



**VILNIAUS UNIVERSITETAS**  
**MATEMATIKOS IR INFORMATIKOS FAKULTETAS**  
**PROGRAMŲ SISTEMŲ STUDIJŲ PROGRAMA**

Bakalauro baigiamasis darbas

**Varžymosi principais grįstų atakų aptikimas  
naudojant paaiškinamo dirbtinio intelekto metodą  
kenkėjiškų programų kontekste**  
**Defense Against Adversarial Malware Obfuscation Attacks  
Using Explainable Artificial Intelligence**

Liudas Kasperavičius

Darbo vadovas : prof. dr. Olga Kurasova

Recenzentas : assoc. prof. Linas Petkevičius

**Vilnius**  
**2025**

# Turinys

<b>Terminų žodynas</b>	<b>2</b>
<b>Santrumpos</b>	<b>3</b>
<b>Įvadas</b>	<b>5</b>
<b>1. Literatūros apžvalga</b>	<b>6</b>
1.1. Naudojami kenkėjiškų programų požymiai	6
1.1.1. PE formato programų požymiai	6
1.1.2. Baitų lygio požymiai	6
1.2. Perturbacijos	7
1.2.1. Baitų lygio perturbacijos	7
1.2.2. Semantinės perturbacijos	8
1.2.3. Kompleksinės perturbacijos	9
1.3. GAN tipo modelių karkasai	10
1.3.1. <i>MalGAN</i>	10
1.3.2. <i>N-gram MalGAN</i>	11
1.3.3. <i>MalFox</i>	11
1.4. Skatinamojo mokymosi tipo modelių karkasai	12
1.4.1. <i>DQEAF</i>	12
1.4.2. <i>MalInfo</i>	13
1.5. Genetinių algoritmų tipo modelių karkasai	14
1.5.1. <i>AIMED</i>	14
1.5.2. <i>GAMMA</i>	15
1.6. Nevalidaus PE formato problema	15
1.7. AE perkeliamumas	16
1.8. AE aptikimo strategijos	16
1.8.1. Varžymosi principais pagrįstas pertreniravimas	16
1.8.2. Gradientų slėpimas	16
1.8.3. Kategorijų švelninimas	16
1.8.4. Perkeliamumo blokavimas	16
1.8.5. Bazės keitimas, transformacijos	17
1.9. <i>LIME</i> metodas	17
1.9.1. Varžymosi principais pagrįstų atakų aptikimas IDS naudojant <i>LIME</i>	18
1.10. Dimensijų mažinimo metodai	19
1.10.1. Principinių komponentų analizė (PCA)	19
1.10.2. Daugialypės korespondencijos analizė (MCA)	19
<b>2. MCA ir <i>LIME</i> AE aptikimo metodų sintezė</b>	<b>20</b>
2.1. <i>LIME</i> pritaikymo AE aptikimui metodo modifikacijos	20
2.2. <i>LIME</i> branduolio pločio pasirinkimas	20
<b>3. Klasifikavimo tikslumo TODO tyrimas</b>	<b>22</b>
3.1. Tyrimo metodika	22
3.2. MCA komponentų pasirinkimas	22
3.3. Eksperimentai	25
3.3.1. Originalaus klasifikatoriaus tikslumo nustatymas	25
3.3.2. MCA transformacijos klasifikatoriaus tikslumo nustatymas	26
3.3.3. Modifikuoto <i>LIME</i> pritaikymo AE aptikimui tikslumo nustatymas	27
3.3.4. <i>LIME</i> ir MCA metodų sintezės tikslumo nustatymas	28
<b>Literatūra ir šaltiniai</b>	<b>29</b>

# Terminų žodynas

**Inercija** (*angl. inertia*). variacijos dalis, kurią „paaiškina“ PCA komponentė 17, 19, 23

**Karkasas** (*angl. Framework*). Nurodo specifines technologijas, naudojamus požymius ir perturbacijas, siekiamus tikslus AE generacijai. Skirtas apibrėžti procesą ir įrankius, kuriuos naudojant būtų galima generuoti nurodytų tikslų siekiančius AE 2, 6, 7, 10, 11, 12, 13, 14, 15

