra taisyklingai išverstas, taip pat ar teisingai pateikiami tekstiniai formatai, pavyzdžiui, datos, valiutos ir kt.

Jeigu sistema palaiko kelias kalbas, toks testavimas dar yra įmanomas skiriant pakankamai laiko. Tačiau kai programėlė yra tarptautinė ir palaiko dešimt ar daugiau kalbų, testavimas tampa labai imlus laikui, o tikimybė nepastebėti tam tikrų klaidų ženkliai išauga. Tai gali lemti naudotojų nepasitenkinimą ir neigiamai paveikti produkto reputaciją.

2.1.2. Veiklos kontekstas

Veiklos kontekstui iliustruoti buvo nubraižyta abstrakti veiklos konteksto diagrama (žr. 6 pav.). Joje matyti, kad procese dalyvauja programuotojai ir testuotojai. Programuotojai yra atsakingi už sistemos kūrimą ir klaidų taisymą, o testuotojai vertina tai, ką mato galutinis naudotojas, t. y. analizuoja mobiliosios programėlės vaizdą ir tikrina, ar jis atitinka nustatytus reikalavimus.

Jeigu testuotojas aptinka lokalizacijos ar kitą klaidą, jis ją užregistruoja klaidų sąraše (angl. bug tracking system). Iš šio sąrašo klaidos yra priskiriamos programuotojams arba jie jas pasiima savarankiškai ir atlieka taisymus, po to pateikdami atnaujintą programinį kodą. Šis procesas kartojamas cikliškai tol, kol sistema atitinka visus keliamus reikalavimus.



6 pav. Veiklos konteksto diagrama

2.1.3. Veiklos suskaidymas

Siekiant geriau suprasti, kaip sistemos kontekstas veikia šiuo metu, buvo sudaryta pagalbinė lentelė, kurioje aprašomos veiklos, jų suskaidymas ir paaiškinimai (žr. 3 lentelę).

3 lentelė. Veiklos suskaidymo lentelė

Pavadinimas	Išeinantis/Įeinantys	Aprašymas
Ekrano vaizdas	Išeinantis	Viena programėlės ekrano nuotrauka arba vaizdas.
Klaidos informacija	Išeinantis	Užregistruota lokalizacijos klaida, įtraukiama į darbų sąrašą, kurį reikia įgyvendinti.
Lokalizacijos klaida	Išeinantis	Identifikuota klaida, jos aprašymas, vieta ir atsiradimo sąlygos.
Programinis kodas	Išeinantis	Sistemos programinis kodas, kurį sukompiliavus sukuriamas programėlės failas.
Pataisytas programinis kodas	Įeinantis	Ištaisyta lokalizacijos klaida; atnaujinimas įkeliamas į mobiliąją programėlę.

2.2. Sistemos funkciniai reikalavimai

Sukurta sistema buvo internetinė svetainė su vidine dalimi, skirta duomenų apdorojimui, ir defektų aptikimo posisteme, atsakinga už lokalizacijos defektų nustatymą ir jų pateikimą. Sistemą sudaro daug funkcijų (žr. 7 pav.), kurios priskiriamos skirtingiems sistemos naudotojams.

Administratorius yra atsakingas už sistemos statistikos stebėjimą ir naudojimo analizę, siekiant identifikuoti tobulinimo galimybes. Administratorius turi šias funkcijas:

- Peržiūrėti bendrą statistiką – matyti, kiek naudotojų naudojasi sistema, kiek sukurta defektų ataskaitų, kiek jų yra naujų, bei kitą detalią informaciją.
- Peržiūrėti sistemos veikimo istoriją – analizuoti naudotojų veiklą, stebėti sistemos veikimą ir nustatyti galimas klaidas bei naudotojų reakcijas.
- Keisti roles – priskirti naudotojams roles, suteikiant papildomas sistemos valdymo galimybes.
- Peržiūrėti naudotojų veiklą – pasirinkti konkretų naudotoją ir analizuoti jo naudojimąsi sistema, nepažeidžiant privatumo, t. y. matant tik apibendrintus duomenis.

Aptarnavimo skyriaus darbuotojas yra atsakingas už pagalbos teikimą sistemos naudotojams, komunikaciją su jais ir grįžtamojo ryšio rinkimą. Jo funkcijos:

- Peržiūrėti užklausas – matyti kitų naudotojų pateiktas užklausas.
- Blokuoti naudotojus – riboti prieigą piktnaudžiaujantiems naudotojams.
- Atsakyti į užklausas – bendrauti su užklausų autoriais ir teikti pagalbą.

- Keisti užklausos būseną – pavyzdžiui, pažymėti užklausą kaip uždarytą.
- Keisti prioritetą – nustatyti užklausos svarbą, skiriant daugiau dėmesio svarbioms problemoms.

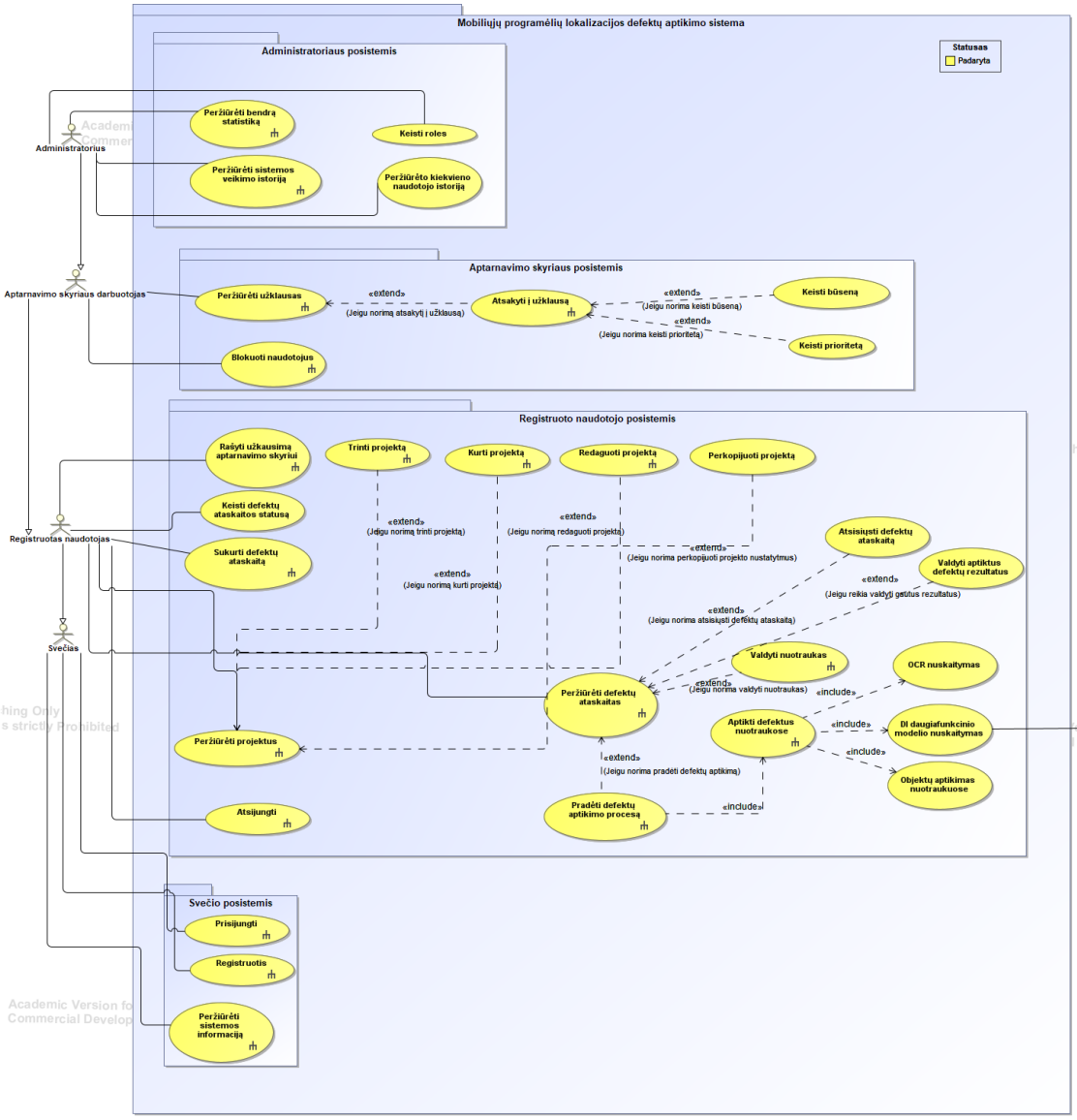
Svečias yra neregistruotas naudotojas, kuris gali susipažinti su sistema ir nuspręsti, ar ja naudotis. Jo funkcijos:

- Prisijungti – prisijungti prie sistemos turint paskyrą.
- Registruotis – susikurti naują paskyrą.
- Peržiūrėti sistemos informaciją – susipažinti su bendru sistemos aprašymu ir veikimo būsena.

Registruoti naudotojai turi daugiausia funkcijų ir aktyviausiai naudojasi pagrindine sistemos paslauga – lokalizacijos defektų aptikimu. Jų funkcijos:

- Kurti užklausas aptarnavimo skyriui – pateikti pagalbos prašymus, atsiliepimus ar kitus pranešimus.
- Keisti defektų ataskaitos būseną – nustatyti, ar ataskaita yra aktyvi, ar archyvuota.
- Kurti defektų ataskaitas – aprašyti parametrus ir įkelti analizuojamus vaizdus.
- Atsijungti – saugiai atsijungti nuo sistemos.
- Peržiūrėti projektus – analizuoti sukurtus projektus, kurie atitinka konkrečias programėlės versijas.
- Trinti projektus – pašalinti neaktualius projektus.
- Kurti projektus – įvesti naujų projektų informaciją.
- Redaguoti projektus – keisti projekto duomenis ir pridėti papildomus naudotojus.
- Kopijuoti projektus – dubliuoti esamus projektus su visa jų informacija.
- Peržiūrėti defektų ataskaitas – analizuoti nustatytus defektus.
- Atsisiųsti defektų ataskaitas – eksportuoti duomenis PDF (angl. Portable Document Format) formatu.
- Valdyti defektų rezultatus – peržiūrėti istoriją ir šalinti nereikalingus įrašus.
- Valdyti nuotraukas – įkelti, peržiūrėti ir trinti vaizdus.
- Pradėti defektų aptikimo procesą – inicijuoti analizę, vykdomą fone.
- Aptikti defektus nuotraukose – vykdyti defektų nustatymo procesą.
- Teksto atpažinimas – naudoti OCR technologiją tekstui nuskaityti.

- Dirbtinio intelekto modelių analizė – taikyti daugiafunkcius modelius defektams aptikti ir siūlyti taisymo būdus.
- Objektų aptikimas nuotraukose – naudojant kompiuterinę regą (angl. computer vision) ir neuroninius tinklus identifikuoti ir vizualiai pažymėti defektus.



7 pav. Panaudojimų atvejų diagrama

2.3. Sistemos nefunkciniai reikalavimai

Kaip ir visoms kuriamoms sistemoms, taip ir šiai buvo iškelti tam tikri nefunkciniai reikalavimai, kuriuos sistema turi tenkinti. Šie nefunkciniai reikalavimai yra:

- Sistema turi būti visiškai išversta ir prieinama anglų kalba.
- Sistemos puslapiai turi užsikrauti greičiau nei per 2 sekundes, esant geros kokybės naudotojo interneto ryšiui.
- Sistema turi būti veikianti bent 99,9 % laiko per metus.

- Sistema turi nuolat kurti atsargines kopijas, kad būtų galima atkurti duomenis įvykus nenumatytam incidentui.
- Sistema turi palaikyti didelį aktyvių naudotojų srautą – bent 1000 naudotojų vienu metu.
- Sistema turi būti suderinama su naujausiomis interneto naršyklėmis, ne senesnėmis nei 2022 m.
- Sistemos naudotojai turi būti aiškiai ir patogiai informuojami apie netikėtus nesklandumus informaciniais ar klaidų pranešimais bei kitomis priemonėmis.

Įgyvendinus šiuos nefunkcinius reikalavimus, bus užtikrintas optimalus sistemos veikimas, išlaikytas naudotojų skaičius ir sklandus defektų aptikimo procesas. Taip pat, net ir įvykus klaidoms, naudotojai bus tinkamai informuojami, o šios problemos ateityje bus sprendžiamos sistemos kūrėjų komandos.

2.4. Sistemos bendradarbiaujančios sistemos

Sistema gali veikti beveik autonomiškai savo aplinkoje, tačiau siekiant efektyvaus veikimo, duomenų bazė, klientinė dalis, vidinė sistema ir defektų aptikimo posistemė turi būti atskirtos ir tarpusavyje sąveikauti. Viena iš pagrindinių bendradarbiaujančių sistemų yra defektų aptikimo posistemėje integruota „OpenRouter“ sąsaja.

„OpenRouter“ suteikia sąsają su didžiaisiais dirbtinio intelekto modeliais, kurie ir naudojami šiame projekte. Per šią sąsają sistemai perduodamos ekrano nuotraukos, kontekstas ir instrukcijos, o mainais gaunami atsakymai iš dirbtinio intelekto modelių. „OpenRouter“ API yra integruota į defektų aptikimo posistemę ir naudojama kaip pagrindinis mechanizmas sąveikai su dirbtinio intelekto modeliais užtikrinti.

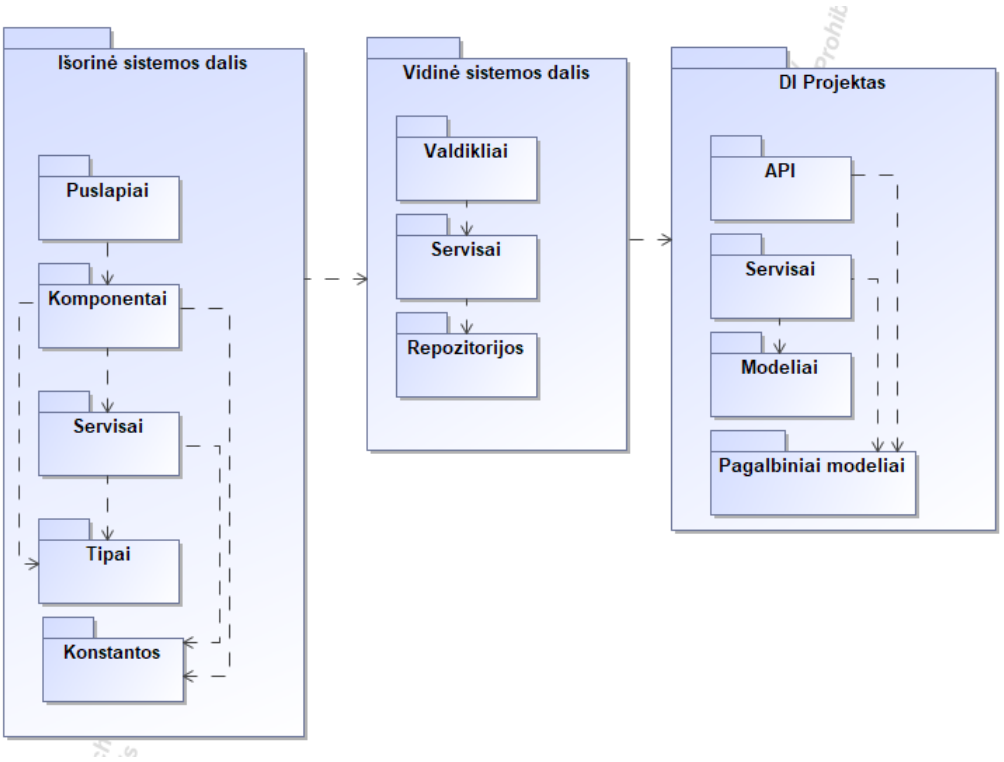
2.5. Sistemos statinis vaizdas

2.5.1. Paketų diagrama

Sistema sudaryta iš trijų pagrindinių dalių (žr. 8 pav.), kurios tarpusavyje sąveikauja. Šios sistemos dalys yra:

- Išorinė sistemos dalis – šios dalies paskirtis yra tiesiogiai sąveikauti su klientu, pateikti jam visą reikalingą informaciją ir sudaryti galimybes naudotis sistemos funkcijomis siekiant užsibrėžtų tikslų. Ši dalis turi būti patogi, modernaus dizaino ir lengvai valdoma.
- Vidinė sistemos dalis – ši dalis yra atsakinga už duomenų valdymą, resursų teikimą klientams per išorinę dalį, taip pat už sąveiką su defektų aptikimo posisteme. Be to, ji vykdo naudotojų autentifikaciją, sąveikauja su duomenų baze, apdoroja duomenis ir pateikia rezultatus.
- Dirbtinio intelekto posistemė – ši dalis yra atsakinga už defektų aptikimą. Joje integruoti reikalingi DI modeliai ir algoritmai. Posistemė gauna duomenis iš vidinės

serverio dalies, jų nekaupia, o apdorotus rezultatus grąžina atgal į vidinę dalį, kuri juos išsaugo.



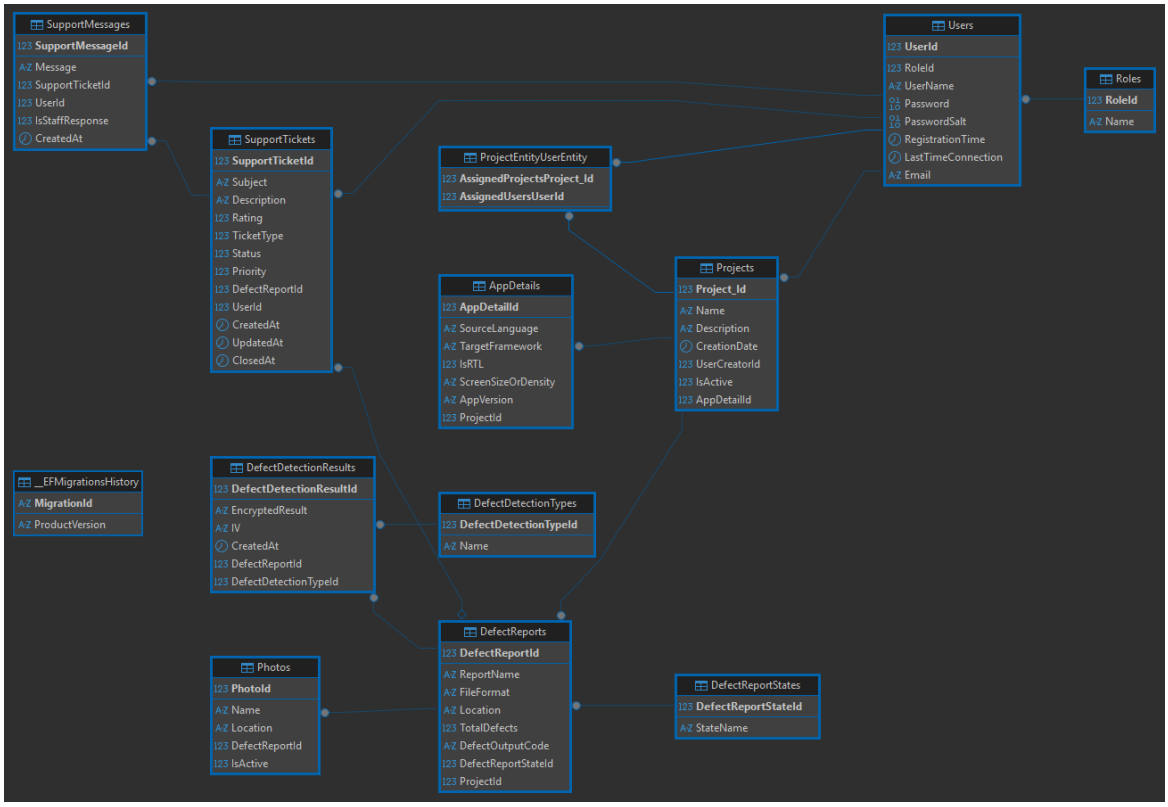
8 pav. Sistemos paketų diagrama

2.5.2. Duomenų bazės diagrama

Duomenys sistemoje saugomi SQL (angl. Structured Query Language) duomenų bazėje, kurios struktūra pateikta (žr. 9 pav.). Joje apibrėžta, kaip duomenys tarpusavyje susiejami, kokie ryšiai egzistuoja ir kokia informacija yra saugoma.