**Maišymo funkcija**. Tai funkcija  $f : \{0,1\}^* \rightarrow \{0,1\}^m$ . Naudojama, kai iš begalinės įvesčių erdvės norima gauti fiksuoto dydžio ( $m$ ) išvestį 3, 6

**Nulinės sumos žaidimas** (*angl. Zero-Sum Game*). Dviejų žaidėjų žaidimas, kuriame galimas vienas laimėtojas. Laimėtojo laimėta suma yra lygi pralaimėtojo pralaimėtai sumai 10

**Pėdsakas**. Programos struktūros ir požymių santrauka, beveik unikalčiai identifikuojanti programą (pvz., maišymo funkcija) 5

**Q-Funkcija** (*angl. Q-Function*).  $Q : S \times A \rightarrow \mathbb{R}$ , čia  $S$  – galimų būsenų erdvė (*angl. State Space*),  $A$  – galimų veiksmų erdvė (*angl. Action Space*) 12

**Sprendimų priėmimo riba** (*angl. Decision Boundary*). Paprasčiausiems ML modeliams tai yra kreivė plokštumoje. Sudėtingesniems – daugiadimensiniams modeliams – daugdara (*angl. manifold*) 5

**Strategija** (*angl. Policy*). Tai funkcija  $\pi : S \times A \rightarrow \{0,1\}$ , čia  $S$  – galimų būsenų erdvė (*angl. State Space*),  $A$  – galimų veiksmų erdvė (*angl. Action Space*). Šią funkciją RL modelis „išmoksta“ mokymosi metu 12

**Surogatinis Modelis** (*angl. Surrogate Model*). ML modelis, aproksimuojantis kitą ML modelį, kurio parametrai (svoriai) nėra žinomi 10, 11, 12, 13, 14, 15

**Varžymosi principais grįstas treniravimas**. ML modelio treniravimas naudojant AE kaip mokymosi duomenis. Viena iš apsisaugojimo nuo varžymosi principais pagrįstų atakų strategijų 5

**Varžymosi principais pagrįstos atakos** (*angl. Adversarial Attacks*). Tai atakos, pritaikytos „apgauti“ ML klasifikatorius 2, 3, 5, 6, 9, 11, 12, 15, 16, 17, 18, 19, 22, 28

**„Juodos dėžės“ ataka**. varžymosi principais pagrįstos atakos atvejis, kai atakuojamo ML modelio parametrai bei klasifikacijos tikimybiniai įverčiai nėra žinomi. 16

## Santrumpos

- AE.** Varžymosi principais pagrįstomis atakomis obfuskuoti kenkėjiško kodo pavyzdžiai (*angl. adversarial examples*) 2, 3, 5, 7, 10, 11, 13, 14, 16, 17, 18, 20, 21, 22, 23, 27
- API.** *angl. Application Programming Interface* 6, 8, 10, 15
- CNN.** *angl. convolutional neural network* 11, 12
- DI.** Dirbtinis intelektas (*angl. artificial intelligence*) 5, 16
- DLL.** Dinamiškai susieta biblioteka (*angl. dynamic link library*) 6, 8, 9, 12, 15
- GA.** Genetiniais algoritmais pagrįstas ML modelis (*angl. genetic algorithm*) 14
- GAN.** Generatyviniai priešiški tinklai (*angl. generative adversarial networks*) 10, 11, 14
- GBDT.** *angl. gradient boosted decision trees* 11, 12, 14, 15
- IDS.** *angl. Intrusion Detection System* 2, 18
- KNN.** *angl. K-Nearest Neighbours* 11
- MCA.** *angl. Multiple Correspondence Analysis* 2, 19, 20, 21, 22, 23, 24, 26, 28
- ML.** Mašininis mokymasis (*angl. machine learning*) 3, 4, 5, 6, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20
- NLP.** Skaitmeninis natūraliosios kalbos apdorojimas (*angl. natural language processing*) 6
- PCA.** *angl. Principal Component Analysis* 2, 3, 17, 19
- PE.** *angl. portable executable* 2, 6, 7, 8, 15
- RL.** Skatinamasis mokymasis (*angl. reinforcement learning*) 3, 12, 13, 14
- SVM.** *angl. support vector machine* 10, 11
- XAI.** paaiškinamas dirbtinis intelektas (*angl. explainable artificial intelligence*) 17