Klientų duomenys saugomi naudotojų lentelėje, kurioje pateikiama pagrindinė informacija apie naudotoją bei saugiai užšifruotas slaptažodis. Kiekvienas naudotojas gali teikti užklausas aptarnavimo skyriui, kurti projektus ir juos valdyti. Kiekvienas projektas gali turėti kelias defektų ataskaitas, o kiekviena defektų ataskaita gali turėti daugiau nei vieną rezultatą, jei defektų aptikimo procesas buvo vykdomas kelis kartus.

Siekiant apsaugoti klientų programėlių informaciją, defektų aptikimo rezultatai taip pat yra šifruojami, kad duomenų nutekėjimo atveju nebūtų atskleista jautri informacija. Be to, kiekvienai defektų ataskaitai priskiriamas tam tikras nuotraukų rinkinys. Pačios nuotraukos nėra saugomos tiesiogiai duomenų bazėje – jos laikomos atskiroje saugykloje, o duomenų bazėje saugoma tik informacija apie jų vietą bei būseną.



9 pav. Duomenų bazės diagrama

Apibendrinant galima teigti, kad sukūrus tokią duomenų struktūrą buvo realizuotas įrankis, gebantis efektyviai saugoti duomenis, juos susieti tarpusavio ryšiais ir užtikrinti jautrios informacijos saugumą ją šifruojant. Net ir įvykus duomenų nutekėjimui, jautrūs duomenys nebūtų lengvai prieinami ar paviešinti.

2.5.3. Sistemos diegimo diagrama

Sistema išbandymui ir naudojimui diegiama į atitinkamas aplinkas (žr. 10 pav.), su kuriomis sąveikauja tiek sistemos naudotojai, tiek kitos sistemos dalys. Diegimo procesas susideda iš kelių pagrindinių žingsnių, kuriuos būtina atlikti siekiant sėkmingai paleisti sistemą:

1. Sistemoje naudojama reliacinė SQL duomenų bazė, kurioje saugomi visi sistemos duomenys. Pirmiausia reikia pasirinkti SQL serverį, sukurti jame duomenų bazę ir gauti prisijungimo duomenis. Šie duomenys įrašomi į vidinės serverio dalies konfigūraciją. Naudojant „.NET Entity Framework Core“ biblioteką, atliekamos migracijos, kurių metu automatiškai sukuriamos visos reikalingos lentelės ir jų ryšiai.

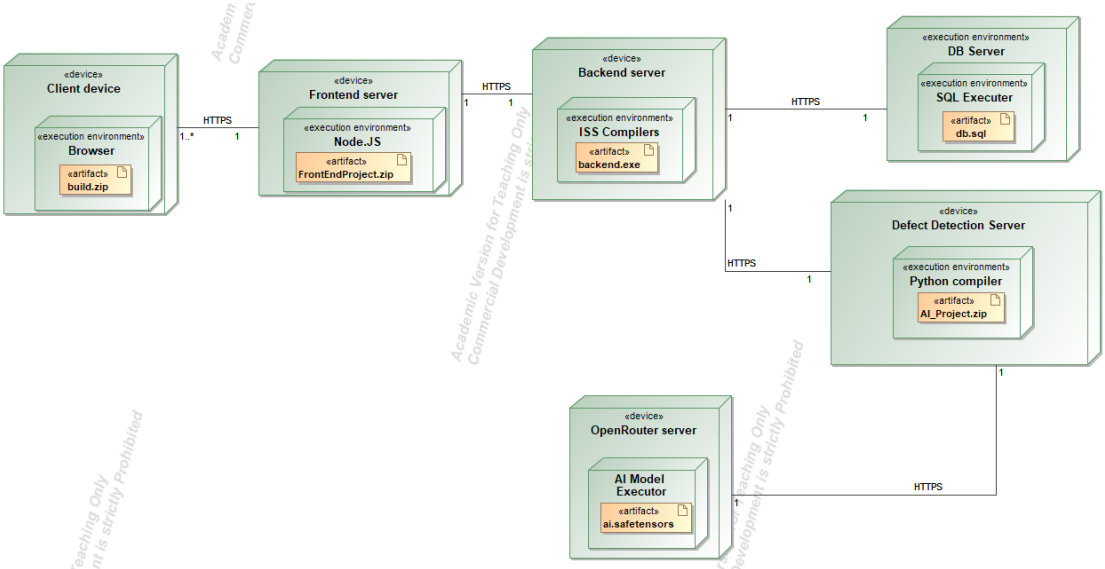
2. Toliau diegiama defektų aptikimo posistemė, veikianti kaip atskiras API. Ji gali būti talpinama naudojant konteinerio (angl. Docker) technologiją. Posistemė turi būti sukonfigūruota taip, kad būtų pasiekiama vidinei serverio daliai. Kadangi ši posistemė duomenų bazėje duomenų nesaugo, papildoma duomenų bazės konfigūracija nėra reikalinga.

3. Vidinė serverio dalis, sukurta naudojant .NET diegiama serveryje, palaikančiame šią technologiją. Ji taip pat gali būti diegiama naudojant konteinerį arba kitą pasirinktą sprendimą. Prieš diegimą būtina paruošti konfigūracinius failus skirtingoms

aplinkoms (pvz., vystymo, testavimo, produkcinei), kurie pateikiami kartu su projekto kodu.

4. Klientinė dalis diegiama serveryje, palaikančiame Next.JS, React ir TypeScript technologijas. Ji taip pat gali būti talpinama naudojant konteinerį. Svarbu tinkamai sukonfigūruoti ryšį su vidine serverio dalimi.

5. Sistema taip pat sąveikauja su „OpenRouter“ serveriu, kuris suteikia prieigą prie daugiafunkcinių dirbtinio intelekto modelių. Šis serveris bendrauja su defektų aptikimo posisteme: posistemė siunčia užklausas, o modelio serveris grąžina rezultatus, kurie vėliau yra apdorojami sistemoje.



10 pav. Diegimo diagrama

2.6. Sistemos dinaminis vaizdas

Pateikiamos kelios svarbiausios sistemos funkcionalumo veiklos diagramos, kurios atvaizduoja pagrindines sudėtingesnes sistemos funkcijas, naudojamas šiame tyrime ir eksperimente.

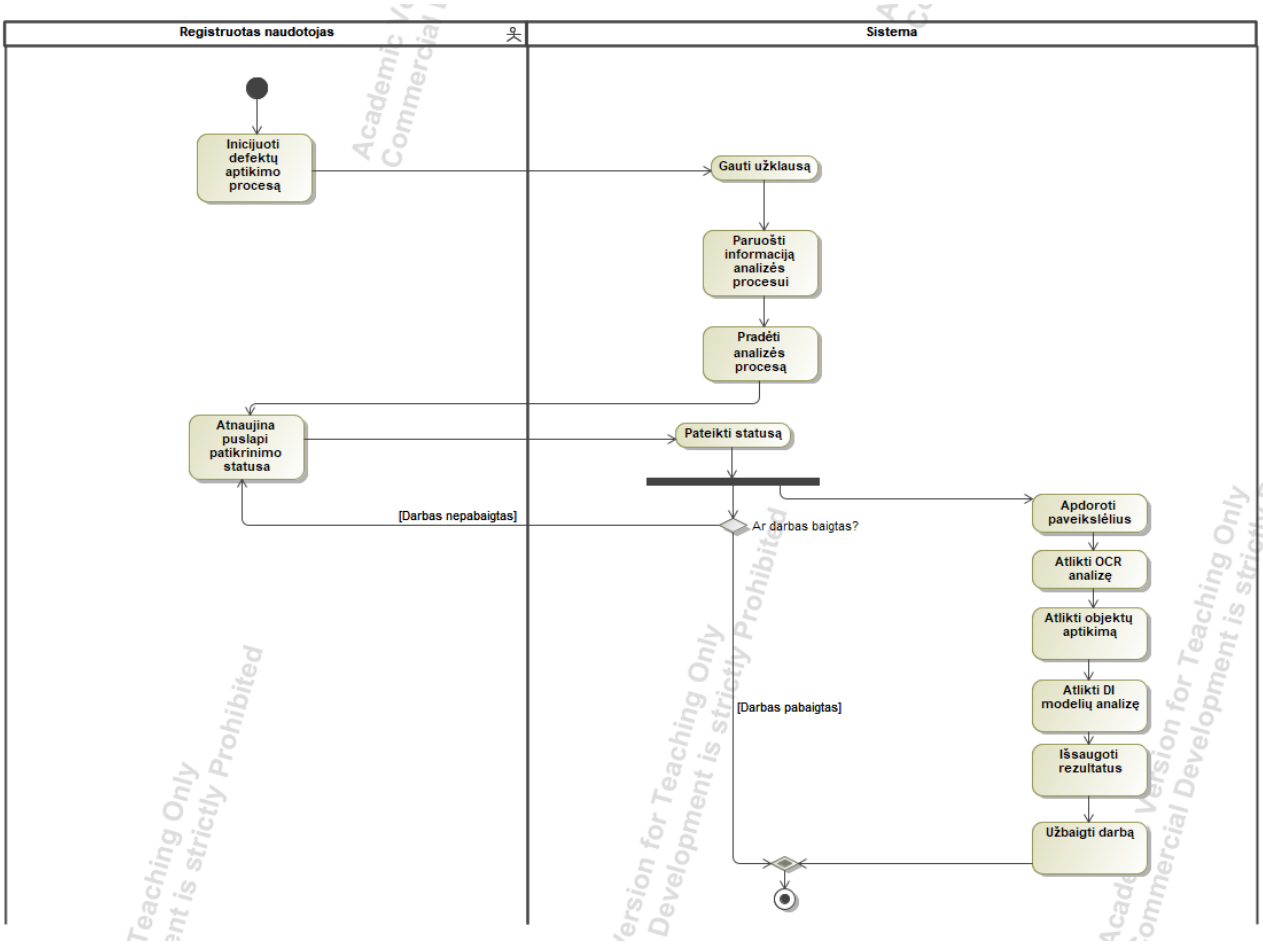
2.6.1. Veiklos diagramos

Vienas iš pagrindinių sukurtos sistemos procesų yra defektų aptikimo procesas (žr. 11 pav.), kurio metu analizė vykdoma fone. Sistemos naudotojas naudotojo sąsajoje inicijuoja analizę paspausdamas pradžios mygtuką, prieš tai paruošęs visus reikalingus duomenis.

Sistema, gavusi užklausą, pradeda vykdymą: suformuoja duomenų rinkinį, paruošia nustatymus ir perduoda juos analizės servisui, atsakingam už defektų aptikimą. Po užklausos išsiuntimo procesas vyksta fone, o klientas periodiškai kreipiasi į sistemą, siekdamas sužinoti analizės būseną.

Analizės procesas prasideda nuo nuotraukų apdorojimo ir paruošimo. Toliau duomenys perduodami per skirtingus analizės metodus: atliekamas teksto atpažinimas, objektų

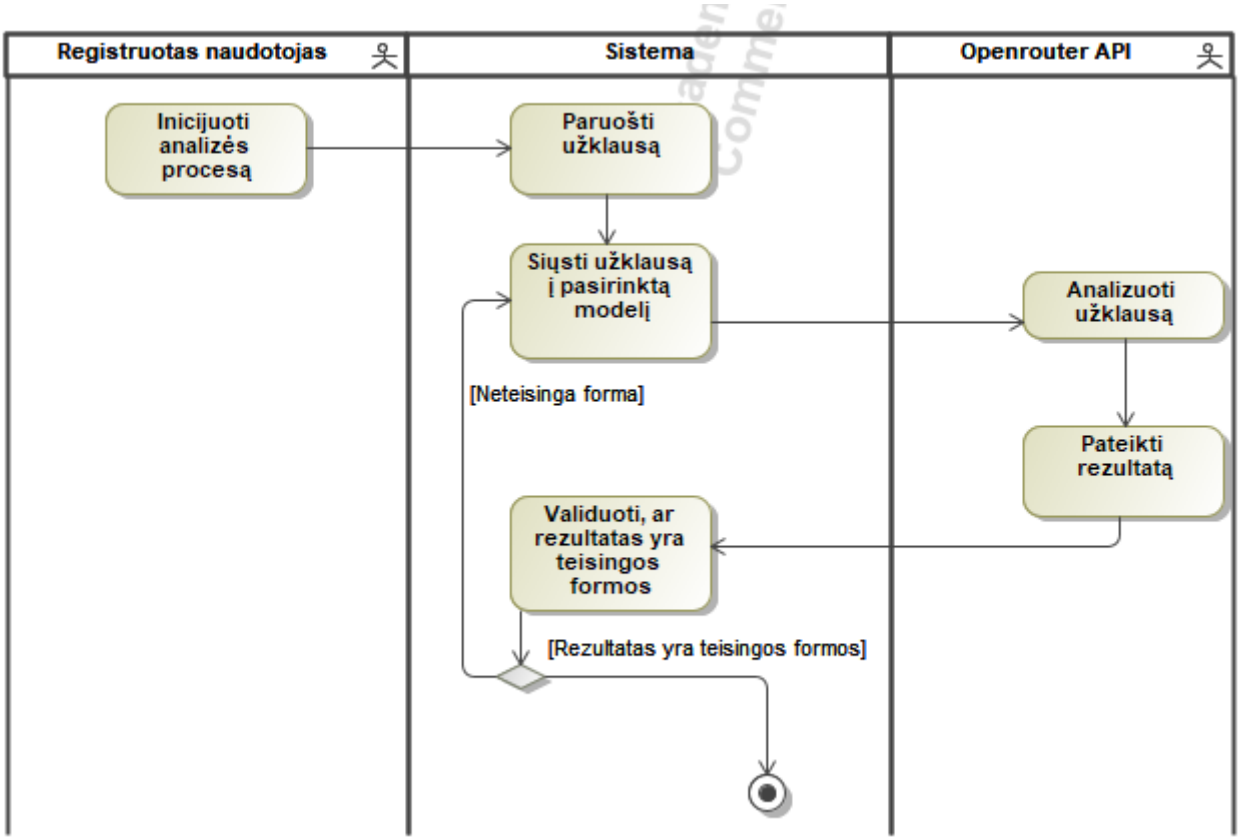
aptikimas ir dirbtinio intelekto daugiafunkcių modelių analizė. Galiausiai visi gauti rezultatai perduodami vidinei sistemai, kur jie saugiai išsaugomi duomenų bazėje. Tuomet atnaujinama proceso būsena, nurodant, kad analizė yra baigta, ir naudotojas gali peržiūrėti gautus rezultatus.



11 pav. Defektų aptikimo procesas

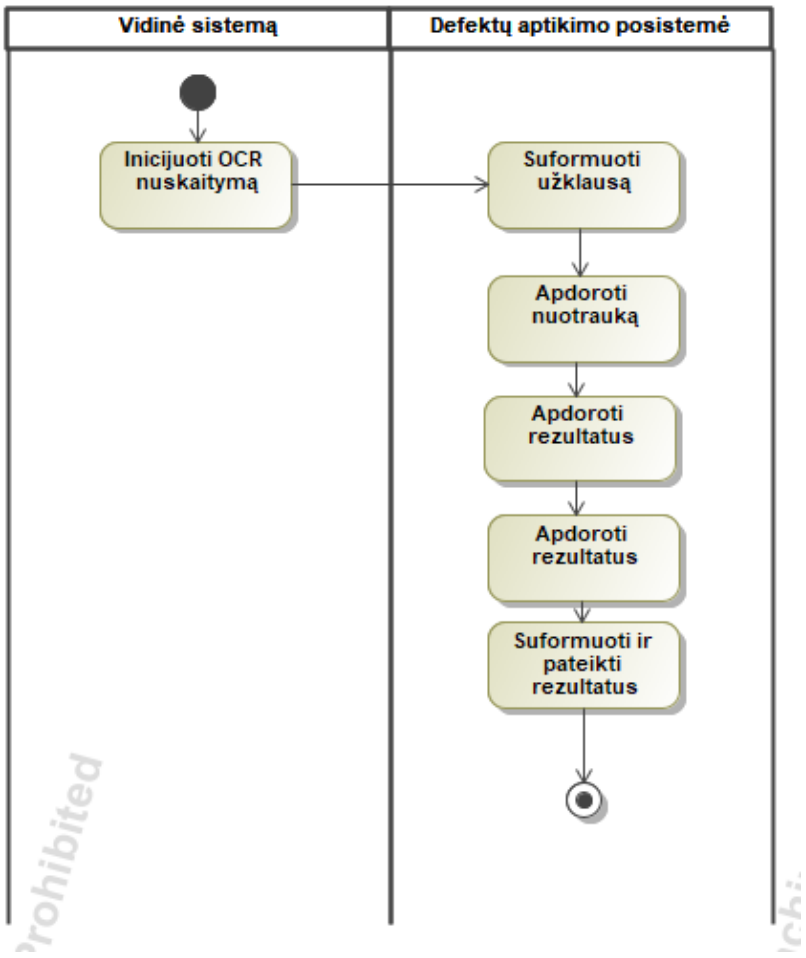
Vienas iš analizės būdų yra daugiafunkcio dirbtinio intelekto modelio panaudojimas (žr. 12 pav.). Šio proceso metu paruošiama užklausa, į kurią įtraukiama nuotrauka ir pateikiamos instrukcijos, nurodančios, kokie defektai turi būti aptikti. Tuomet užklausa siunčiama modelių tiekėjui, kuris ją analizuoja ir pateikia rezultatą.

Jeigu gautas rezultatas yra netinkamo formato ir jo negalima korektiškai interpretuoti, užklausa siunčiama pakartotinai, siekiant gauti tinkamai struktūruotą ir apdorojamą atsakymą.



12 pav. DI modelio analizė veiklos diagrama

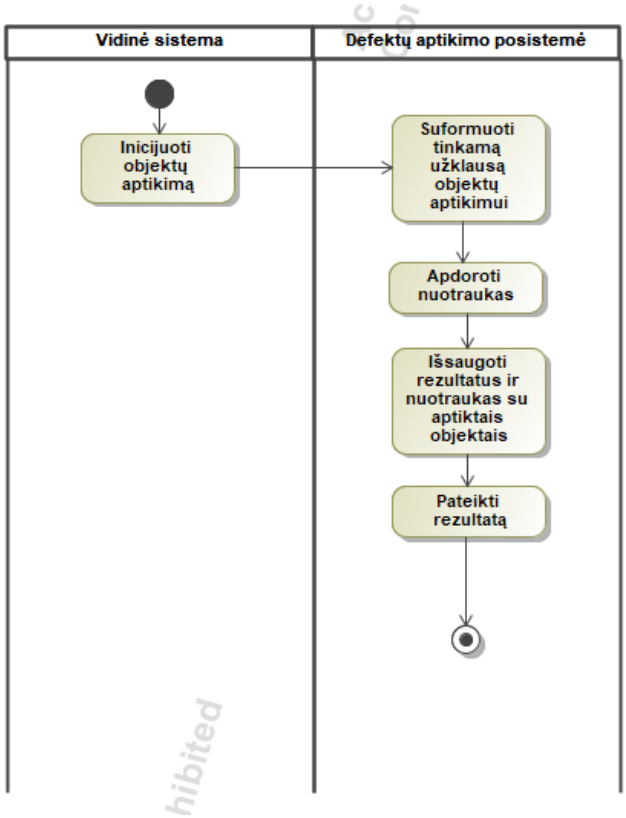
Vienas iš analizės etapų yra OCR nuskaitymas (žr. 13 pav.). Vidinė serverio dalis inicijuoja OCR analizę ir perduoda duomenis defektų aptikimo posistemei, kur užklausa suformuojama tinkama forma tolesnei analizei. Tuomet kiekviena nuotrauka yra apdorojama taip, kad joje esantis tekstas būtų išryškintas, ir paleidžiamas OCR analizės procesas.
OCR nuskaitymui naudojama „PaddleOCR“ biblioteka, kuri grąžina aptiktą tekstinę informaciją. Gauti rezultatai yra validuojami, pašalinami galimi artefaktai ir papildomai apdorojami siekiant užtikrinti kuo aukštesnę kokybę. Galiausiai apdoroti rezultatai perduodami atgal į vidinę sistemą.



13 pav. OCR nuskaitymo veiklos diagrama

Kitas žingsnis – naudojant jau apmokytą objektų aptikimo modelį atlikti pateiktų nuotraukų analizę (žr. 14 pav.). Kaip ir kituose etapuose, į defektų aptikimo posistemę perduodamas nuotraukų rinkinys, kuris joje transformuojamas į modeliui tinkamą formatą ir perduodamas analizei.

Modelis analizuoja nuotraukas ir, aptikęs lokalizacijos defektus, pateikia jų koordinates bei klasifikaciją, nurodydamas defekto tipą. Taip pat sistema sukuria vizualinius pažymėjimus paveiksluose, kuriuose aiškiai išskiriamos aptiktos problemos. Galiausiai visi rezultatai, įskaitant defektų duomenis ir pažymėtas nuotraukas, perduodami atgal į vidinę sistemą.

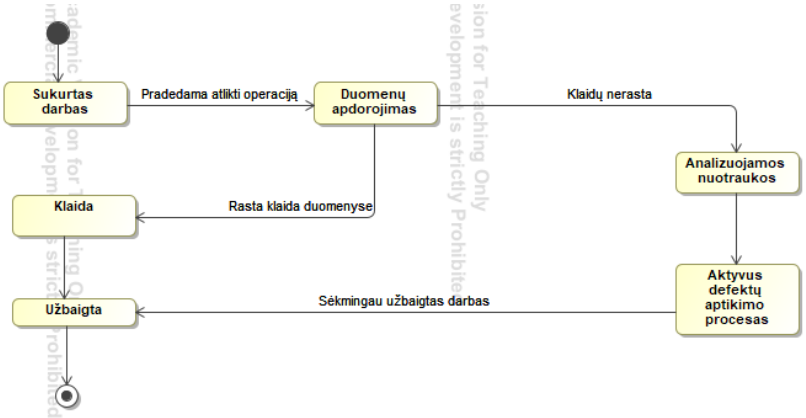


14 pav. Objektų aptikimo veiklos diagrama

2.6.2. Būsenų diagramos

Pateiktoje būsenos diagramoje (žr. 15 pav.) matyti, kad procesas vyksta nuosekliai: pirmiausia inicijuojamas nuotraukų apdorojimo etapas, tuomet prieš perduodant duomenis analizei atliekama užklausos validacija. Jei validacija sėkminga, duomenys perduodami defektų aptikimo posistemei, kurioje vykdomi visi algoritmai ir analizės metodai.

Atlikus visus veiksmus, lokalizacijos defektų aptikimo procesas užbaigiamas, o analizės būsena pažymima kaip baigta. Naudotojas, pridėjęs naujų nuotraukų ar pastebėjęs trūkumų, gali pakartotinai inicijuoti defektų aptikimo procesą.



15 pav. Aptikti defektus nuotraukose būsena

3. Tiriamoji dalis

3.1. Įvadas

Sistemų kūrėjai ir jų atstovai viso kūrimo ciklo metu susiduria su įvairiais programėlių trūkumais, nesklandumais ir kitais įvykiais, galinčiais sukelti tiek programėlės veikimo sutrikimus, tiek naudotojų nepasitenkinimą. Kokybiška, rinkai tinkama programėlė yra svarbi kiekvienai komandai, siekiančiai išlaikyti konkurencingumą ir gerą reputaciją, todėl taikomi įvairūs testavimo metodai. Testavimas gali būti tiek automatizuotas, tiek neautomatizuotas, pastarasis dažnai vadinamas statiniu testavimu.

Testavimas yra imlus laikui, ypač atliekamas statiniu būdu, ir neapsaugo nuo žmogiškųjų klaidų, tokių kaip nepastebėti ar praleisti defektai. Laiko sąnaudos dar labiau išauga, kai sistema palaiko kelias kalbas, nes kiekvienai kalbai reikia pakartotinai atlikti visą testavimo procesą, siekiant patikrinti, ar neatsirado naujų klaidų įvedus naujus naudotojo sąsajos elementus ar funkcionalumus. Naujos kalbos įdiegimas dar labiau apsunkina testavimo procesą.

Šiai problemai spręsti buvo pasiūlyta sukurti pagalbinę sistemą, gebančią, naudojant specializuotus algoritmus ir dirbtinio intelekto modelius, identifikuoti lokalizacijos defektus programėlėse remiantis jų ekrano vaizdais. Sistema pateikia aptiktas klaidas ir siūlo jų taisymo būdus sistemų kūrėjams. Remiantis sukurta sistema ir sistemų kūrėjų surinkta nuomone apie nagrinėjamą problematiką, atliekamas tyrimas, leidžiantis išsamiau įvertinti problemos mastą ir sprendimo efektyvumą naudojant siūlomą sistemą.

3.2. Tyrimo tikslas

Tyrimo tikslas – naudojant surinktus respondentų apklausos duomenis, sukurtus algoritmus ir dirbtinio intelekto modelius, ištirti, ar įmanoma automatizuoti ir paspartinti testavimo procesą lokalizacijos aspektu.

3.3. Tyrimo hipotezės

Pagrindinės šio tyrimo hipotezės yra:

1. Automatizuota sistema lokalizacijos defektus aptinka greičiau nei rankinis testavimas.

2. Dirbtinio intelekto modeliai geba kokybiškai identifikuoti įvairius lokalizacijos defektus ir pateikti jų taisymo pasiūlymus.

3. Objektų aptikimo ir dirbtinio intelekto modelių taikymas lokalizacijos defektų aptikime užtikrina aukščiausią rezultatų kokybę.

3.4. Tiriama aplinka

Šiuo tyrimu analizuojama programėlės ekrano vaizduose pateikta informacija, siekiant nustatyti lokalizacijos defektus, nepasitelkiant programinio kodo analizės, o remiantis tik vizualiniu turiniu ir kontekstu apie sukurtą sistemą. Tyrimo metu vertinama, ar taikomi metodai ir dirbtinio intelekto modeliai geba aptikti šiuos defektus:

- gramatines klaidas;

- teksto nukirpimus dėl dizaino sprendimų;
- teksto išėjimą už jam skirtų ribų;
- netinkamą datų formatą pagal pasirinktos kalbos standartus;
- neteisingą valiutos formatavimą;
- nepilnus vertimus;
- tekstinius artefaktus, likusius po vertimo proceso.

Lokalizacijos defektai atsiranda tuomet, kai į sistemą įvedus naują kalbą programėlė nesugeba tinkamai prie jos prisitaikyti. Dėl to atsiranda įvairių klaidų tiek programėlės vizualiniame pateikime, tiek kultūriniuose aspektuose, prie kurių sistema turėtų automatiškai adaptuotis. Tačiau dažniausiai to neįvyksta, todėl reikalingi papildomi testavimo ir taisymo procesai.

3.5. Tyrimo metodika

Tyrimas buvo sudarytas iš dviejų pagrindinių dalių. Pirmojoje dalyje atlikta apklausa: anonimiškai apklausti asmenys, kurie kasdien susiduria su nagrinėjama problematika. Šios dalies tikslas – nustatyti pagrindines problemas, su kuriomis susiduria respondentai, ir įvertinti, kokia kryptimi turėtų būti tobulinama kuriama sistema.

Antrojoje tyrimo dalyje taikomi algoritmai ir dirbtinio intelekto modeliai, siekiant įvertinti jų gebėjimą spręsti nagrinėjamą problemą. Atliekami eksperimentai su nuotraukomis, siekiant nustatyti, ar pasirinkti metodai yra tinkami, ar jie duoda naudą testavimo procesui ir ar verta juos toliau tobulinti.

3.5.1. Apklausos metodika

Buvo sukurta kokybinė apklausos forma, kurioje pateikiami klausimai, susiję su dabartine lokalizacijos testavimo situacija bei respondentų susidomėjimu siūlomais sprendimais šiai problemai spręsti. Ši apklausa leido geriau suprasti nagrinėjamą problematiką, identifikuoti reikalingus sprendimus ir įvertinti naujų algoritmų poreikį bei pagrindinius iššūkius, trukdančius kokybiškam lokalizacijos defektų testavimui.

Apklausos tyrimo metodika sudaryta iš kelių etapų:

- Respondentų grupės nustatymas – pirmiausia apibrėžiama tikslinė grupė, kuriai ši problema yra aktuali ir kuriai kuriamas sprendimas galėtų būti naudingas. Tai įvairūs mobiliųjų programėlių kūrimo procese dalyvaujantys specialistai: testuotojai, dizaineriai, programuotojai, produktų savininkai ir kiti.
- Klausimyno sudarymas – parengiami klausimai, skirti įvertinti esamą situaciją, nustatyti dabartinių procesų trūkumus bei išsiaiškinti respondentų nuomonę apie kuriamą sprendimą, jo aktualumą ir naudą. Klausimynas sudaromas naudojant „Google Forms“ platformą.

- Klausimyno platinimas – apklausa buvo platinama specializuotuose forumuose ir siunčiama tikslinės grupės atstovams. Respondentai taip pat buvo kviečiami per „Facebook“ ir „Discord“ platformas. Buvo surinkti apie 20–25 respondentus iš skirtingų sričių, siekiant gauti išsamesnį situacijos vertinimą.
- Rezultatų analizė – surinkus pakankamą atsakymų kiekį, apklausa uždaroma, o duomenys analizuojami. Kiekvienas klausimas nagrinėjamas atskirai, rezultatai apibendrinami ir pateikiamos išvados.
- Rekomendacijų sudarymas – remiantis gautais duomenimis ir analize, formuluojamos rekomendacijos dėl kuriamos sistemos tobulinimo ir nagrinėjamos problemos sprendimo.

3.5.2. Eksperimentinio tyrimo metodika

Eksperimentinio tyrimo metu buvo analizuojami trys defektų aptikimo būdai, naudojami sukurtoje sistemoje. Buvo vertinamos jų tobulinimo galimybės, atliekami palyginamieji tyrimai, įskaitant daugiafunkcių modelių tarpusavio palyginimą, algoritmų tikslumą ir veikimo greitį.

Pirmiausia, siekiant ištirti esamus sistemos algoritmus ir modelius, buvo sudaryta speciali duomenų aibė, kuri nebuvo naudota modelių apmokymui. Tai atskira duomenų aibė, skirta tik šiam tyrimui – modelių tikslumo vertinimui ir analizei. Buvo surinkta 15 nuotraukų, sudarytų iš programėlių ekrano vaizdų. Šios nuotraukos buvo atrinktos tiek rankiniu būdu, tiek dirbtinai modifikuotos naudojant specialius įrankius, siekiant gauti reprezentatyvią ir įvairią duomenų aibę. Tyrimui buvo naudojama tik defektų aptikimo posistemė, sukurta naudojant Python ir REST API (angl. Representational State Transfer Application Programming Interface), todėl buvo galima tiksliai įvertinti algoritmų ir modelių veikimą.

Turint paruoštą duomenų aibę, buvo pradėtas eksperimentinis tyrimas, siunčiant nuotraukas į skirtingus lokalizacijos defektų aptikimo metodus. Buvo analizuojami šie metodai ir atliekami jų palyginimai:

- Nuotraukos buvo siunčiamos į didžiuosius dirbtinio intelekto modelius: „Google Gemini 3 Flash“, „Mistral-Small“ ir „ChatGPT 4o mini“, naudojant vienodą instrukciją. Buvo vertinama, kaip tiksliai modeliai interpretuoja vaizdinę informaciją ir kokius sprendimus siūlo defektams taisyti.
- OCR technologijos analizė – buvo lyginami „PaddleOCR“ ir „Tesseract OCR“ rezultatai, vertinant jų gebėjimą atpažinti tekstą ir analizuoti jo kokybę.
- Apmokytas objektų aptikimo modelis – buvo vertinamas modelio gebėjimas identifikuoti vizualinius lokalizacijos defektus, pavyzdžiui, teksto išėjimą už mygtukų ribų ar kitus sąsajos išdėstymo pažeidimus.

Kiekvienam metodui buvo pateikta ta pati nuotraukų aibė, o gauti rezultatai buvo sistemingai dokumentuojami ir analizuojami. Buvo vertinami tokie aspektai kaip tikslumas, veikimo greitis, rezultatų kokybė ir bendras metodų efektyvumas, siekiant nustatyti jų pritaikomumą lokalizacijos defektų aptikimo procese.

3.6. Tyrimo siekiama nauda

Šiuo tyrimu siekiama, remiantis sistemų kūrėjų nuomone ir poreikiais bei pasitelkiant sukurtus algoritmus, egzistuojančius daugiafunkcius dirbtinio intelekto modelius ir kitus metodus, nustatyti lokalizacijos defektus programėlių ekrano vaizduose. Tikslas – geriau identifikuoti šiuos defektus, pasiūlyti galimas jų korekcijas ir sudaryti sąlygas sistemų kūrėjams efektyviau juos aptikti bei ištaisyti.

Tokiu būdu siekiama sutaupyti sistemų kūrėjų laiką, kuris galėtų būti skiriamas kitų sistemos dalių vystymui, bei paskatinti nebijoti diegti naujų kalbų programėlėse. Sukurti algoritmai leistų automatizuotai aptikti lokalizacijos defektus ir sumažinti pakartotinio testavimo poreikį.

3.7. Tyrimo vertinimo metrikos

Pagrindinės šio tyrimo metrikos, skirtos įvertinti, ar sukurta sistema tenkina iškeltus reikalavimus ir ar hipotezės pasitvirtina, yra šios:

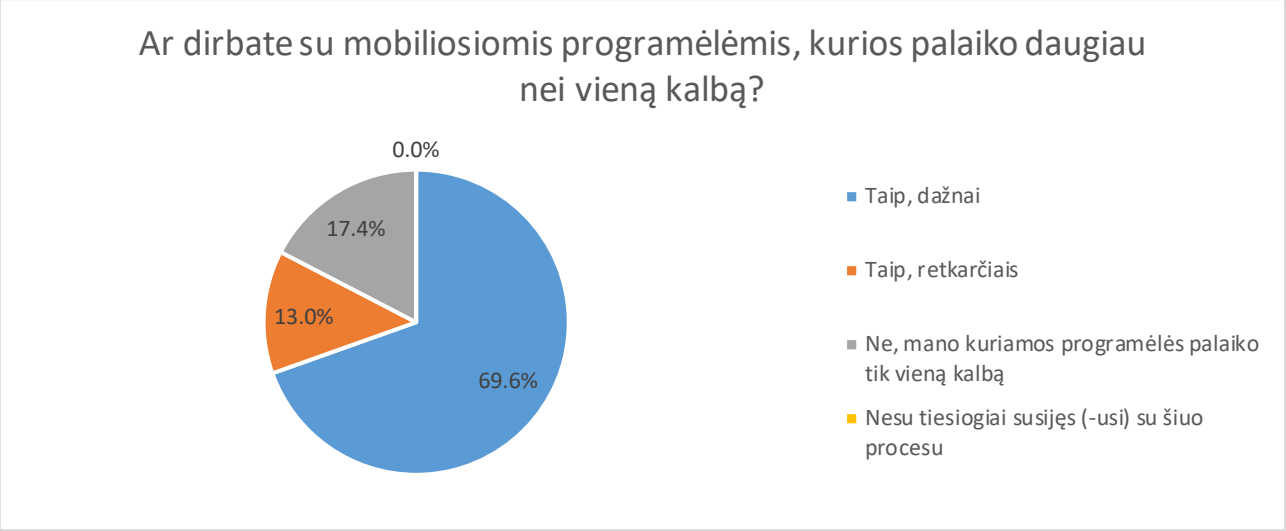
- Tikslumas – modelių gebėjimas tiksliai ir kokybiškai aptikti lokalizacijos defektus.
- Laiko trukmė – metrika, rodanti, per kiek laiko apdorojama kiekviena nuotrauka, leidžianti įvertinti sistemos veikimo greitį.
- Aptiktų defektų skaičius – rodiklis, parodantis, kiek defektų sistema aptiko; į šį skaičių įtraukiami tiek teisingai, tiek klaidingai identifikuoti atvejai.
- Dirbtinio intelekto modelių siūlomų korekcijų vertinimas – ekspertų grupė analizuoja, ar modelių pateikti pasiūlymai yra pagrįsti ir tinkami, siekiant įvertinti jų kokybę.
- Dirbtinio intelekto daugiafunkcių modelių gebėjimas aptikti žmogaus nepastebėtas klaidas.

3.8. Apklausos rezultatai

Surinkus respondentų atsakymus, vienas svarbiausių apklausos klausimų buvo, ar jų kuriamos programėlės palaiko daugiau nei vieną kalbą. Šio klausimo tikslas – nustatyti, kiek kalbų paprastai palaiko respondentų kuriamos programėlės, t. y. ar jos orientuotos į vienakalbę, ar daugiakalbę aplinką. Tai yra svarbus aspektas vertinant kuriamos sistemos aktualumą.

Didžioji dalis respondentų (žr. 16 pav.) nurodė, kad jų kuriamos programėlės dažniausiai palaiko daugiau nei vieną kalbą - tai sudarė 69,6 %. Atitinkamai 13 % respondentų pasirinko, kad jų kuriamos programėlės tik kartais palaiko daugiau nei vieną kalbą, o 17 % nurodė, kad jų kuriamos programėlės dažniausiai palaiko tik vieną kalbą.

Galima daryti išvadą, kad sistemų kūrėjų kuriamos programėlės dažniausiai palaiko daugiau nei vieną kalbą. Taip pat galima teigti, kad kuo daugiau kalbų palaiko programėlė, tuo didesnė tikimybė atsirasti lokalizacijos defektams, kuriuos būtina identifikuoti ir ištaisyti.