## Įvadas

Pastaraisiais metais kenkėjiškas kodas ir programos kuriamos itin sparčiai (~450000 kenkėjiškų programų per dieną 2024 m. AV-TEST<sup>1</sup> duomenimis). Kenkėjiško kodo aptikimo programos, kurios tradiciškai remiasi programų pėdsakais, nespėja atnaujinti pėdsakų duomenų bazių pakankamai greitai. Dėl to DI, tiksliau mašininio mokymosi (ML), naudojimas kenkėjiškų programų ar kenkėjiško kodo aptikimo srityje tapo itin populiarus [DCB<sup>+</sup>21]. Tačiau ML modeliai, nors ir geba aptikti kenkėjiškas programas iš naujų, dar nematytų, duomenų, yra pažeidžiami varžymosi principais pagrįstoms atakoms [CSD19; HT17; RSR<sup>+</sup>18; ZHZ<sup>+</sup>22]. Šių atakų principas yra ML modelio – klasifikatoriaus – sprendimų priėmimo ribos radimas – žinant šią ribą pakanka pakeisti kenkėjiškos programos veikimą taip, kad ML modelis priimtų sprendimą klasifikuoti ją kaip nekenksmingą [DCB<sup>+</sup>21]. Nustatyta, jog šią ribą galima rasti tiek žinant klasifikatoriaus parametrus, tiek jų nežinant ir net turint labai ribotą prieigą prie klasifikatoriaus rezultatų (pvz., klasifikacijos rezultatą be tikimybių – tokios sąlygos vadinamos „juodos dėžės“ atvejais) [FWL<sup>+</sup>19].

Vis tik varžymosi principais pagrįstos atakos nėra neįveikiamos. Nuolat kuriami nauji jų aptikimo metodai, tokie kaip varžymosi principais grįstas treniravimas, gradientų slėpimas ir kt. Kiekvienas metodas turi savų stiprybių ir silpnų bei dažniausiai remiasi viena iš specifinio ML modelio įgyvendinimo savybių, kitaip tariant, nėra vieno geriausio, tinkamiausio ar teoriškai teisingo varžymosi principais pagrįstų atakų aptikimo metodo. Tiksliau, nėra pačių AE konstravimo teorinio modelio, dėl šio proceso kompleksiskumo, tad jų aptikimo strategijos teorinis modelis taip pat nėra žinomas [CAD<sup>+</sup>21]. Šiame darbe siekiama generalizuoti AE aptikimą apjungiant panašiam kenkėjiškų programų aptikimo kontekste naudojamą *LIME* [RSG16] metodą ir kitas mokslinėje literatūroje aprašytas technikas.

**Tikslas** – pritaikyti *LIME* metodą sėkmingų varžymosi principais grįstų atakų aptikimui ir paaiškinimui vertinant bet kokius požymius.

### Uždaviniai:

1. Apžvelgti kenkėjiško kodo obfuskacijos metodus bei apsisaugojimo nuo jų strategijas.
2. Pritaikyti varžymosi principais pagrįstų atakų aptikimą dvejetainius požymių vektorius naudojantiems modeliams taikant dimensijų mažinimo metodus.
3. Sukurti klasifikavimo proceso praplėtimą į jį įtraukiant varžymosi principais pagrįstos atakos aptikimą.
4. Ištirti praplėsto klasifikavimo proceso tikslumą (*angl. accuracy*).

---

<sup>1</sup><https://www.av-test.org/en/statistics/malware>

# 1. Literatūros apžvalga

## 1.1. Naudojami kenkėjiškų programų požymiai

Varžymosi principais pagrįsta ataka taikosi į ML modeliais paremtus kenkėjiškų programų detektorius. Šie detektoriai yra klasifikatoriai – pateiktą (programą) klasifikuoja kaip kenkėjišką (*angl. malicious*) arba nekenkėjišką (*angl. benign*). Kadangi programos nėra fiksuoto dydžio, klasifikatoriai remiasi programų požymiais, kurie gaunami atliekant požymių ištraukimą (*angl. feature extraction*). Laikoma, jog „juodos dėžės“ atvejais sužinoti, kokius tiksliai požymius vertina kenkėjiškų programų detektoriai, yra neįmanoma, tad ERRORkarkasas apibrėžimuose, priklausomai nuo jų specifikos ir tikslų, neretai pateikiami jų vertinami programų požymiai. Šiame poskyryje išskiriami ir klasifikuojami mokslinėje literatūroje minimi požymiai.