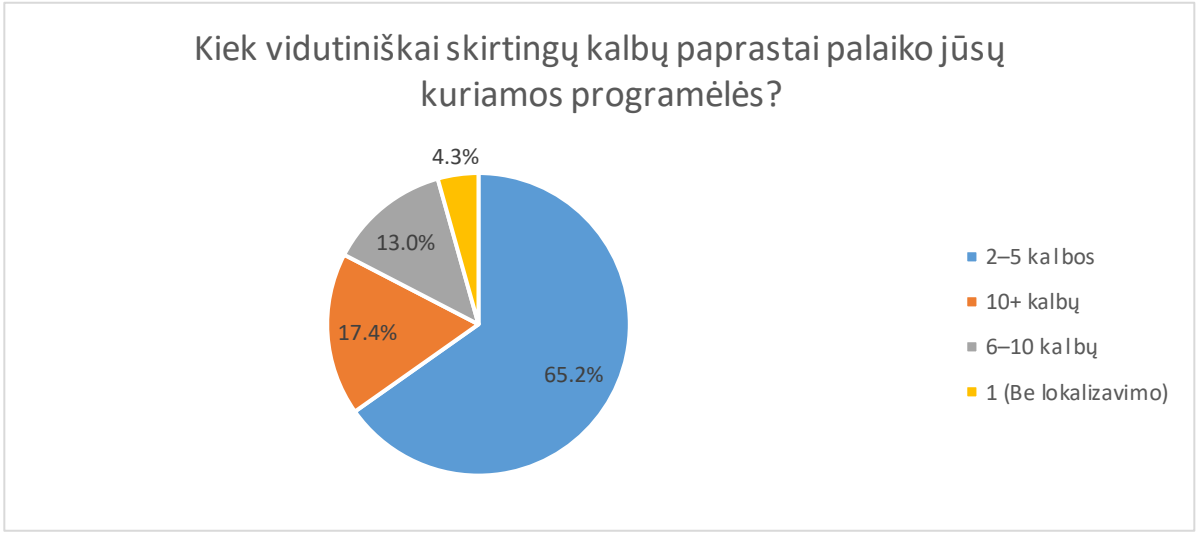


16 pav. Programėlių daugiakalbystės pasiskirstymas

Kitas apklausos klausimas buvo skirtas išsiaiškinti, kiek kalbų dažniausiai palaiko respondentų kuriamos programėlės. Šio klausimo tikslas - įvertinti, koks kalbų skaičius yra dažniausiai naudojamas ir kaip dažnai susiduriama su daugiakalbe aplinka. Ši informacija yra svarbi vertinant kuriamos sistemos poreikį ir pritaikomumą.

Iš pateiktų rezultatų (žr. 17 pav.) matyti, kad dažniausiai programėlėse naudojamos 2–5 kalbos – šį variantą pasirinko 65 % respondentų. Taip pat galima pastebėti, kad net 17 % respondentų nurodė, jog jų kuriamos programėlės palaiko 10 ar daugiau kalbų. Atsižvelgiant į tai, kad egzistuoja įrankiai, leidžiantys automatizuoti vertimo procesą, arba kad vertimus atlieka pačios naudotojų bendruomenės, toks rezultatas yra tikėtinas. Be to, 13 % respondentų nurodė, kad palaiko 6–10 kalbų.

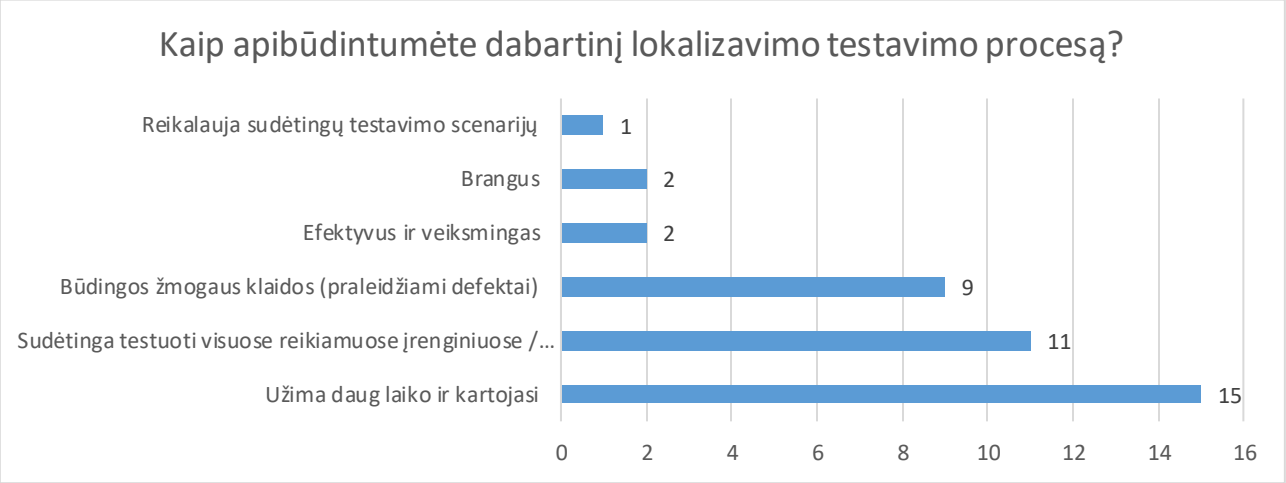
Tai rodo, kad dauguma respondentų kuriamų programėlių yra daugiakalbės, t. y. palaiko bent dvi kalbas. Dėl to kiekviena naujai įvesta kalba turi būti testuojama, siekiant įvertinti, ar ji neiškraipo esamo dizaino, o tai reikšmingai didina testavimo sąnaudas.



17 pav. Kalbų skaičiaus pasiskirstymas programėlėse

Toliau vienas iš svarbesnių klausimų ir buvo skirtas įvertinti, kaip respondentų komandos apibūdina savo dabartinį lokalizacijos testavimą. Šiuo klausimu buvo siekiama nustatyti pagrindinius esamo proceso trūkumus, į kuriuos būtų galima atsižvelgti kuriant naują sistemą (žr. 18 pav.).

Didžioji dalis respondentų nurodė, kad lokalizacijos testavimas yra itin laikui imlus ir pasikartojantis procesas, nes tuos pačius elementus tenka tikrinti ne vieną kartą. Šį variantą pasirinko net 15 respondentų. Antras pagal dažnumą pasirinkimas buvo susijęs su sunkumais testuojant lokalizacijos defektus skirtinguose įrenginiuose ir ekrano dydžiuose. Tai pagrįsta problema, kadangi išmaniųjų įrenginių yra labai daug, o visų jų ištestuoti dažnai neįmanoma nei dėl laiko, nei dėl finansinių išteklių trūkumo. Trečias pagal populiarumą atsakymas buvo susijęs su testavimo patikimumu - respondentai nurodė, kad dėl žmogiškojo faktoriaus testavimo procesas nėra visiškai patikimas. Šį variantą pasirinko 9 respondentai. Dėl to net ir nedidelės klaidos, tokios kaip rašybos ar smulkūs lokalizacijos netikslumai, gali likti nepastebėtos.

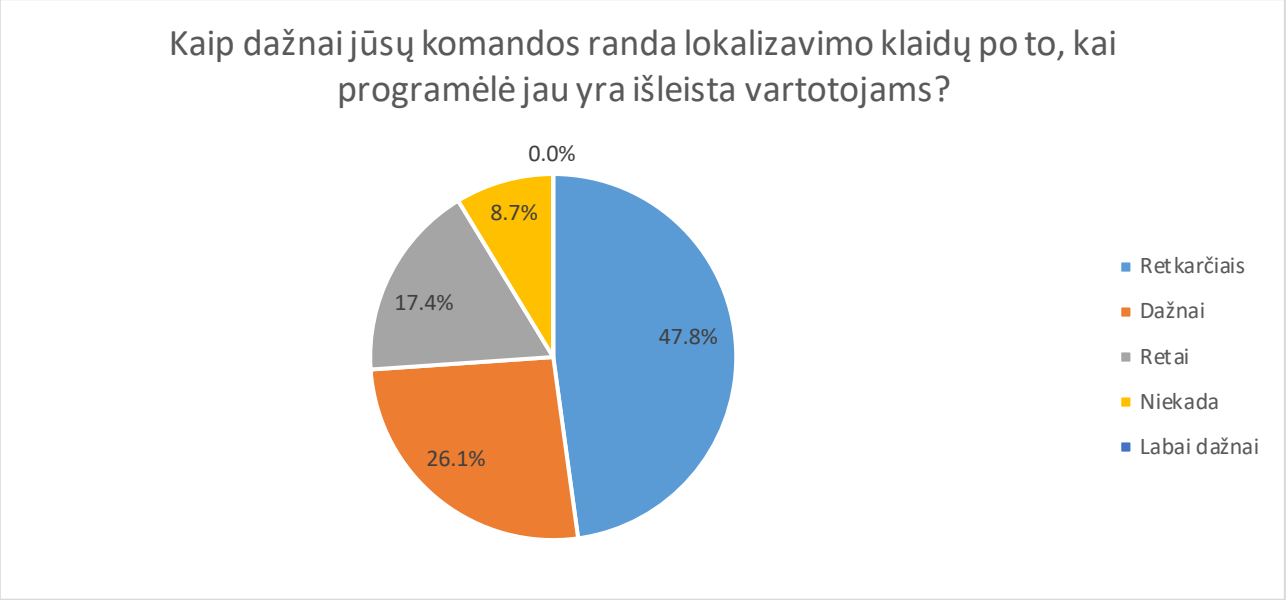


18 pav. Lokalizacijos testavimo iššūkiai komandose

Kitas klausimas buvo gana svarbus norint išsiaiškinti, kaip dažnai lokalizacijos defektai yra aptinkami jau po programėlės paleidimo. Šio klausimo tikslas - nustatyti, kiek klaidų lieka nepastebėta testavimo proceso metu ir pasireiškia tik pasibaigus paleidimui.

Iš diagramos (žr. 19 pav.) matyti, kad 47,8 % respondentų nurodė, jog lokalizacijos defektus po programėlės išleidimo aptinka kartais. Tai svarbus rodiklis, parodantis, kad nors testavimo procesas yra vykdomas, vis tiek lieka neišvengtų klaidų. Mažesnėse sistemose tai gali neturėti reikšmingo poveikio, tačiau didesnėms ar plačiai naudojamoms programėlėms tokios klaidos gali turėti neigiamą įtaką reputacijai.

Be to, 26 % respondentų teigė, kad lokalizacijos defektai jų sistemose aptinkami dažnai, kas rodo poreikį efektyvesniems testavimo sprendimams lokalizacijos srityje. Likusieji respondentai nurodė, kad lokalizacijos defektai aptinkami labai retai (17 %), o 8,7 % apklaustųjų teigė, jog po programėlės išleidimo tokių defektų apskritai nepasitaiko.



19 pav. Lokalizacijos defektų aptikimo dažnumas po paleidimo

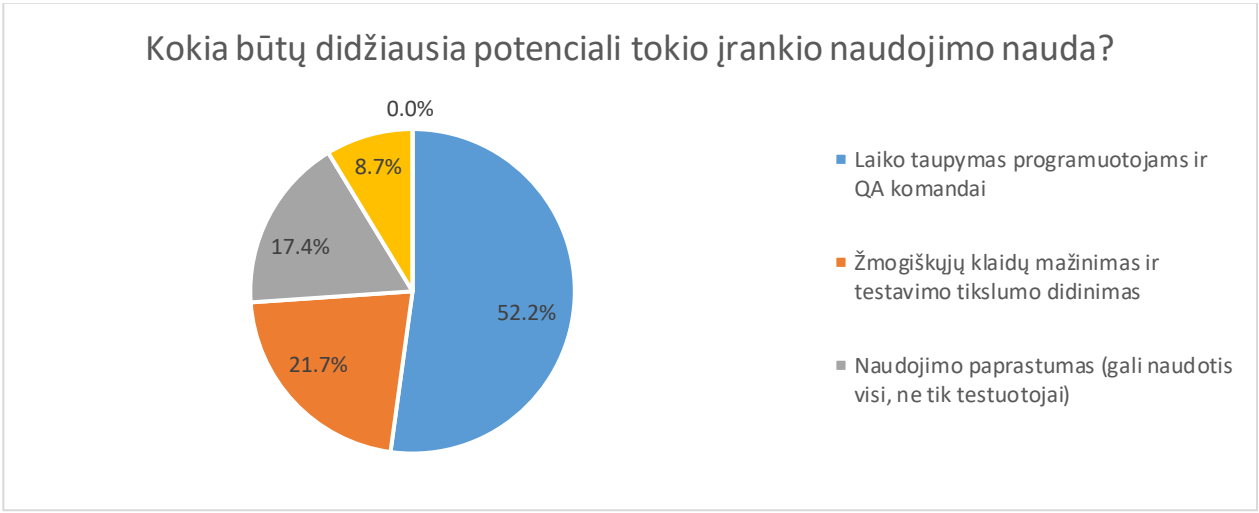
Tolimesnis klausimas, kuriame galima pasirinkti kelis variantus ir buvo skirtas išsiaiškinti, kokius lokalizacijos defektus respondentai dažniausiai aptinka savo kuriamose sistemose bei kokius trūkumus pastebi. Šis klausimas yra svarbus kuriant sistemą, nes leidžia suprasti, su kokiomis problemomis susiduria kūrėjų komandos ir į ką verta atkreipti dėmesį ieškant naujų sprendimų.

Iš rezultatų (žr. 20 pav.) matyti, kad dažniausiai pasitaikantys lokalizacijos defektai mobiliųjų programėlių kūrėjų nuomone yra susiję su naudotojo sąsajos problemomis: tekstas išeina iš rėmų, dizainas nesiderina su kitomis kalbomis nei pagrindinė. Šį variantą pasirinko 15 respondentų. Antroje vietoje pagal dažnumą yra įvairios vertimo klaidos - gramatikos, rašybos, skyrybos ir panašios klaidos. Šį variantą nurodė 11 respondentų. Trečioje vietoje - funkcionalūs nukrypimai dėl kultūrinių skirtumų, pavyzdžiui, tekstų išdėstymas iš dešinės į kairę, būdingas Vidurio Rytų ir kai kuriems Afrikos regionams. Kadangi tai yra didelės populiacijos šalys, sistemų kūrėjai yra suinteresuoti pritaikyti programėles ir šioms rinkoms. Šį variantą pasirinko 8 respondentai. Toliau nurodomi įvairūs datos formatų trūkumai, kurie atsiranda, kai programėlė platinama keliose šalyse su skirtingais datos formatais. Šį variantą pasirinko 7 respondentai. Mažiausiai pasitaikantys defektai - įvairios dizaino deformacijos dėl sutrumpinto ar nukirpto teksto - buvo nurodyti 4 respondentų

20 pav. Lokalizacijos defektų tipų pasiskirstymas

Vienas iš paskutinių klausimų buvo skirtas išsiaiškinti, kokį potencialą respondentai įžvelgia galimai lokalizacijos defektų aptikimo sistemoje, t. y. kokius privalumus jie matytų, jei būtų sukurta tokia sistema. Buvo pateikti keli galimi variantai, iš kurių respondentai galėjo pasirinkti kelis.

Iš rezultatų matyti (žr. 21 pav.), kad didžiausią naudą respondentai mato laiko taupymą tiek sistemų programuotojams, tiek testavimo komandoms. Šį variantą pasirinko 52 % respondentų. Antras pagal populiarumą variantas buvo žmogiškųjų klaidų sumažinimas testavimo metu. Trečias pasirinkimas - lengvas naudojimasis sistema, kurį pasirinko 17,4 %, o ketvirtas pagal populiarumą buvo didesnis testų padengimas. Tokį variantą pasirinko 8,7 %.



21 pav. Potencialūs kuriamos sistemos privalumai

3.9. Apklausos išvados

Sudarius klausimyną ir surinkus respondentų atsakymus, buvo nustatyta, kad mobiliųjų programėlių lokalizacijos defektai egzistuoja ir juos taisoma įvairiais būdais. Vis dėlto net ir po programėlės išleidimo dalis defektų išlieka. Iš respondentų atsakymų matyti, kad egzistuoja poreikis naujiems sprendimams, kurie galėtų spręsti tradicinių testavimo įrankių trūkumus, tokius kaip ne visi defektai aptinkami, reikalingos specifinės įgūdžių žinios bei daug laiko sąnaudų. Todėl kuriami nauji algoritmai ir sprendimai turi pagrindą egzistuoti, nes jie būtų naudinga sistemų kūrėjų komandai, o šis poreikis aiškiai atsispindi apklausos rezultatuose.

3.10. Bendros tyrimo išvados

1. Atlikus tyrimą išryškėjo pagrindinė problema – sistemų kūrėjai siekia kuo kokybiškiau ištestuoti savo sistemas, tačiau laiko sąnaudos ir kiti ištekliai neleidžia to atlikti pakankamai efektyviai. Viena iš sudėtingiausių sričių yra lokalizacijos testavimas, kai tikrinamos skirtingos kalbos programėlėje. Naujos kalbos įvedimas reikalauja pakartotinai testuoti tas pačias sistemos dalis, todėl procesas tampa itin imlus laikui, o žmogiškasis faktorius didina tikimybę, kad dalis klaidų liks nepastebėtos.

2. Buvo iškeltas tikslas rasti būdą automatizuoti lokalizacijos defektų aptikimo testavimo procesą, pritaikant modernius sprendimus, siekiant sumažinti darbo sąnaudas.

3. Siekiant įvertinti šio tikslo įgyvendinimą ir jo naudą, atliekamas eksperimentinis tyrimas, kuriame analizuojami trys algoritmai ir procesai, skirti lokalizacijos defektams aptikti, identifikuoti ir pasiūlyti galimus taisymo būdus. Tai leidžia sumažinti laiko sąnaudas ir sudaro sąlygas sistemų kūrėjams lengviau bei efektyviau identifikuoti ir ištaisyti klaidas.

4. Eksperimentinė dalis

4.1. Įvadas

Atlikus bendrą tyrimą ir apklausą, kurioje 47,8 % respondentų nurodė susiduriantys su lokalizacijos defektais, buvo atliktas eksperimentinis tyrimas, siekiant įvertinti, ar šiuolaikiniai metodai ir naujų technologijų taikymas gali automatizuotai spręsti šią problemą. Tokiu būdu siekiama sumažinti programinių sistemų testuotojų laiko sąnaudas, pagerinti testavimo procesą ir galimai tiksliau aptikti defektus, taip sumažinant žmogiškojo faktoriaus įtaką.

4.2. Eksperimento eiga

Šio eksperimento eiga buvo sudaryta iš kelių žingsnių, kurie yra:

- Surinkti nematytą duomenų aibę, kurioje pateikiami įvairių tipų lokalizacijos defektai, siekiant užtikrinti atvejų įvairovę.
- Iš anksto apibrėžti dalį lokalizacijos defektų, esančių ekrano nuotraukose, o dalį palikti identifikuoti rankiniu būdu, kad būtų galima įvertinti algoritmų ir modelių gebėjimą juos aptikti.
- Naudojant visus tris defektų aptikimo metodus, pateikti nuotraukas analizei ir gauti rezultatus. Atliekami šie eksperimentai:
    - Daugiafunkcių modelių analizė, naudojant ekonomiškus, tačiau galingus modelius: „Gemini 3 Flash“, „Mistral-small-3.2-24b“ ir „ChatGPT 4o mini“;
    - OCR technologijos taikymas, lyginant dvi populiarias atvirojo kodo bibliotekas – „Tesseract OCR“ ir „PaddleOCR“, siekiant aptikti gramatines ir kitas tekstines klaidas;
    - Specializuoto, apmokyto kompiuterinės modelio taikymas, gebančio identifikuoti, klasifikuoti ir vizualiai pažymėti lokalizacijos defektus nuotraukose.
- Atlikus analizę, visi rezultatai dokumentuojami, pateikiant juos lentelėse, grafikuose ir kituose formatuose, įskaitant paklaidų vertinimą.
- Galiausiai atliekama rezultatų analizė ir nustatoma, ar iškeltos hipotezės pasitvirtino.

4.3. Eksperimentinė aplinka

Darbui su algoritmais ir vizualinių objektų aptikimo modelių apmokymui buvo naudojamas vienas nešiojamasis kompiuteris. Jo techninės charakteristikos: „Intel Core i7-6700HQ“ 2,6 GHz procesorius, 16 GB DDR4 operatyvioji atmintis ir atskira vaizdo plokštė „GTX 1070“ su 8 GB VRAM. Taip pat buvo naudojama „Windows 10“ operacinė sistema, kurioje atlikti visi modelių apmokymai ir pats tyrimas.

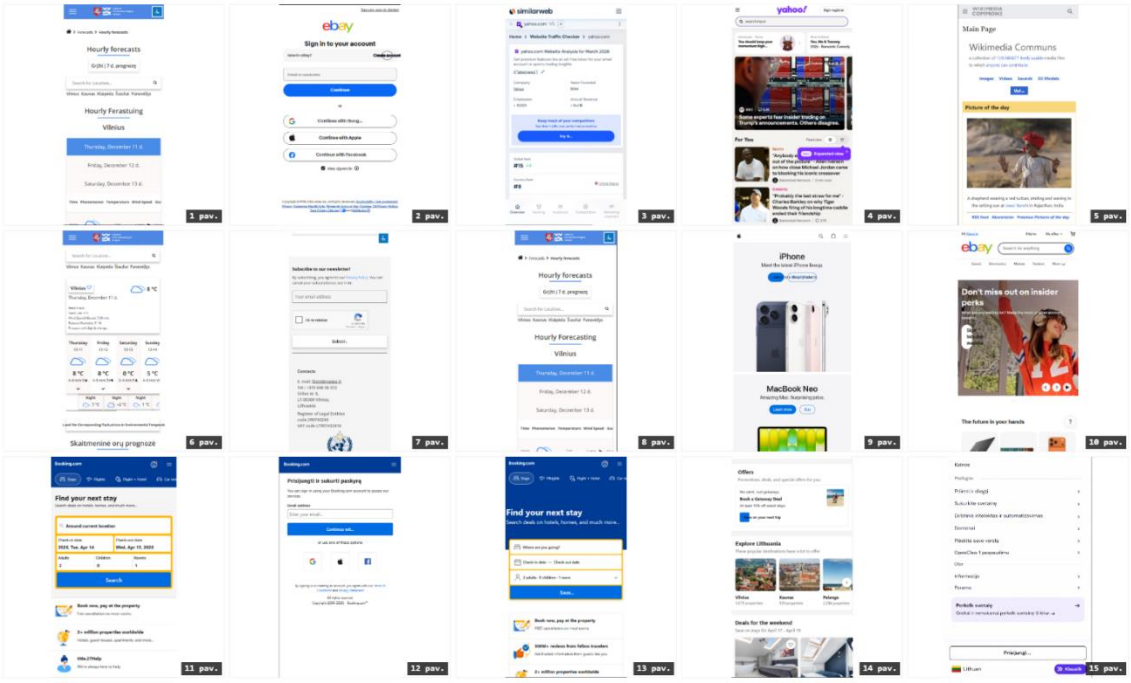
Kaip minėta, algoritmai buvo realizuoti naudojant „Python“ programavimo kalbą (3.11 versija), o visi analizuojami metodai buvo įgyvendinti per REST API.

4.4. Naudojama vaizdo medžiaga eksperimentui

Šiam eksperimentui buvo atrinkta duomenų aibė, su kuria buvo testuojami sistemos algoritmai ir metodai (žr. 22 pav.). Ši aibė sudaryta iš mobiliųjų programėlių vaizdų, kuriuose yra įvairių lokalizacijos defektų, kuriuos siekiama aptikti naudojant sukurtus algoritmus. Kadangi realių duomenų su aiškiai identifikuotais defektais gauti yra sudėtinga, duomenų aibė buvo suformuota dirbtinai – rankiniu būdu sukuriant defektus, atitinkančius realias situacijas.

Ši duomenų aibė nebuvo naudota modelių apmokymo ar kituose procesuose, todėl yra tinkama objektyviam eksperimentiniam vertinimui. Svarbu pažymėti, kad dalis vaizdų buvo paimti iš skirtingų šaltinių, įskaitant atvejus, kai programėlės sąsaja nebuvo pilnai lokalizuota į anglų kalbą. Tai leidžia įvertinti, kaip algoritmai reaguoja į akivaizdžius lokalizacijos neatitikimus.

Duomenų aibė daugiausia sudaryta iš anglų kalba pateiktų programėlių ekrano vaizdų. Eksperimentui atrinkta 15 atsitiktinių paveikslėlių, kuriuose dirbtinai įterpti lokalizacijos defektai, tokie kaip gramatinės klaidos, teksto perpildymas mygtukuose, žodžių nukirpimai ir kiti panašūs atvejai. Kiekvienas iš šių vaizdų bus analizuojamas taikant pasirinktus metodus.



22 pav. Eksperimentui naudota duomenų aibė

Visa ši duomenų aibė buvo atrinkta atsitiktinai iš Lietuvos ir tarptautinių programėlių ekrano nuotraukų. Šiose nuotraukose dirbtinai sukurti įvairūs lokalizacijos defektai, tokie kaip rašybos klaidos, neišversti ar nepilnai išversti tekstai, kitų kalbų žodžių įterpimai, naudotojo sąsajos deformacijos dėl teksto ilgio bei nukirptas tekstas.

Eksperimentui pasirinkta pagrindinė kalba – anglų, todėl kiekviena ekrano nuotrauka turėtų būti pilnai išversta į anglų kalbą ir tinkamai funkcionuoti.

4.5. OCR technologijos eksperimentinė analizė

Pirmasis analizės etapas buvo nuotraukų aibės apdorojimas naudojant dvi atvirojo kodo OCR (angl. Optical Character Recognition) bibliotekas – „Tesseract OCR“ ir „PaddleOCR“. Sistemos realizavimo ir demonstravimo metu buvo pasirinkta naudoti „PaddleOCR“, nes ši technologija pasižymėjo geresniu tikslumu ir paprastesniu įdiegimu, tačiau eksperimento metu buvo testuojamos abi bibliotekos, siekiant objektyviai įvertinti jų tikslumą ir veikimo kokybę.

Svarbu pažymėti, kad abiem OCR metodams prieš teksto atpažinimą buvo taikomas nuotraukų apdorojimas, siekiant pagerinti teksto matomumą ir nuskaitymo tikslumą. Buvo taikomi šie metodai:

- Pilkųjų atspalvių keitimas (angl. grayscaling) – pašalinamas spalvinis triukšmas, išryškinant tekstą.
- Dydžio keitimas – nuotraukos mastelis koreguojamas interpoliacijos metodu, siekiant optimalaus teksto dydžio.
- Pasukimo korekcijos – užtikrinama vienoda nuotraukų orientacija.
- Triukšmo mažinimas – taikomi filtravimo ir morfologiniai metodai, siekiant pašalinti artefaktus ir pagerinti teksto kokybę.

Eksperimento metu kiekviena nuotrauka buvo siunčiama analizei, o rezultatai grąžinami JSON formatu. Atsakyme pateikiamas apdorojimo laikas, aptiktų klaidų skaičius bei detalus klaidų sąrašas, kuriame nurodomos identifikuotos teksto dalys, galimos klaidos ir siūlomi pataisymai.

Kiekviena aptikta klaida buvo papildomai vertinama rankiniu būdu, siekiant nustatyti, ar ji yra pagrįsta. Remiantis šiuo vertinimu nustatytas teisingai ir klaidingai aptiktų klaidų skaičius bei atliktas abiejų OCR metodų palyginimas.

Kiekvieno analizuojamo paveikslėlio rezultatai buvo registruojami ir pateikiami duomenų lentelėje, kurią sudaro šie stulpeliai:

- Paveikslėlio Nr. – analizuojamo paveikslėlio numeris duomenų aibėje.
- Apdorojimo laikas (s) – laikas, per kurį paveikslėlis buvo pilnai apdorotas nuo proceso pradžios iki pabaigos.
- Rastų klaidų skaičius – bendras aptiktų klaidų skaičius, nepriklausomai nuo jų teisingumo.
- Teisingų klaidų skaičius – klaidų skaičius, kurios, atlikus vertinimą, buvo pripažintos realiais lokalizacijos defektais.
- Neteisingų klaidų skaičius – klaidingai identifikuotų defektų skaičius.

- Tikslumas – procentinė metrika, rodanti, kiek aptiktos klaidos atitinka realius defektus, leidžianti įvertinti algoritmo ar bibliotekos gebėjimą tiksliai analizuoti paveikslėlius.

Pirmiausia eksperimentas buvo atliktas naudojant „Tesseract OCR“ biblioteką. Buvo taikytos visos anksčiau aprašytos sąlygos – nuotraukų apdorojimas siekiant išryškinti tekstą, o po teksto išgavimo jis papildomai apdorotas, pašalinant įvairius artefaktus.

Tačiau gauti rezultatai (žr. 4 lentelę) parodė, kad šio metodo veikimo kokybė yra gana žema. Vidutiniškai teisingai aptiktų defektų dalis siekia apie 21,71 %, o tai rodo nepakankamą tikslumą. Tyrimo metu pastebėta, kad „Tesseract OCR“ sunkiai tiksliai išgauna tekstą iš programėlių ekrano vaizdų, todėl išgautas tekstas dažnai neatitinka originalo.

Taip pat nustatyta, kad paprastesnėse programėlėse, kuriose yra mažiau grafinių ir spalvinių elementų, teksto atpažinimas yra tikslesnis, o lokalizacijos defektų aptikimas – efektyvesnis.

4 lentelė. Tesseract OCR eksperimentas

Paveikslėlio Nr.	Apdorojimo laikas (s)	Rasta klaidų skaičius	Teisingų klaidų skaičius	Neteisingų klaidų skaičius	Tikslumas (proc.)
1 pav.	76,86	14	2	12	14
2 pav.	64,55	20	4	16	20
3 pav.	65,21	3	1	2	33
4 pav.	93,35	8	1	7	12,5
5 pav.	80,44	0	0	0	-
6 pav.	74,38	23	2	21	8.6
7 pav.	70,05	8	2	6	25
8 pav.	71,15	11	2	9	18,18
9 pav.	62,61	5	1	4	20
10 pav.	96,69	7	0	7	0
11 pav.	62,94	11	1	10	9
12 pav.	59,36	4	3	1	75
13 pav.	64,94	3	1	2	33
14 pav.	78,04	7	1	6	14
15 pav.	64,05	0	0	0	-

Atlikus eksperimentą tokiomis pačiomis sąlygomis kaip ir su „Tesseract OCR“, pastebėti gerokai geresni rezultatai naudojant „PaddleOCR (žr. 5 lentelė). Vidutinis teisingai aptiktų defektų procentas siekė apie 52,5 %, o tai yra reikšmingai geresnis rezultatas, palyginti su „Tesseract OCR“.

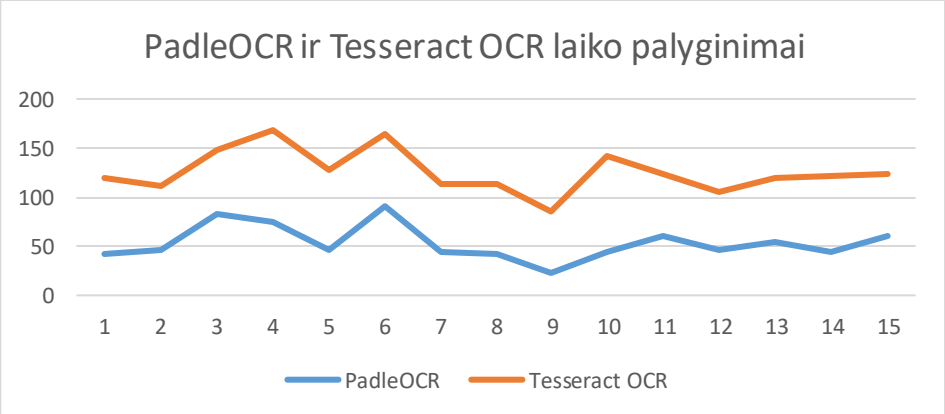
Taip pat nustatyta, kad iš nuotraukų išgautas tekstas yra gerokai kokybiškesnis – su mažesniu kiekiu artefaktų, kas iš esmės pagerina viso OCR proceso efektyvumą ir leidžia tiksliau identifikuoti lokalizacijos defektus.

5 lentelė. Paddle OCR eksperimentas

Paveikslėlio Nr.	Apdorojimo laikas (s)	Rasta klaidų skaičius	Teisingų klaidų skaičius	Neteisingų klaidų skaičius	Tikslumas (proc.)
1 pav.	41,78	13	4	9	30
2 pav.	46,91	7	5	2	71
3 pav.	82,6	3	3	0	100
4 pav.	74,91	8	6	2	75
5 pav.	46,98	2	1	1	50
6 pav.	90,76	13	6	7	46
7 pav.	43,45	9	4	5	44
8 pav.	42,37	5	2	3	40
9 pav.	22,74	1	1	0	100
10 pav.	44,47	2	1	1	50
11 pav.	61,16	3	1	2	33
12 pav.	45,88	7	5	2	71
13 pav.	55,22	0	0	0	-
14 pav.	44,35	4	1	3	25
15 pav.	59,53	0	0	1	0

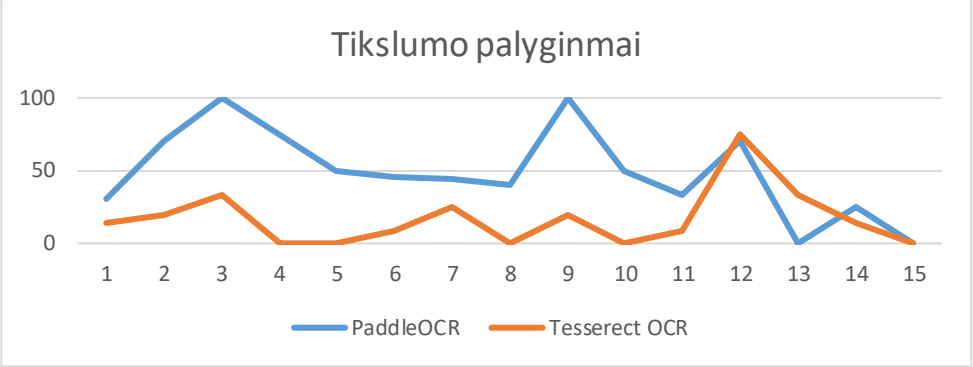
Galiausiai, atlikus abu tyrimus, buvo galima palyginti jų veikimo laiką (žr. 23 pav.). Rezultatai rodo, kad visais eksperimento atvejais greičiau veikė „PaddleOCR“. Vidutinis vienos nuotraukos apdorojimo laikas siekė apie 53 sekundes, tuo tarpu „Tesseract OCR“ vidutiniškai užtruko apie 73 sekundes.

Šis skirtumas paaiškinamas naudojamomis technologijomis. „PaddleOCR“ remiasi neuroniniais tinklais ir efektyviai išnaudoja grafinės plokštės resursus, tuo tarpu „Tesseract OCR“ yra labiau taisyklėmis ir šablonais pagrįstas sprendimas, kuris daugiausia naudoja centrinio procesoriaus (angl. CPU) išteklius. Dėl to „PaddleOCR“ veikia greičiau ir efektyviau esant toms pačioms sąlygoms.



23 pav. Paddle OCR ir Tesseract OCR laiko palyginimai

Toliau analizuojant tikslumo kitimą kiekvienam pateiktam paveikslėliui matyti (žr. 24 pav.), kad geriausius rezultatus daugeliu atvejų pasiekė „PaddleOCR“ biblioteka. Ji dažniausiai pateikė tiksliausius rezultatus, pasižyminčius mažiausia paklaida, lyginant su „Tesseract OCR“, kuris tik pavieniais atvejais sugebėjo pasiekti ar viršyti „PaddleOCR“ tikslumą.



24 pav. Tikslumo palyginimai

Atlikus šių dviejų bibliotekų palyginamąją analizę, nustatyta, kad „PaddleOCR“ veikia gerokai efektyviau nei „Tesseract OCR“ tiek tikslumo, tiek greitaveikos aspektais. Eksperimento rezultatai parodė, kad „PaddleOCR“ yra tinkamesnis sprendimas nagrinėjamai problemai spręsti.

Vis dėlto, remiantis eksperimento metu pastebėtais trūkumais, galima teigti, kad OCR technologijos taikymas programėlių lokalizacijos defektų aptikimui yra galimas, tačiau nėra optimalus sprendimas. Nuskaitytas tekstas dažnai turi artefaktų, o pati technologija geba tik išgauti tekstą, bet ne visapusiškai suprasti jo kontekstą ar išdėstymą. Eksperimento metu pastebėta, kad tekstas dažnai sujungiamas į vientisą bloką, nors realiose programėlėse jis yra išskaidytas į skirtingus sąsajos elementus, pavyzdžiui, mygtukus ar laukus. Dėl to atsiranda klaidų, susijusių su skyryba, sakinių struktūra ar raidžių naudojimu, kurios iš tikrųjų kyla ne dėl neteisingo vertimo, o dėl netinkamo teksto interpretavimo.

Taip pat nustatyta, kad spalvos ir grafiniai elementai gali sukelti papildomų artefaktų, nes OCR technologija juos kartais interpretuoja kaip tekstą. Siekiant pagerinti rezultatus, reikėtų ne tik taikyti išankstinį vaizdų apdorojimą, bet ir struktūrizuoti vaizdą – identifikuoti, kuris tekstas priklauso konkrečiam sąsajos elementui, bei ignoruoti nereikšmingus grafinius elementus.

OCR technologijos pritaikymas lokalizacijos defektų aptikimui buvo sėkmingai išbandytas, o galutinėje sistemos versijoje pasirinkta naudoti „PaddleOCR“, kuris pasižymi geresniu tikslumu ir greičiu. Tačiau esama realizacija dar nėra pakankamai patikima, todėl būtina toliau tobulinti metodus, įvertinant su tekstu susijusius artefaktus ir vaizdo struktūros ypatumus, siekiant užtikrinti aukštesnę rezultatų kokybę.

4.6. Objektų aptikimo eksperimentinė analizė

Kitas eksperimento etapas buvo pasirinktos nuotraukų aibės analizė naudojant apmokytą objektų aptikimo modelį, siekiant nustatyti, ar jis geba tiksliai aptikti vizualinius lokalizacijos defektus. Šiam tikslui buvo apmokytas objektų aptikimo modelis, paremtas „YOLOv8“ architektūra.

Defektų aptikimo procesas vykdomas tokia seka:

- Įkeliama nuotraukų aibė – analizuojami programėlių ekrano vaizdai, kuriuose siekiama aptikti lokalizacijos defektus.
- Pradedamas aptikimo procesas naudojant apmokytą modelį.
- Modelis identifikuoja defektus, pateikdamas jų ribas (angl. bounding boxes) ir pasitikėjimo rodiklį.
- Aptikti objektai filtruojami pagal nustatytą pasitikėjimo slenkstį – šiame eksperimente naudota 0,6 reikšmė.
- Sugeneruojama defektų ataskaita JSON formatu, įtraukiant tik tuos aptikimus, kurių pasitikėjimo rodiklis yra ≥ 0,6.
- Išsaugomos nuotraukos su pažymėtais defektais tolimesnei analizei.
- Visi rezultatai grąžinami naudotojui, kuris gali juos perduoti kitiems suinteresuotiems asmenims.