### 1.1.1. PE formato programų požymiai

Išskiriami šie pagrindiniai PE formato programų požymiai:

- **DLL vardai (arba API vardai [HT17])** [ZCY<sup>+</sup>24]. PE faile turi būti nurodyti visi naudojami DLL ir jų API. Prieš pradedant mokytį ML modelį, atliekama visų turimų programų analizė ir nustatoma visų naudojamų DLL ar jų API aibė  $D$ . Tarkime  $|D| = n$ . Tuomet, požymių vektorius programai, naudojančiai  $X \subseteq D$  DLL, bus  $n$ -matis dvejetainis vektorius, kurio  $i$ -asis elementas yra 
$$\begin{cases} 0, & \text{jei } D_i \notin X, \\ 1, & \text{jei } D_i \in X \end{cases} \quad \text{čia } D_i - i\text{-asis } D \text{ elementas.}$$
- **PE metaduomenys** [AKF<sup>+</sup>18]. Tai visi PE formato faile esantys metaduomenys, tokie, kaip sekcijų pavadinimai, sekcijų dydžiai, *ImportTable* ir *ExportTable* metaduomenys ir kt. Formuojant požymių vektorių skaičiuojama metaduomenų maišymo funkcija.

### 1.1.2. Baitų lygio požymiai

Baitų lygio požymiai gali būti ištraukiami iš bet kokio formato failų. Mokslinėje literatūroje minimi šie pagrindiniai baitų lygio požymiai:

- **Prasmingų žodžių (angl. Strings) kiekis** [AKF<sup>+</sup>18]. Prasmingus žodžius suprantame kaip turinčius prasmę žmogui (*angl. human readable*). Tai gali būti URL, failų keliai (*angl. file paths*) ar registro raktų pavadinimai. Kadangi prasmingų žodžių kiekis tėra vienas skaičius, požymių vektorius dažniausiai formuojamas prijungiant ir kitus požymius.
- **Baitų/entropijos histograma** [SB15]. Specifinis metodas, užkoduojantis dažniausiai pasikartojančias baitų ir entropijos poras  $n$  dimensių vektoriumi.
- **$n$ -gramos** [ZZY<sup>+</sup>22]. Dažniausiai sutinkamos skaitmeniniame natūraliosios kalbos apdorojime (NLP). Tai yra  $n$  žodžių junginiai, arba, sukompiliuotų programų apdorojimo kontekste,  $n$

baitų junginiai. Nustatant požymių vektorių, visos  $n$ -gramos surikiuojamos pagal pasikartojimą programoje mažėjimo tvarka („populiariausios“ viršuje). Iš pirmų  $m$  reikšmių sudaromas  $m$ -matis vektorius – tai ir yra požymių vektorius.

## 1.2. Perturbacijos

Perturbacijos – tai pagrindinis obfuskacijos metodas AE kūrimui. Perturbacijų tikslas yra pakeisti kenkėjiškos programos veikimą išsaugant originalų funkcionalumą. Perturbacijos gali būti sudėtingos ir apimti visą programą (pvz., visos programos užšifravimas ir pridėjimas prie kitos programos), semantinės (pvz., tam tikrų mašininio kodo instrukcijų keitimas į ekvivalentų rezultatą pasiekiančias) arba baitų lygio (pvz., nulinių baitų pridėjimas programos gale) [HT17]. Perturbacijų parinkimas įeina į karkaso apibrėžimą. Šiame poskyryje aptariamos mokslinėje literatūroje minimos perturbacijos.