Modelis buvo apmokytas naudojant 100 paveikslėlių duomenų aibę, treniruojant 50 epochų. Ši duomenų aibė buvo sukurta sintetiniu būdu, nes viešai prieinamos duomenų aibės nebuvo. Todėl įvairūs lokalizacijos defektai buvo rankiniu būdu kuriami ekrano nuotraukose, kurios vėliau buvo naudojamos apmokymui, o klasės buvo priskiriamos rankiniu būdu.

Taip pat buvo generuojami atskiri komponentai, pasižymintys įvairiais dizaino ir stiliaus sprendimais bei specifiniais defektais, pagal kuriuos buvo apibrėžtos kai kurios defektų klasės. Taigi apie 50 nuotraukų sudarė realių programėlių ekrano nuotraukos, kuriose rankiniu būdu buvo sukurti lokalizacijos defektai, o kitos 50 nuotraukų buvo įvairių defektų klasių objektai, sugeneruoti su įterptais defektais.

Apmokymo metu išskirtos trys pagrindinės lokalizacijos defektų klasės:

- teksto išėjimas už mygtuko ribų (angl. button text overflow);
- įvedimo lauko teksto nukirpimas (angl. input text truncation);
- teksto persidengimas (angl. text overlapping).

Šios klasės pasirinktos todėl, kad jos yra vienos dažniausių lokalizacijos defektų, atsirandančių įvedus naujas kalbas į sistemas. Vis dėlto šią klasifikaciją galima plėsti, siekiant padidinti modelio universalumą.

Kiekvieno eksperimento metu buvo renkama informacija, gauta po modelio analizės. Skaitinės reikšmės registruojamos duomenų lentelėje ir naudojamos tolesniems skaičiavimams. Lentelę sudaro šie stulpeliai:

- Pav. – paveikslėlio numeris duomenų aibėje;
- Laikas (s) – laikas nuo proceso inicijavimo iki jo pabaigos;
- Tikras skaičius – realus defektų skaičius, nustatytas žmogaus;
- Aptiktas skaičius – modelio aptiktų defektų skaičius;
- TP (angl. True Positives) – teisingai aptikti defektai;
- FP (angl. False Positives) – klaidingai aptikti defektai;
- FN (angl. False Negatives) – praleisti defektai;
- Tikslumas – modelio tikslumo rodiklis intervale nuo 0 iki 1;
- Atkūrimo įvertis (angl. recall) – rodiklis, nusakantis, kiek realių defektų buvo aptikta;
- F1 – bendras modelio kokybės rodiklis, apjungiantis tikslumą ir atkūrimo įvertį.

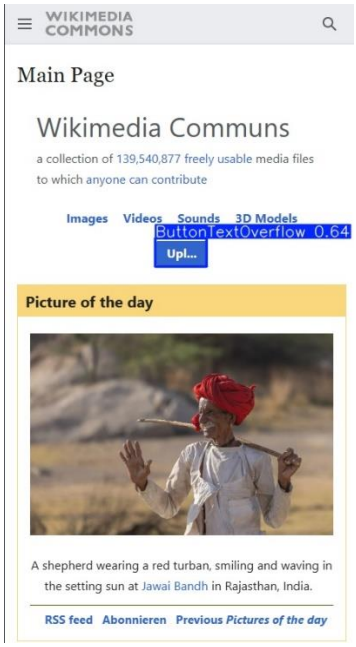
Atlikus eksperimentą su objektų aptikimo modeliu (žr. 6 lentelė), nustatyta, kad vidutinis tikslumas siekia apie 0,31, atkūrimo įvertis – apie 0,36, o vidutinis apdorojimo laikas – apie 0,29 sekundės. Rezultatai rodo, kad modelis geba aptikti aiškiai matomus defektus ir dažniausiai juos teisingai klasifikuoja. Tačiau taip pat pastebėta nemažai klaidingų aptikimų ir praleistų defektų, kas leidžia daryti išvadą, jog modeliui reikalinga didesnė treniravimo duomenų aibė bei ilgesnis apmokymo procesas, siekiant pagerinti jo veikimo kokybę.

6 lentelė. Objektų aptikimo modelio rezultatai

Pav.	Laikas (s)	Patikimumo kof.	Tikras skaičius	Aptiktas skaičius	TP	FP	FN	Tikslumas	Atkūrimo įvertis	F1
1	0,170	—	0	0	0	0	0	-	-	-
2	0,491	1,000	1	0	0	0	1	0,00	0,00	0,00
3	0,180	—	0	0	0	0	0	-	-	-
4	0,181	0,6355	1	1	1	0	0	1,00	1,00	1,00
5	0,165	0,7045	1	1	1	0	0	1,00	1,00	1,00
6	0,220	0,6338	1	1	1	0	0	1,00	1,00	1,00
7	0,394	0,00	1	0	0	0	1	0	0	0
8	0,142	—	0	0	0	0	0	-	-	-
9	0,192	0,6351	1	2	1	1	0	0,50	1,00	0,67
10	0,444	0,7975	1	1	0	1	1	0,00	0,00	0,00
11	0,174	—	0	0	0	0	0	-	-	-
12	0,402	1,000	1	0	0	0	1	0,00	0,00	0,00
13	0,183	1,000	1	0	0	0	1	0,00	0,00	0,00

14	0,271	1,000	1	0	0	0	1	0,00	0,00	0,00
15	0,527	0,8609	1	1	0	1	1	0,00	0,00	0,00

Matyti, kad jei modelis aptinka defektą, anotacijos dažniausiai yra korektiškai uždedamos tiesiai ant paveikslėlio (žr. 25 pav.). Siekiant pagerinti metodo veikimą, būtina išplėsti treniravimo duomenų aibę, kad modelis galėtų pasiekti didesnį tikslumą ir aptikti defektus su aukštesniu pasitikėjimo lygiu.



25 pav. Objektų aptikimo anotacijos pavyzdys

Eksperimento metu pastebėta, kad objektų aptikimo metodas pasižymi labai dideliu greičiu – kiekvienas paveikslėlis apdorojamas itin greitai. Didžiausios laiko sąnaudos tenka modelio apmokymui ir tinkamų parametrų parinkimui. Taip pat nustatyta, kad objektų aptikimo metodas yra pagrįstas lokalizacijos defektų nustatymui, nes pasižymi tiek greitu veikimu, tiek nedidelėmis eksploatavimo sąnaudomis.

Svarbiausias aspektas – kokybiškas modelio apmokymas. Siekiant geresnių rezultatų, būtina išplėsti treniravimo duomenų aibę ir padidinti lokalizacijos defektų klasių skaičių, kad modelis galėtų tiksliau klasifikuoti skirtingus defektų tipus. Taip pat šis metodas turi papildomą pranašumą – leidžia vizualiai identifikuoti defektų vietas, kas palengvina jų taisymą kuriamose sistemose.

Eksperimento metu pasiektas tikslumas nebuvo aukštas dėl ribotos duomenų aibės, didinant treniravimo duomenų kiekį, optimizuojant parametrus ir plečiant klasių skaičių, būtų galima reikšmingai pagerinti lokalizacijos defektų aptikimo kokybę. Be to, analizuojamas metodas yra greitas ir ekonomiškas, todėl turi didelį potencialą būti pritaikytas praktikoje, tinkamai parengus modelį.

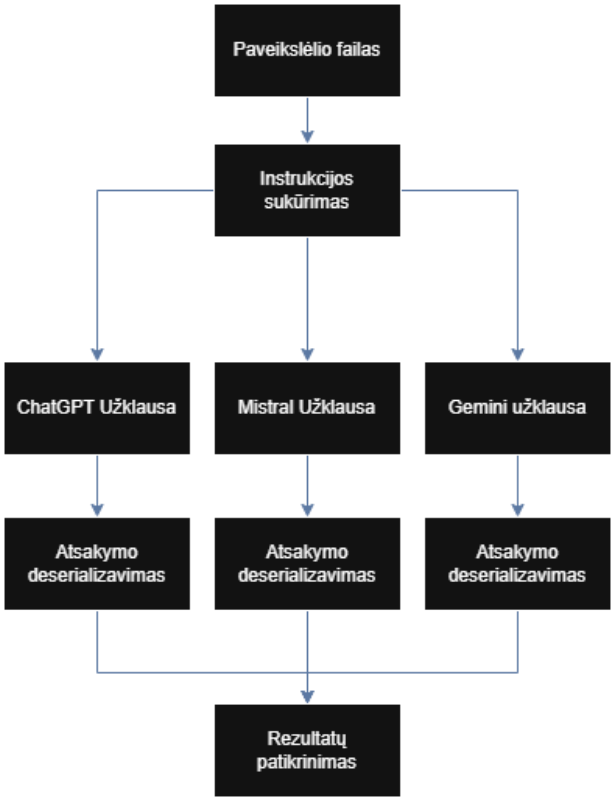
4.7. DI modelių aptikimo eksperimentinė analizė

Paskutinis eksperimento etapas skirtas didžiųjų dirbtinio intelekto pokalbių modelių išbandymui. Tyrimo metu analizuojami „Gemini 3 Flash“, „Mistral-small“ ir „ChatGPT 4o mini“ modeliai, gebantys apdoroti paveikslėlius ir paversti juos kontekstine informacija. Remiantis šia informacija, modeliai turi identifikuoti lokalizacijos defektus, juos apibūdinti ir pateikti siūlomus pataisymus. Bus lyginami jų rezultatai, siekiant nustatyti, kuris modelis tiksliausiai supranta vaizdinę informaciją, kokybiškai identifikuoja defektus ir pateikia tinkamus sprendimus.

Modeliai pasirinkti atsižvelgiant į jų sudėtingumą, galimybes ir sąnaudas. „Gemini 3 Flash“ laikomas vienu iš pažangesnių modelių, tačiau yra brangesnis. „Mistral-small“ yra mažesnis, atvirojo kodo modelis, kurį galima naudoti lokaliai, todėl įdomu įvertinti jo efektyvumą. „ChatGPT 4o mini“ yra optimizuotas mažesnės apimties modelis, leidžiantis įvertinti, ar optimizacija gali kompensuoti mažesnius modelio pajėgumus.

Eksperimento eiga (žr. 26 pav.) yra tokia: pirmiausia pasirenkamas paveikslėlis iš duomenų aibės, kuris perduodamas sistemai. Prieš siunčiant jį analizei, suformuojama instrukcija, susidedanti iš statinės ir dinaminės dalių, pavyzdžiui, nurodančių kalbą ar analizės kontekstą. Tuomet lygiagrečiai paleidžiami visi trys modeliai, kuriems perduodamas paveikslėlis ir instrukcija. Modeliai analizuoja vaizdą, identifikuoja lokalizacijos defektus ir grąžina rezultatus.

Gauti atsakymai validuojami pagal formatą, papildomi reikalingais metaduomenimis ir perduodami tolimesnei analizei. Kiekvienas aptiktas defektas vertinamas rankiniu būdu, nustatant, ar jis yra realus. Svarbu pažymėti, kad jei dirbtinio intelekto modelis pateikia pagrįstą defektą, kuris nebuvo iš anksto pažymėtas duomenų aibėje, jis vis tiek laikomas teisingu aptikimu. Tai leidžia įvertinti modelių gebėjimą aptikti defektus, kurių žmogus galėjo nepastebėti, ir suteikia papildomą metriką jų efektyvumui įvertinti.



26 pav. DI modelių eksperimento eiga

Atlikus eksperimentus su pasirinktais modeliais, buvo įvertinti jų gebėjimai aptikti lokalizacijos defektus ir nustatyta jų praktinė nauda. Eksperimento rezultatų lentelė sudaryta iš šių elementų:

- Nr. – paveikslėlio numeris iš eksperimentinės duomenų aibės;
- Apdorojimo laikas – laikas, per kurį modelis išanalizavo paveikslėlį ir pateikė rezultatą;
- Žetonai (angl. tokens) – bendras įvesties ir išvesties žetonų skaičius;
- Rankiniu būdu aptiktos klaidos – defektai, nustatyti žmogaus prieš eksperimentą;
- Teisingai įvardintos klaidos – modelio teisingai identifikuotų defektų skaičius. Į šį skaičių įtraukiamos tiek rankiniu būdu nustatytos klaidos, tiek naujai aptiktos. Skliausteliuose nurodoma, kiek iš jų sutampa su rankiniu būdu nustatytais defektais;
- Įvardintos klaidos – visos modelio pateiktos klaidos, nepriklausomai nuo jų teisingumo, kartu su siūlomais taisymo būdais.

Pirmiausia buvo analizuojamas „Gemini 3 Flash“ modelis. Nors tikslus šio modelio parametrų skaičius nėra viešai žinomas, galima teigti, kad tai yra vienas iš sudėtingesnių ir didelės apimties modelių. Remiantis gautais rezultatais (žr. 7 lentelę), modelis veikia pakankamai gerai – pasižymi aukštu tikslumu ir geba aptikti defektus, kurie nebuvo nustatyti

rankiniu būdu. Modelio pateikiama klaidų klasifikacija yra tiksli, o siūlomi taisymo variantai – kokybiški ir praktiškai pritaikomi.

Vis dėlto pastebėta, kad modelis kartais neteisingai interpretuoja datas. Kadangi modelis neturi prieigos prie realaus laiko informacijos, jis gali laikyti tam tikras datas neteisingomis (pvz., interpretuoti jas kaip ateities datas), nors iš tikrųjų jos yra korektiškos. Tai rodo, kad kai kurie kontekstiniai aspektai, tokie kaip dabartinė data, gali turėti įtakos rezultatų tikslumui.

7 lentelė. Gemini 3 Flash modelio rezultatai

Nr.	**Apdorojimo laikas (s)**	**Žetonai (viso)**	**Rankiniu aptiktos klaidos**	**Aptikti defektai (modelis)**	**Teisingai įvardintos klaidos (klaidos aptiktos rankiniu būdu)**	**Įvardintos klaidos**
1 pav.	10,18	3277	4	7	6 (4)	Vertimai, gramatika, datos formatas, UI perpildymas
2 pav.	7,75	2975	4	4	4 (3)	Vertimai, UI persidengimas, teksto nukirpimas, gramatika
3 pav.	7,7	3086	3	5	3 (3)	Vertimai (raktai), valiutos formatas, datos tikslumas, UI išdėstymas
4 pav.	8,35	3015	2	5	3 (2)	Vertimai, įkoduoti tekstas, teksto nukirpimas, terminijos nuoseklumas
5 pav.	7,18	2898	2	4	4 (2)	Gramatika teksto nukirpimas, vertimai, terminijos nuoseklumas
6 pav.	13,91	3346	4	7	4 (3)	Vertimai (mišri kalba), datos formatas, tiesiogiai įkoduotas tekstas, UI perpildymas
7 pav.	8,47	3049	2	5	4 (2)	Teksto nukirpimas, vertimai (reCAPTCHA), gramatika, terminijos nuoseklumas
8 pav.	9,49	3249	4	7	3 (4)	Vertimai, datos formatas, terminijos

						nuoseklumas, teksto nukirpimas
9 pav.	8,51	2922	1	4	3 (1)	UI persidengimas, simbolių/teksto klaidos, išdėstymas, terminijos nuoseklumas
10 pav.	9,53	3062	2	5	2 (2)	Teksto nukirpimas, gramatika, mygtuko dydis, terminijos nuoseklumas
11 pav.	8,93	3158	2	5	3 (2)	Tiesiogiai įkoduotas tekstas, vertimai (mišri kalba), datos formatas, teksto nukirpimas
12 pav.	6,8	2820	2	3	2 (1)	Vertimai, teksto nukirpimas, terminijos nuoseklumas
13 pav.	8,09	2879	2	4	2 (2)	Gramatika, teksto nukirpimas, UI pritaikymas
14 pav.	7,8	2941	2	4	2 (2)	Teksto nukirpimas, gramatika, mygtuko dydis, UI išdėstymas
15 pav.	9,06	3030	5	5	5 (3)	Vertimai (visas UI), teksto nukirpimas, gramatika, terminijos nuoseklumas

Kitas testuotas modelis buvo „Mistral-small 3.2“, sukurtas „Mistralai“, turintis apie 24 milijardus parametrų. Tai vienas iš mažesnių modelių, naudotų šiame eksperimente, kurio privalumas – galimybė veikti lokaliai, naudojant asmeninius ar nedidelius serverius, be išorinių paslaugų.

Tačiau, remiantis gautais rezultatais (žr. 8 lentelę), nors modelis geba analizuoti vaizdinę informaciją, jo tikslumas yra žymiai mažesnis, palyginti su kitais tirtais modeliais. Pastebėta, kad modelis generuoja nemažai klaidingų ar netikslių defektų aptikimų. Be to, jis dažnai interpretuoja stilistinius programėlės aspektus kaip lokalizacijos defektus, nors tokie vertinimai daugeliu atvejų nėra pagrįsti ir yra atmetami kaip netinkami.

8 lentelė. mistralai/mistral-small-3.2-24b-instruct modelio rezultatai

Nr.	**Apdorojimo laikas (s)**	**Žetonai (viso)**	**Rankiniu aptiktos klaidos**	**Aptikti defektai (modelis)**	**Teisingai įvardintos klaidos (klaidos aptiktos rankiniu būdu)**	**Įvardintos klaidos**

1 pav.	14,51	2825	4	10	3 (3)	Vertimai, datos formatas, terminija, teksto nukirpimas
2 pav.	11,0	2090	4	3	1 (1)	Teksto nukirpimas, UI išdėstymas, tarpai
3 pav.	8,86	2109	3	3	0 (0)	Teksto nukirpimas, UI išdėstymas, tarpai
4 pav.	21,95	2311	2	5	1 (1)	Teksto nukirpimas, UI išdėstymas, vertimai
5 pav.	6,01	2146	2	3	2 (1)	Rašyba, teksto nukirpimas, UI pritaikymas
6 pav.	12,6	2634	4	10	2 (2)	Vertimai, datos formatas, teksto nukirpimas, UI išdėstymas
7 pav.	8,31	2307	2	6	2 (2)	Vertimai, teksto nukirpimas, UI išdėstymas
8 pav.	16,34	2787	4	10	3(3)	Vertimai, datos formatas, teksto nukirpimas, UI išdėstymas
9 pav.	13,5	2043	1	2	2(1)	UI išdėstymas, teksto perpildymas
10 pav.	12,31	2134	2	3	1 (1)	UI išdėstymas, teksto perpildymas
11 pav.	10,88	2080	2	3	0	UI išdėstymas, teksto perpildymas
12 pav.	7,53	2066	2	2	2 (2)	Vertimai, teksto perpildymas
13 pav.	5,71	2115	2	3	0	UI išdėstymas, teksto perpildymas
14 pav.	14,05	2170	2	3	0	Didinti teksto konteinerius, tvarkyti išlygiavimą
15 pav.	14,77	4374	5	10	2 (2)	Ištaisyti vertimus, pašalinti teksto nukirpimą ir suvienodinti elementų išlygiavimą bei tarpus.

Kitas analizuotas modelis – „GPT-4o mini“, sukurtas „OpenAI“. Nors tikslus šio modelio parametrų skaičius nėra viešai skelbiamas, manoma, kad jis siekia apie 8 milijardus, todėl tai yra vienas ekonomiškiausių modelių šiame tyrime.