### 1.2.1. Baitų lygio perturbacijos

Pačias paprasčiausias baitų lygio perturbacijas galima taikyti bet kokio formato failams, tačiau labiau prasmingos perturbacijos taikomos PE formato failams. Išskiriamos šios pagrindinės baitų lygio perturbacijos:

- **ARBE (Append Random Bytes at the End)** [FWL<sup>+</sup>19]. PE formato failo gale pridedami atsitiktiniai baitai.
- **ARI (Append Random Import)** [FWL<sup>+</sup>19]. PE formato failo *ImportAddressTable* lentelėje pridedama atsitiktinai pavadinta biblioteka su atsitiktinai pavadinta funkcija.
- **ARS (Append Randomly named Section)** [FWL<sup>+</sup>19]. PE formato failo *SectionTable* lentelėje pridedamos atsitiktinės sekcijos (sekcijos ir jų tipai yra apibrėžti PE formate).
- **RS (Remove Signature)** [FWL<sup>+</sup>19]. Sertifikato pašalinimas iš PE formato failo *CertificateTable* lentelės.
- **Naujas įeities taškas** [AKF<sup>+</sup>18]. Prasidėjus programai, iškart peršokama nuo naujo įeities taško į originalųjį.
- **Header Fields** [DCB<sup>+</sup>21]. PE formato failo *PE Header* ir *Optional Header* dalių specifinių laukų keitimas (pvz., sekcijos pavadinimo keitimas [AKF<sup>+</sup>18]).
- **Partial DOS** [DCB<sup>+</sup>21]. PE formato failo *DOS Header* dalies pirmi 58 baitai po *MZ* skaičiaus yra nenaudojami moderniose operacinėse sistemose, tad juos galima keisti.
- **Slack Space** [DCB<sup>+</sup>21]. Dėl PE formato specifikos, kiekviena nauja sekcija turi prasidėti tam tikro skaičiaus, nurodyto *PE Header* dalyje, kartotiniu nuo pradžios. Kompiliatoriai šį reikalavimą išpildo sekcijų gale pridėdami tiek nulinių baitų, kiek reikia teisingam sulygiavimui pasiekti. Būtent ši nulinių baitų erdvė gali būti keičiama be jokios įtakos originaliai programai.

- **Padding** [DCB<sup>+</sup>21]. Nulinių baitų pridėjimas failo gale.
- **Full DOS** [DCB<sup>+</sup>21]. Perturbacijos esmė tokia pat, kaip ir *Partial DOS*, tik naudojami visi *DOS* dalies baitai, išskyrus *MZ* ir *PE Offset* (*Partial DOS* manipuliacijoms naudoja tik dalį tarp *MZ* ir *PE Offset*).
- **Extend** [DCB<sup>+</sup>21]. Pakeičiama PE formato faile *DOS* dalyje esanti *PE Offset* reikšmė į didesnę<sup>2</sup>. Taip padidinama (išplečiama) visa *DOS* dalis. Tolesnis perturbacijos principas yra toks pat, kaip ir *Full DOS*.
- **Shift** [DCB<sup>+</sup>21]. PE formato failuose kiekvienas sekcijos blokas prasideda su sekcijos vieta nuo pradžios (*angl. offset*). Tarkime ši reikšmė yra  $S$ . Sekcijos kodas pradedamas vykdyti tik nuo adreso  $P + S$ , kur  $P$  – programos pradžios adresas. Vadinasi, padidinus<sup>2</sup>  $S$  per  $n$ , atsiranda  $n$  baitų laisvos vietos iki sekcijos pradžios, kurią galima keisti be jokios įtakos programos veikimui.

### 1.2.2. Semantinės perturbacijos

Semantinių perturbacijų įgyvendinimas taip pat atliekamas baitų lygyje, tačiau šie pokyčiai turi aukštesnio lygio prasmę. Išskiriamos šios semantinės perturbacijos:

- **Nereikalingų DLL/API vardų požymių pridėjimas** [HT17]. PE formato faile *ImportTable* lentelėje pridedami originalios programos nenaudojami DLL/API vardai.
- **Binary Rewriting** [DCB<sup>+</sup>21]. Semantinis instrukcijų perrašymas. Pavyzdžiui,  $A + B$  instrukcijos pakeitimas į  $A - (-B)$ .