Remiantis rezultatais (žr. 9 lentelę), modelis pasirodė pakankamai gerai, ypač atsižvelgiant į jo nedidelį dydį. Daugeliu atvejų jis gebėjo aptikti klaidas, kurios buvo identifikuotos rankiniu

būdu. Tai rodo, kad net ir mažesnės apimties modeliai gali būti efektyvūs sprendžiant lokalizacijos defektų aptikimo užduotis.

Taip pat pastebėta, kad šis modelis pasižymi dideliu žetonų (angl. tokens) sunaudojimu – dažnai viršijančiu 15 tūkst., nors visiems modeliams buvo taikomos vienodos sąlygos. Tai gali turėti įtakos bendram sprendimo efektyvumui ir sąnaudoms.

9 lentelė. gpt-4o-mini modelio rezultatai

Nr.	Apdorojimo laikas (s)	Žetonai (viso)	Rankiniu aptiktos klaidos	Aptikti defektai (modelis)	Teisingai įvardintos klaidos (klaidos aptiktos rankiniu būdu)	Įvardintos klaidos
1 pav.	12,06	15919	4	6	4 (3)	Vertimo klaidos; Rašybos klaidos; Datos formatas
2 pav.	10,26	15831	4	5	2 (2)	Neteisinga kalba; Trūksta vertimo; Mygtuko tekstas
3 pav.	8,24	15825	3	5	2 (2)	Gramatikos klaida; Teksto aiškumas; Lauko pavadinimas
4 pav.	14,59	15855	2	5	0	Teksto nukirpimas; Mygtuko dydis; Tarpų tvarka
5 pav.	12,09	15838	2	5	1 (1)	Teksto perpildymas; Vertimo aiškumas; Konteksto trūkumas
6 pav.	9,58	15884	4	6	2 (2)	Trūksta vertimo; Datos formatas; Mygtuko dydis
7 pav.	13,49	15858	2	5	2 (2)	Neteisinga kalba; Teksto ilgis; Mygtuko dydis
8 pav.	15,26	15838	4	5	3 (3)	Trūksta vertimo; Datos formatas; Vertimo kokybė
9 pav.	8,77	15567	1	2	1 (1)	Teksto nukirpimas; Teksto formuluotė
10 pav.	7,76	15669	2	3	1 (1)	Teksto ilgis; Lygiavimo klaida; Teksto tonas
11 pav.	12,84	15770	2	4	2 (2)	Neteisingas tekstas; Gramatikos klaida; Tarpų klaida
12 pav.	11,71	15826	2	5	2 (2)	Trūksta vertimo; Teksto nukirpimas; Lygiavimo klaida

13 pav.	4,98	15319	2	0	0	Nėra klaidų
14 pav.	5,26	15319	2	0	0	Nėra klaidų
15 pav.	12,94	38473	5	5	3 (3)	Neteisinga kalba; Trūksta vertimo; Teksto ilgis

Apibendrinus visus rezultatus (žr. 10 lentelę), nustatyta, kad geriausiai pasirodė „Gemini 3 Flash“ modelis, pasiekęs 67,57 % defektų aptikimo tikslumą. Antroje vietoje – „GPT-4o mini“, kurio tikslumas siekė 40,98 %, o prasčiausiai pasirodė „Mistral-small 3.2“ modelis, pasiekęs 27,63 % tikslumą.

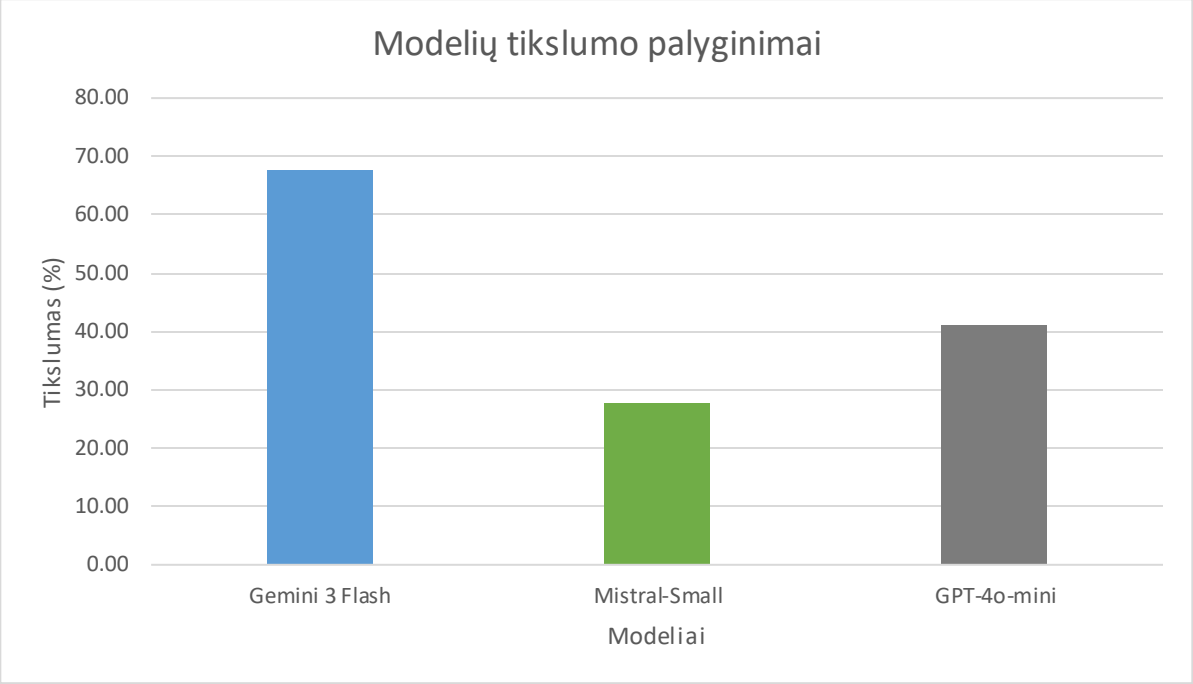
10 lentelė. Apibendrinti modelių rezultatai

Modelis	**Vid. Laikas (s)**	**Viso Žetonų**	**Žetonų skaičius vidutiniškai**	**Viso Modelio Defektų**	**Teisingai Atpažinta (žmogaus + nauji)**	**Atitiko Žmogaus Sąrašą**	**Naujai Aptikta (Pranoko žmogų)**	**Tikslumas (%)**
Gemini 3 Flash	8,78	45707	3047	74	50	36	14	67,57
Mistral-Small	11,89	36191	2413	76	21	19	2	27,63
GPT-4o-mini	10,66	258791	17253	61	25	24	1	40,98

Tai galima aiškiai matyti ir vizualiai (žr. 27 pav.) – geriausiai pasirodė „Gemini 3 Flash“, po jo – „GPT-4o mini“, o prasčiausius rezultatus demonstravo „Mistral-small 3.2“. Galima daryti prielaidą, kad didesnis modelio parametrų skaičius dažnai lemia geresnius rezultatus, nes modelis turi daugiau gebėjimų apdoroti sudėtingą informaciją.

Vis dėlto svarbu pabrėžti, kad ne tik modelio dydis, bet ir jo optimizavimas bei mokymo duomenys turi didelę įtaką rezultatams. Tai patvirtina „GPT-4o mini“ atvejis – nors modelis yra mažesnis, jis pasiekė geresnius rezultatus nei „Mistral-small 3.2“.

Taip pat reikia įvertinti ir sąnaudų aspektą – „Mistral-small“ yra pigesnis modelis, todėl tam tikrais atvejais gali būti pasirinktas kaip ekonomiškesnis sprendimas, nepaisant mažesnio tikslumo.

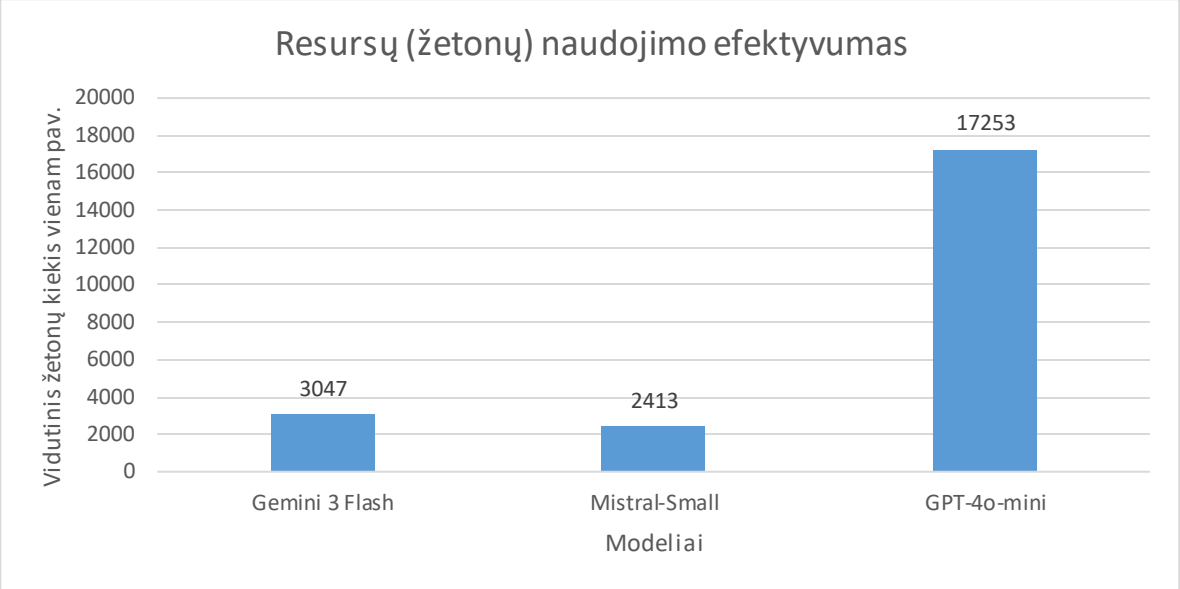


27 pav. Modelių tikslumo palyginimai

Toliau iš diagramos (žr. 28 pav.) matyti vidutinis žetonų sunaudojimas vienam paveikslėliui. Kadangi visi paveikslėliai ir jų parametrai buvo vienodi, eksperimentinės sąlygos visiems modeliams išliko tokios pačios.

Rezultatai rodo, kad mažiausiai žetonų sunaudojo „Mistral-small 3.2“ – vidutiniškai apie 2413 žetonų vienam paveikslėliui. Toliau seka „Gemini 3 Flash“, kurio vidutinis sunaudojimas siekė apie 3047 žetonus. Didžiausias žetonų kiekis buvo stebimas „GPT-4o mini“ modelyje – vidutiniškai apie 17 tūkst. žetonų vienam paveikslėliui, kas yra ženkliai daugiau nei kitų modelių atveju.

Remiantis šiais rezultatais galima teigti, kad efektyviausias pagal resursų panaudojimą yra „Mistral-small 3.2“, o mažiausiai efektyvus – „GPT-4o mini“, kuris, nors ir pasižymi geresniu tikslumu nei „Mistral-small“, reikalauja gerokai didesnių skaičiavimo resursų.



28 pav. Resursų (žetonų) naudojimo efektyvumas

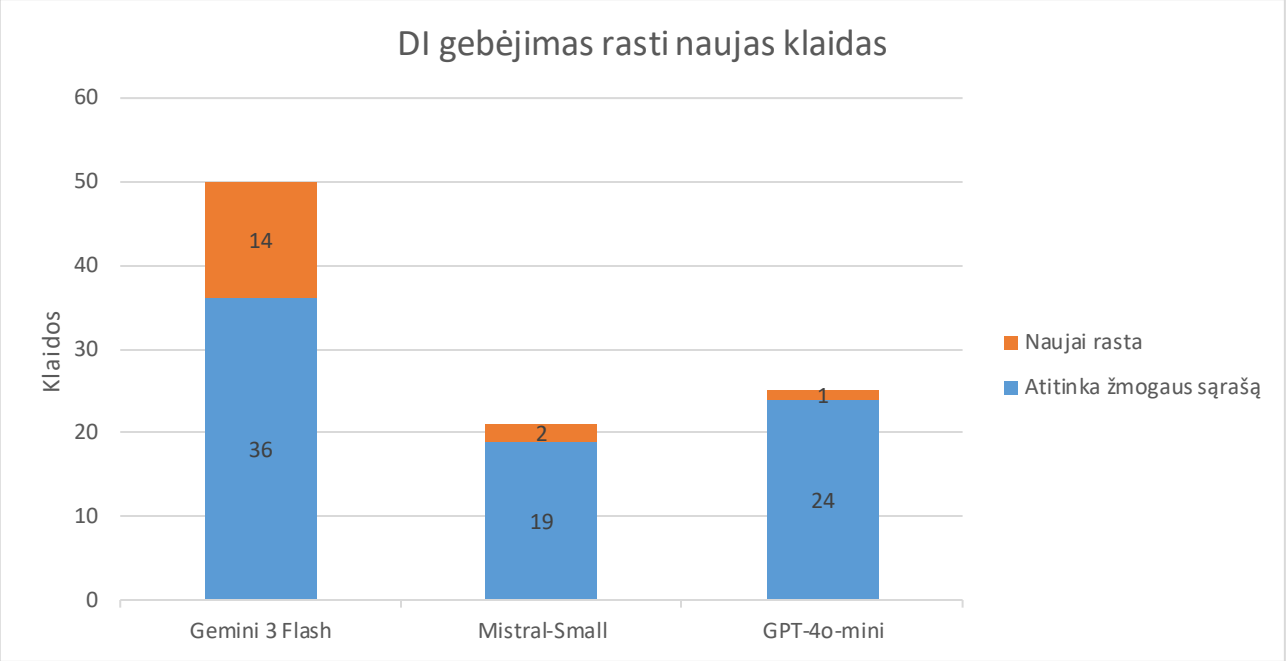
Žetonų sunaudojimas ne visada tiesiogiai atspindi modelio naudojimo kainą. Atlikus eksperimentą, buvo apskaičiuota bendra 15 paveikslėlių analizės kaina (žr. 11 lentelę). Rezultatai parodė, kad brangiausias modelis buvo „Gemini 3 Flash“ – visų užklausų analizė kainavo apie 0,07 JAV dolerio. Toliau seka „GPT-4o mini“ modelis, kuris, nepaisant didesnio žetonų sunaudojimo, kainavo apie 0,04 JAV dolerio už visas 15 užklausų. Tai rodo, kad modelio kainodara nepriklauso vien tik nuo žetonų kiekio, bet ir nuo taikomų tarifų. Mažiausias sąnaudas turėjo „Mistral-small 3.2“ modelis – apie 0,02 JAV dolerio už visą analizę. Be to, šio modelio atveju svarbu pažymėti, kad jis gali būti vykdomas lokaliai, todėl naudojant nuosavą infrastruktūrą analizė gali būti atliekama be papildomų išlaidų už užklausas.

11 lentelė. Modelių kaina

Modelio pavadinimas	**Suma ($)**
Gemini 3 Flash	0,07
GPT-4o-mini	0,04
Mistral Small 3.2 24B	0,02

Vienas įdomiausių eksperimento rezultatų – modelių gebėjimas aptikti klaidas, kurios nebuvo identifikuotos rankiniu būdu. Iš diagramos (žr. 29 pav.) matyti, kad daugiausia tokių papildomų defektų aptiko „Gemini 3 Flash“. Toliau seka „Mistral-small 3.2“ ir „GPT-4o mini“, tačiau šių modelių gebėjimas rasti naujas, anksčiau neidentifikuotas klaidas buvo gerokai ribotesnis – dažniausiai jie aptikdavo tas pačias klaidas kaip ir žmogus.

Tai rodo, kad pažangesni modeliai, tokie kaip „Gemini 3 Flash“, gali ne tik atkartoti žmogaus atliekamą analizę, bet ir ją papildyti, identifikuodami subtilius lokalizacijos ar vizualinius niuansus, kurie gali būti nepastebėti rankinio testavimo metu. Šis rezultatas leidžia teigti, kad dirbtinio intelekto modeliai tam tikrais atvejais gali pranokti žmogų lokalizacijos defektų aptikimo srityje, ypač kai reikalingas platesnis konteksto ir kalbinių niuansų supratimas.



29 pav. DI gebėjimas rasti naujas klaidas

Po daugiafunkcių dirbtinio intelekto modelių eksperimento išskirti keli svarbūs pastebėjimai. Pirmiausia, visi modeliai susidūrė su sunkumais vertinant programėlių dizaino aspektus, pavyzdžiui, neteisingai interpretuodavo lygiavimą, teksto matomumą ar kitus vizualinius elementus, kurie iš tikrųjų nebuvo susiję su lokalizacijos defektais. Tokie atvejai buvo vertinami kaip klaidingi aptikimai. Taip pat pastebėta, kad modeliai sunkiai susidoroja su smulkiais stilistiniais niuansais ir dažnai generuoja netikslius arba klaidingus atsakymus.

Be to, nustatyta, kad instrukcijų formulavimas turi didelę įtaką rezultatų kokybei. Siekiant tikslesnių rezultatų, būtina aiškiai ir griežtai apibrėžti, kokie defektai turi būti aptinkami ir kaip turi būti atliekama analizė. Tikslus galimų defektų apibrėžimas padeda sumažinti klaidingų identifikacijų skaičių. Taip pat svarbus platesnis sistemos kontekstas – kuo daugiau informacijos apie programėlę, jos paskirtį ir naudojimo aplinką pateikiama, tuo tiksliau modeliai gali identifikuoti lokalizacijos defektus, atsižvelgdami ir į kultūrinius aspektus.

Apibendrinant galima teigti, kad eksperimentas buvo sėkmingas – daugiafunkciai dirbtinio intelekto modeliai geba identifikuoti lokalizacijos defektus ir tam tikrais atvejais net pranoksta rankinį testavimą. Vis dėlto šis metodas turi trūkumų: dažniausiai yra susijęs su papildomomis sąnaudomis, reikalauja kruopštaus konfigūravimo ir aiškiai apibrėžtų instrukcijų. Taip pat būtina užtikrinti, kad testuojamos sistemos būtų pakankamai aprašytos, jog modeliai galėtų tiksliai interpretuoti kontekstą ir pateikti patikimus rezultatus.

4.8. Eksperimento išvados

1. Tyrimo metu nustatyta, kad OCR technologija turi tam tikrą potencialą aptikti lokalizacijos defektus, ypač susijusius su gramatika ir datų formatais. Eksperimentai parodė, kad programėlių lokalizacijos defektų aptikimui efektyvesnė yra „PaddleOCR“ biblioteka, su kuria pasiektas apie 52,5 % tikslumas. Tačiau svarbu pažymėti, kad ši technologija ne visais

atvejais veikė tinkamai, nes programėlių sąsajose tekstas dažniausiai yra išdėstytas fragmentiškai, o OCR metodai yra labiau pritaikyti vientiso teksto analizei, todėl jų efektyvumas tokio tipo užduotyse yra ribotas.

2. Antrojo eksperimento metu, naudojant objektų aptikimo modelį, buvo vertinama, ar modelis geba aptikti lokalizacijos defektus, su kuriais buvo apmokytas, ir ar šis metodas turi tobulinimo potencialą. Nors pasiektas tikslumas siekė apie 31 %, nustatyta, kad metodas turi didelį potencialą. Išplėtus treniravimo duomenų aibę, padidinus klasių skaičių ir prailginus mokymo laiką, būtų galima reikšmingai pagerinti rezultatų kokybę. Be to, šis metodas pasižymi labai dideliu greičiu – vidutinis apdorojimo laikas siekė apie 0,2 sekundės – ir leidžia vizualiai identifikuoti defektus, todėl yra perspektyvus praktiniam taikymui.