---

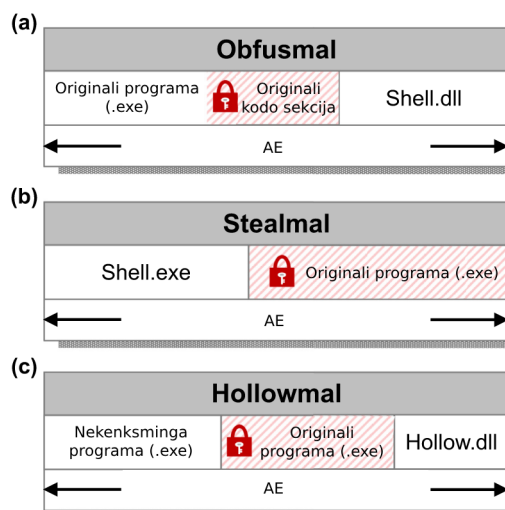
<sup>2</sup>šios reikšmės padidinimas reiškia visos failo struktūros keitimą (*DOS* dalis yra failo pradžioje). Būtina pakeisti visų sekcijų vietas nuo pradžios (*angl. offset*) jų metaduomenyse.



### 1.2.3. Kompleksinės perturbacijos

Kompleksinės perturbacijos yra pritaikomos tam tikriems tikslams. Obfuskacijos ir varžymosi principais pagrįstų atakų tikslams literatūroje minimos šios kompleksinės perturbacijos:

- **Obfusmal** [ZCY<sup>+</sup>24]. Užšifruojama originalios programos kodo sekcija. Sukuriama ir originalios programos gale pridedama programa *Shell.dll*, kurioje laikomas atšifravimo raktas, originalios programos kodo sekcijos adresas ir dydis. Be to, *Shell.dll* geba atšifruoti originalios programos kodo sekciją ir jai perduoti kontrolę. *Shell.dll* pridedama prie naudojamų DLL, o programos pradžios taškas nustatomas į *Shell.dll* pradžios tašką. Iliustracija pateikiama 1-ame pav.
- **Stealmal** [ZCY<sup>+</sup>24]. Visa originali programa užšifruojama ir pridedama prie programos *Shell.exe* galo. *Shell.exe* geba atšifruoti originalią programą ir perduoti jai kontrolę. Iliustracija pateikiama 1-ame pav.
- **Hollowmal** [ZCY<sup>+</sup>24]. Užšifruojama visa originali programa. Ji pridedama prie kurios nors nekenksmingos programos galo. Prie šio junginio galo pridedama *Hollow.dll* programa, kurios veikiamas panašus į *Shell.exe* iš *Stealmal*. Viso junginio pradžios taškas nustatomas į *Hollowmal.dll* pradžios tašką. Iliustracija pateikiama 1-ame pav.



**1 pav.** Obfusmal (a), Stealmal (b) ir Hollowmal (c) perturbacijų veikimo principų iliustracijos. Adaptuota iš [ZH<sup>+</sup>22]

### 1.3. GAN tipo modelių karkasai

GAN modelių karkasai paremti generatyviniais priešiškais tinklais (GAN), kurių veikimo principas yra du neuroniniai tinklai (generatorius ir diskriminatorius), žaidžiantys nulinės sumos žaidimas [CDH<sup>+</sup>16]. Kenkėjiško kodo obfuskacijos kontekste ir ypač „juodos dėžės“ atvejais, diskriminatorius atlieka surogatinis modelis vaidmenį. Bendras GAN modelių mokymosi etapas yra tokia seka [HT17; ZCY<sup>+</sup>24; ZZY<sup>+</sup>22]:

1. Generatorius, naudodamas požymių vektorių ir tokios pačios dimensijos „triukšmo“ (*angl. noise*) vektorių, sugeneruoja perturbacijas.
2. Originali kenkėjiška programa modifikuojama pagal perturbacijas (sukuriamas AE).
3. Diskriminatorius klasifikuoja sugeneruotą AE (kenkėjiškas / nekenkėjiškas). Pagal klasifikacijos rezultatą skaičiuojamos diskriminatoriaus ir generatoriaus nuostolių funkcijos reikšmės (diskriminatoriaus nuostolių funkcijos reikšmė priklauso nuo tikro detektoriaus klasifikacijos).
4. Visa seka kartojama nustatytą kiekį kartų.

#### 1.3.1. MalGAN

Tai vienas iš pirmųjų ir populiariausių GAN tipo modelių ERRORkarkasas. „Juodos dėžės“ detektoriai čia apibrėžiami kaip populiarius ML klasifikatoriai, tokie, kaip MLP (*angl. Multilayer Perceptron*), RF (*angl. Random Forest*), DT (*angl. Decision Tree*), SVM (*angl. Support Vector Machine*). MalGAN karkaso [HT17] tikslas ir apibrėžimas pateikiami 1-oje lentelėje.

**1 lentelė. MalGAN karkasas**

<b>Tikslas</b>	Efektyviai išvengti AE aptikimo, kai ML kenkėjiškų programų detektoriaus įgyvendinimas nežinomas („juodos dėžės“ atvejis).
<b>Surogatinis modelis</b>	Daugiasluoksnis tiesioginio sklidimo neuroninis tinklas – klasifikatorius. Įvestis – programos požymių vektorius. Išvestis – klasifikacija į kenksmingą arba nekenksmingą. Šis tinklas taip pat naudojamas kaip diskriminatorius GAN architektūroje.
<b>ML modelis</b>	Daugiasluoksnis tiesioginio sklidimo neuroninis tinklas. Įvestis – programos požymių vektorius ir tokios pačios dimensijos „triukšmo“ vektorius. Išvestis – modifikuotas požymių vektorius. Šis tinklas naudojamas kaip generatorius GAN architektūroje.
<b>Požymiai</b>	MalGan straipsnyje [HT17] naudojami tik API vardų požymiai, patenkantys į PE formato programų požymių kategoriją (žr. 1.1.1.), tačiau autoriai nurodo, jog gali būti naudojami bet kokie požymiai <sup>3</sup> .
<b>Perturbacijos</b>	Semantinės perturbacijos (1.2.2.) – nereikalingų API vardų požymių pridėjimas.

<sup>3</sup>autoriai nagrinėja „juodos dėžės“ atvejį su prielaida, jog detektoriaus naudojami požymiai yra žinomi.

### 1.3.2. *N-gram MalGAN*

Šis karkasas remiasi *MalGAN* (1.3.1.) karkasu ir siekia jį pagerinti. *N-gram MalGAN* karkaso [ZZY<sup>+</sup>22] tikslas ir apibrėžimas pateikiami 2-oje lentelėje.

**2 lentelė.** *N-gram MalGAN* karkasas

<b>Tikslas</b>	Supaprastinti, pagreitinti ir pagerinti varžymosi principais pagrįstas atakas. Pašalinti prielaidas <sup>3</sup> apie detektorių „juodos dėžės“ atvejais.
<b>Surogatinis modelis</b>	Surogatinio modelio veikimas ir architektūra tokia pati, kaip ir <i>MalGAN</i> (1.3.1.).
<b>ML modelis</b>	Pagrindinio modelio veikimas ir architektūra labai panašūs į <i>MalGAN</i> (1.3.1.), tačiau norėdami stabilizuoti mokymosi procesą, autoriai siūlo nenaudoti „triukšmo“ vektorių. Vietoje to, generatoriaus išvestis ( $n$ -matis vektorius) modifikuojama nekeičiant pirmų $m$ dimensijų, o kitas $n - m$ pakeičiant nekenksmingų programų požymiais.
<b>Požymiai</b>	Baitų lygio požymiai (1.1.2.) – $n$ -gramos.
<b>Perturbacijos</b>	Autoriai neatliko eksperimentų su perturbuotomis programomis, tačiau pažymi, jog norint gauti sugeneruotus požymių vektorius užtenka pridėti reikiamus baitus programos gale. Tai atitinka 1.2.1. apibrėžtą baitų lygio perturbaciją <i>ARBE</i> , tik šiuo atveju pridėdami baitai nebūtų atsitiktiniai, o norimos $n$ -gramos.

### 1.3.3. *MalFox*

*MalFox* taip pat remiasi *MalGAN* (1.3.1.), tačiau siekia kurti AE realiomis sąlygomis, dėl to atlieka esminius pakeitimus. *MalFox* karkaso [ZCY<sup>+</sup>24] tikslas ir apibrėžimas pateikiami 3-oje lentelėje.