3. Atlikus eksperimentą su daugiafunkciais dirbtinio intelekto modeliais nustatyta, kad sudėtingesni modeliai, turintys daugiau parametrų ir geresnį konteksto suvokimą, gali efektyviai aptikti lokalizacijos defektus. Pavyzdžiui, „Gemini 3 Flash“ pasiekė apie 67 % tikslumą net ir naudojant gana paprastą instrukciją bei ribotą programėlės kontekstą. Tikėtina, kad patobulinus instrukcijas ir pateikus išsamesnį sistemos kontekstą, šių modelių tikslumas dar labiau padidėtų.

4. Galiausiai buvo patikrintos darbui iškeltos hipotezės. Pirmoji hipotezė buvo, ar automatizavimas gali būti greitesnis nei rankinis testavimas. Ji dalinai pasiteisino: jeigu būtų naudojami visi trys algoritmai, vienos nuotraukos analizė vidutiniškai galėtų trukti apie 1 minutę, o esant lygiagrečiam procesų vykdymui – apie 40–50 s. Šį laiką padidina teksto analizė. Esant mažai sudėtingai nuotraukai, žmogus gali greičiau įvertinti įvestus duomenis ir nustatyti, ar tekstas neiškraipo vaizdo. Tačiau toks sprendimas būtų automatizuotas, o analizuojamų nuotraukų kiekis paprastai siektų ne vieną ar kelias, bet kelias dešimtis, todėl tokiu atveju darbas galėtų būti pagreitintas. Antroji hipotezė buvo, kad dirbtinio intelekto modeliai geba pateikti pataisymus. Ši hipotezė pasiteisino: nors tikslumo dar trūksta, modeliai dažniausiai sugebėdavo aptikti defektus, kurie buvo pastebėti žmogaus, ir pateikdavo logiškus jų taisymo būdus. Trečioji hipotezė teigė, kad objektų aptikimo ir dirbtinio intelekto modeliai užtikrina geriausią rezultatą. Ji dalinai pasitvirtino. Objektų aptikimo modeliai yra labai greiti ir, jei yra tinkamai apmokyti, gali aptikti objektus bei juos pažymėti pačioje nuotraukoje. Tuo tarpu dirbtinio intelekto modeliai geba identifikuoti įvairias tekstines ir dizaino klaidas. Vis dėlto jų tikslumas varijuoja, todėl reikalingas tolesnis tobulinimas, siekiant užtikrinti aukštą ir patikimą rezultatų tikslumą.

4.9. Ateities darbai

Atlikus visus tris eksperimentus, rekomenduojama toliau tęsti modelių ir algoritmų kokybės gerinimą. Dirbant su vaizdinių ir objektų aptikimo modeliais, svarbu naudoti didesnę ir įvairesnę treniravimo duomenų aibę bei išplėsti klasių rinkinį, nes dabartinis modelis buvo apmokytas su ribotu duomenų kiekiu ir nedideliu klasių skaičiumi. Padidinus duomenų aibę ir skyrus daugiau laiko apmokymui, būtų galima sukurti greitą ir tikslų modelį, gebantį efektyviai identifikuoti lokalizacijos defektus ir juos vizualiai pažymėti.

Dirbant su OCR technologija, siekiant geresnių rezultatų, reikėtų tobulinti teksto apdorojimą – ypač atsižvelgti į teksto išdėstymą programėlių ekranuose. Nors OCR metodai

pakankamai gerai aptinka žodžių klaidas, programėlių sąsajose tekstas dažnai yra išskaidytas ir netolygiai išdėstytas, todėl atsiranda artefaktų ir prastėja analizės kokybė.

Daugiafunkciai dirbtinio intelekto modeliai pasižymi dideliu potencialu – pažangesni modeliai geba tiksliai identifikuoti lokalizacijos defektus ir pasiūlyti jų taisymo būdus. Tačiau jie taip pat linkę generuoti netikslius ar klaidingus rezultatus. Siekiant pagerinti jų veikimą, reikėtų rinktis modelius, labiau pritaikytus šiai sričiai, arba, jei įmanoma, apmokyti juos su specifiniu domeno kontekstu. Vis dėlto tai reikalauja reikšmingų skaičiavimo resursų ir duomenų paruošimo.

Toliau vystant šiuos metodus ir derinant skirtingus algoritmus galima sukurti efektyvų, automatizuotą sprendimą lokalizacijos defektų aptikimui, o ateityje – galbūt ir automatiniam jų taisymui tiesiogiai programiniame kode.

Išvados

1. Atlikus literatūros ir konkurentų analizę nustatyta, kad šiuo metu automatizuoti lokalizacijos defektų aptikimo sprendimai dažniausiai reikalauja programinių žinių arba darbo su programiniu kodu. Taip pat išsiaiškinta, kad ši problema gali būti sprendžiama taikant įvairias technologijas, tokias kaip OCR, objektų aptikimas, statinė analizė ir naujausi dirbtinio intelekto daugiafunkciai modeliai. Remiantis šia analize, buvo sukurta sistema, skirta išbandyti skirtingas technologijas ir eksperimentiškai įvertinti jų veiksmingumą.
2. Buvo atliktas tyrimas, kurio metu pirmiausia įvardyta pagrindinė problema – per daug sudėtingas lokalizacijos testavimo procesas, ypač įvedus naują kalbą į sistemą, kai tampa sunku identifikuoti ir suvaldyti visas galimas klaidas. Siekiant geriau suprasti situaciją, atlikta apklausa, kurioje nustatyta, kad 47,8 % respondentų susiduria su lokalizacijos defektais, o dalis jų šiuos defektus pastebi jau po naujos sistemos versijos išleidimo. Apklausos rezultatai pagrindė poreikį ieškoti efektyvesnių sprendimų šiai problemai spręsti.
3. Eksperimento metu buvo analizuojamos dvi OCR bibliotekos – „Tesseract OCR“ ir „PaddleOCR“. Nustatyta, kad „PaddleOCR“ veikė greičiau (vidutiniškai apie 53 s vienam paveikslėliui) ir pasiekė apie 52,5% tikslumą, todėl buvo pranašesnis už „Tesseract OCR“. Nors pasiektas tikslumas yra pakankamai geras, rezultatai nėra idealūs. Pagrindinės problemos kyla dėl netolygaus teksto išdėstymo programėlių sąsajose bei spalvinės įvairovės, dėl kurios tekstas gali būti iškraipomas ir turėti artefaktų. Dėl šių priežasčių OCR technologija nėra visiškai tinkamas sprendimas lokalizacijos defektų aptikimui.
4. Eksperimento metu nustatyta, kad objektų aptikimo metodas yra perspektyvus sprendimas lokalizacijos defektų aptikimui. Nors pasiektas tikslumas siekė apie 32 % ir nėra aukštas, metodas pasižymi dideliu greičiu ir mažomis resursų sąnaudomis. Taip pat nustatyta, kad ribotas tikslumas labiau susijęs su nepakankamu treniravimo duomenų kiekiu ir mokymo trukme, o ne su paties metodo netinkamumu. Padidinus duomenų aibę, išplėtus klasių skaičių ir skyrus daugiau laiko modelio apmokymui, būtų galima pasiekti žymiai geresnius rezultatus ir šį metodą efektyviai integruoti į testavimo procesą.
5. Išbandžius didžiuosius dirbtinio intelekto pokalbių modelius nustatyta, kad geriausius rezultatus pasiekė sudėtingiausias modelis – „Gemini 3 Flash“, kurio tikslumas siekė apie 67 %. Kiti, mažesnio sudėtingumo modeliai pasiekė žemesnius rezultatus, tačiau vis tiek gebėjo identifikuoti lokalizacijos defektus ir pateikti jų taisymo pasiūlymus. Taip pat nustatyta, kad dirbtinio intelekto modeliai gali aptikti klaidas, kurios nebuvo pastebėtos rankinio testavimo metu. Siekiant pagerinti šių modelių taikymą, būtina pateikti išsamesnį programėlės kontekstą, kad modeliai galėtų įvertinti kultūrinius ir naudojimo aspektus. Taip pat svarbu naudoti aiškiai apibrėžtas ir griežtas instrukcijas bei iš anksto nurodyti galimus lokalizacijos defektų tipus, siekiant sumažinti klaidingų aptikimų skaičių ir padidinti bendrą modelių tikslumą.
6. Atlikus tyrimą ir eksperimentus nustatyta, kad hibridinis sprendimas, apjungiantis objektų aptikimą ir dirbtinio intelekto daugiafunkcių modelių taikymą, suteiktų didžiausią naudą. Abu metodai pasižymi greitu veikimu: objektų aptikimas užtrunka apie 0,29 sekundės, o DI modeliai, priklausomai nuo pasirinkto modelio, gali trukti iki 15 sekundžių. Jų integravimas į esamus testavimo procesus būtų palyginti paprastas ir patogus – per API

integraciją ir modelių pajungimą. Atsižvelgiant į technologijų vystymosi tendencijas, šie metodai turi didelį tobulėjimo potencialą. Nors eksperimento metu objektų aptikimo tikimybė nebuvo aukšta – tik 31 proc., DI modeliai pasiekė aukštesnius rezultatus, pavyzdžiui, „Gemini 3.0 Flash" – 67 proc. Taip pat buvo nustatyti šių metodų trūkumai ir jų galimi pataisymai, bei įžvelgta, kad juos kombinuojant būtų galima sukurti patrauklų sprendimą. Įgyvendinus reikiamus patobulinimus, testavimo procesas galėtų tapti efektyvesnis, greitesnis ir mažiau priklausomas nuo žmogiškojo faktoriaus.

Literatūros sąrašas

[1] Zippia, „Vital smartphone usage statistics [2023]: facts, data, and trends on mobile use in the U.S.“ Zippia Inc. [Tinkle]. [Žiūrėta: 2025 m. rugsėjo 23 d.]. Prieiga per internetą: https://www.zippia.com/advice/smartphone-usage-statistics/

[2] Testlio, „What is localization testing? Definition, How-to, and Use Cases.“ Testlio. [Tinkle]. [Žiūrėta: 2025 m. rugsėjo 24 d.]. Prieiga per internetą: https://testlio.com/blog/what-is-localization-testing

[3] J. J. Yapinski, T. D. Nursanti ir J. Scoth, „Optimizing E-Service Quality and User Experience to Enhance Customer Loyalty via Satisfaction,“ 2024 3rd Int. Conf. on Creative Communication and Innovative Technology (ICCIT), Tangerang, Indonezija, 2024, p. 1–6. [Žiūrėta: 2025 m. spalio 12 d.].

[4] A. Tariq, „Mobile Application Performance Testing: Advanced Methodologies and Quality Assurance Frameworks for Contemporary Mobile Development,“ IJMAT, t. 1, nr. 2, spal. 2025. [Žiūrėta: 2025 m. spalio 13 d.].

[5] S. Reine De Reanzi ir P. Ranjit Jeba Thangaiah, „A survey on software test automation return on investment, in organizations predominantly from Bengaluru, India,“ Int. J. Eng. Business Management, t. 13, p. 1–17, 2021. [Žiūrėta: 2025 m. spalio 14 d.].

[6] S. K. Paladugu, „Native Enterprise Mobile App Development: Best Practices for Security, Performance, and Scalability,“ Int. J. for Multidisciplinary Research (IJFMR), t. 6, nr. 5, 2024, p. 1–10. [Žiūrėta: 2025 m. spalio 15 d.].

[7] S. Mahajan, K. Gadde, A. Pasala ir W. Halfond, „Detecting and Localizing Visual Inconsistencies in Web Applications,“ 2016, p. 361–364. [Žiūrėta: 2025 m. rugsėjo 25 d.].

[8] A. Alameer, S. Mahajan ir W. G. J. Halfond, „Detecting and Localizing Internationalization Presentation Failures in Web Applications,“ 2016, p. 202–212. [Žiūrėta: 2025 m. rugsėjo 26 d.].

[9] Statista, „Cross-platform mobile frameworks used by software developers worldwide from 2019 to 2022.“ Statista Inc. [Tinkle]. [Žiūrėta: 2025 m. rugsėjo 27 d.]. Prieiga per internetą: https://www.statista.com/statistics/869224/worldwide-software-developer-working-hours/

[10] A. A. Awwad ir W. Slany, „Automated Bidirectional Languages Localization Testing for Android Apps with Rich GUI,“ Mobile Information Systems, 2016, p. 1–13. [Žiūrėta: 2025 m. rugsėjo 28 d.].

[11] G. Almosnino, „UX Best Practices for Bi-Directional Languages," Medium, 2019. [Tinkle]. [Žiūrėta: 2025 m. rugsėjo 30 d.]. Prieiga per internetą: https://medium.com/@giladalmosnino/ux-best-practices-for-bi-directional-languages-9bd7b96dc8c2

[12] A. F. Donoso Diaz, „Improving Automated i18n Testing of Android Apps," 2021, p. 1–21. [Žiūrėta: 2025 m. rugsėjo 29 d.].

[13] C. K. Gomathy, „Optical Character Recognition," 2022. [Žiūrėta: 2025 m. spalio 1 d.].

[14] Amazon Inc., „What is OCR (Optical Character Recognition)?" AWS Amazon. [Tinkle]. [Žiūrėta: 2025 m. spalio 2 d.]. Prieiga per internetą: https://aws.amazon.com/what-is/ocr/

[15] S. M. Jayanthi, D. Pruthi ir G. Neubig, „NeuSpell: A Neural Spelling Correction Toolkit," 2020. [Žiūrėta: 2025 m. spalio 4 d.].

[16] J. Zhang, C. Zhu ir Z. Zhang, „AI-powered language learning: The role of NLP in grammar, spelling, and pronunciation feedback," Applied and Computational Engineering, t. 102, p. 18–23, lapkr. 2024. [Žiūrėta: 2025 m. spalio 5 d.].

[17] Z.-Q. Zhao, P. Zheng, S.-T. Xu ir X. Wu, „Object Detection with Deep Learning: A Review," 2018, p. 3212–3232. [Žiūrėta: 2025 m. spalio 3 d.].

[18] J. Chen, M. Xie, Z. Xing, C. Chen, X. Xu ir L. Zhu, „Object Detection for Graphical User Interface: Old Fashioned or Deep Learning or a Combination?" 2020. [Žiūrėta: 2025 m. spalio 6 d.].

[19] J. Vásquez, H. K. Sharma, T. Furuhata ir K. Shimada, „Improving Deep Learning-based Defect Detection on Window Frames with Image Processing Strategies," 2023. [Žiūrėta: 2025 m. spalio 16 d.].

[20] A. Saberironaghi, J. Ren ir M. El-Gindy, „Defect Detection Methods for Industrial Products Using Deep Learning Techniques: A Review," Algorithms, t. 16, nr. 2, straipsnio nr. 95, vas. 2023. [Žiūrėta: 2025 m. spalio 17 d.].

[21] R. C. Armitage, „Performance of leading large language models in May 2025 in Membership of the Royal College of General Practitioners-style examination questions: a cross-sectional analysis," arXiv:2506.02987 [cs.CL], 2025. [Žiūrėta: 2025 m. spalio 22 d.].

[22] N. S. Harahap, A. Saad ir N. H. Ubaidullah, „Comprehensive Bibliometric Literature Review of Chatbot Research: Trends, Frameworks, and Emerging Applications," Int. J. of Advanced Computer Science and Applications (IJACSA), t. 16, nr. 1, p. 886–895, saus. 2025. [Žiūrėta: 2025 m. spalio 23 d.].

[23] S. Wollny, J. Schneider, D. Di Mitri, J. Weidlich, M. Rittberger ir H. Drachsler, „Are We There Yet? – A Systematic Literature Review on Chatbots in Education," Frontiers in Artificial Intelligence, straipsnio nr. 654924, liepos 2021. [Žiūrėta: 2025 m. spalio 24 d.].

[24] Master of Code, „Chatbot Statistics: How AI Is Powering the Rise of Digital Assistants," 2025. [Tinkle]. [Žiūrėta: 2025 m. spalio 25 d.]. Prieiga per internetą: https://masterofcode.com/blog/chatbot-statistics

[25] G. Singer, „Multimodality: A New Frontier in Cognitive AI," Towards Data Science, 2022. [Tinkle]. [Žiūrėta: 2025 m. spalio 26 d.]. Prieiga per internetą: https://towardsdatascience.com/multimodality-a-new-frontier-in-cognitive-ai-8279d00e3baf

[26] J. Yeboah ir S. Popoola, „Efficacy of static analysis tools for software defect detection on open-source projects," 2023 Int. Conf. on Computational Science and Computational Intelligence (CSCI), Las Vegas, JAV, 2023, p. 1–6. [Žiūrėta: 2025 m. spalio 27 d.].

[27] D. Badampudi, M. Unterkalmsteiner ir R. Britto, „Modern Code Reviews — Survey of Literature and Practice," ACM Trans. on Software Engineering and Methodology, t. 33, nr. 1, straipsnio nr. 4, p. 1–37, saus. 2024, doi: 10.1145/3585004. [Žiūrėta: 2025 m. spalio 18 d.].

[28] P. W. Gonçalves, P. Rani, M.-A. Storey, D. Spinellis ir A. Bacchelli, „Code Review Comprehension: Reviewing Strategies Seen Through Code Comprehension Theories," 2025 IEEE/ACM 43rd Int. Conf. on Program Comprehension (ICPC), Ottawa, Kanada, 2025, p. 589–601, doi: 10.1109/ICPC66645.2025.00068. [Žiūrėta: 2025 m. spalio 19 d.].

[29] L. Dong, H. Zhang, L. Yang, Z. Weng, X. Yang, X. Zhou ir Z. Pan, „Survey on Pains and Best Practices of Code Review," 2021 28th Asia-Pacific Software Engineering Conference (APSEC), Taipei, Taivanas, 2021, p. 482–491, doi: 10.1109/APSEC53868.2021.00055. [Žiūrėta: 2025 m. spalio 20 d.].

[30] M. Yener. „Layout Inspector," Android Studio, Medium. [Tinkle]. [Žiūrėta: 2025 m. spalio 7 d.]. Prieiga per internetą: https://medium.com/androiddevelopers/layout-inspector-1f8d446d048

[31] „Debug your layout with Layout Inspector and Layout Validation," Android Studio, Google Inc. [Tinkle]. [Žiūrėta: 2025 m. spalio 8 d.]. Prieiga per internetą: https://developer.android.com/studio/debug/layout-inspector

[32] Emerge Tools, „UI Automator," [Tinkle]. [Žiūrėta: 2025 m. spalio 9 d.]. Prieiga per internetą: https://www.emergetools.com/glossary/ui-automator

[33] Back4App, „What is Firebase Test Lab?" [Tinkle]. [Žiūrėta: 2025 m. spalio 10 d.]. Prieiga per internetą: https://blog.back4app.com/firebase-test-lab/

[34] Google Inc., „Run a Robo script (Android)," Firebase. [Tinkle]. [Žiūrėta: 2025 m. spalio 11 d.]. Prieiga per internetą: https://firebase.google.com/docs/test-lab/android/run-robo-scripts

[35] A. Ali, Y. Xia, Q. Navid, Z. A. Khan, J. A. Khan, E. A. Aldakheel ir D. Khafaga, „Mobile-UI-Repair: a deep learning based UI smell detection technique for mobile user interface," PeerJ Computer Science, t. 10, art. nr. e2028, geg. 2024, doi: 10.7717/peerj-cs.2028. [Žiūrėta: 2025 m. spalio 21 d.].