**3 lentelė.** *MalFox* karkasas

<b>Tikslas</b>	Generuoti AE, kurių neaptiktų komerciniai detektoriai (prieš tai aptarti karkasai eksperimentams kaip nepriklausomą detektorių naudojo tokius ML modelius, kaip SVM, KNN, GBDT ir kt., bet ne komercinius detektorius). Šio karkaso detektorius yra <i>VirusTotal</i> (viešai prieinama paslauga, agreguojanti virš 70 komercinių kenkėjiškų programų detektorių).
<b>Surogatinis modelis</b>	Surogatinis modelis, kaip ir kituose GAN tipo modelių karkasuose, naudojamas kaip diskriminatorius. Įvestis – perturbuota programa. Išvestis – klasifikacija į kenksmingą arba nekenksmingą. Įgyvendinimas – konvoliucinis neuroninis tinklas (CNN).
<b>ML modelis</b>	Standartinis GAN generatorius, požymių vektorių sujungiantis su „triukšmo“ vektoriumi. Įgyvendinimas – konvoliucinis neuroninis tinklas (CNN).

<b>Požymiai</b>	PE formato programų požymiai (1.1.1.) – DLL vardai.
<b>Perturbacijos</b>	Visos kompleksinės perturbacijos (1.2.3.).

## 1.4. Skatinamojo mokymosi tipo modelių karkasai

Skatinamojo mokymosi (RL) modeliai susideda iš agento ir aplinkos. Aplinka susideda iš informatyvių požymių ištraukimo metodo (*angl. feature extraction*) ir kenkėjiškų programų detektoriaus. Šiuo atveju aplinkos būsenų erdvė  $S$  yra požymių vektorių erdvė. Agentas – tai algoritmas ar neuroninis tinklas, kurio tikslas yra surasti optimalią strategiją strategija (*angl. policy*) ko. Šiuo atveju strategijos veiksmų erdvė  $A$  susideda iš perturbacijų (žr. 1.2.) [FWL<sup>+</sup>19]. Bendras RL modelių mokymosi etapas yra tokia seka [FWL<sup>+</sup>19; ZHZ<sup>+</sup>22]:

1. Agentas, naudodamas dabartinę aplinkos būseną ir praeito veiksmo atlygį (*angl. reward*), parenka sekantį veiksmą iš galimų veiksmų aibės ir taiko mokymosi algoritmą (algoritmas priklauso nuo agento įgyvendinimo).
2. Atliekamas veiksmas – perturbuojama programa arba požymių vektorius (priklauso nuo karkaso).
3. Gaunami aplinkos kitimo įverčiai – nauja būsena ir atlygis, skaičiuojamas pagal detektoriaus klasifikacijos rezultatą.
4. Seka kartojama tol, kol agentas nelaiko strategijos optimalia arba nustatytą kiekį kartų.

### 1.4.1. DQEAF

Šis karkasas taiko gilųjį skatinamąjį mokymąsi, kai agentas implementuojamas kaip gilusis neuroninis tinklas. DQEAF karkaso [FWL<sup>+</sup>19] tikslas ir apibrėžimas pateikiami 4-oje lentelėje.

**4 lentelė.** DQEAF karkasas

<b>Tikslas</b>	Parodyti, jog ML kenkėjiškų programų detektoriai, ypač modeliai, išmokyti prižiūrimu mokymusi, yra pažeidžiami varžymosi principais pagrįstoms atakoms.
<b>Surogatinis modelis</b>	RL karkasuose nenaudojami surogatiniai modeliai. Kaip „juodos dėžės“ detektorius pasirinktas GBDT modelis.
<b>ML modelis</b>	Agentas implementuotas kaip gilusis $Q$ -tinklas (CNN praplėtimas, kai tinklas naudojamas kaip $Q$ -funkcijos aproksimacija). Taip pat taikomas prioritetizuotas patirčių pakartojimo metodas ( <i>angl. prioritized experience replay</i> ), kuomet agentas mokomas tik su aukštą atlygį gavusiais perėjimais ( $S \times A$ ).

<b>Požymiai</b>	Požymių vektorius taip pat apibrėžia visų būsenų erdvę $S$ . Šiuo atveju $S = \mathbb{R}^{513}$ . Baitų lygio požymiai (1.1.2.) – baitų/entropijos histograma.
<b>Perturbacijos</b>	<p>Perturbacijos apibrėžia visų galimų agento veiksmų erdvę <math>A</math>. Šiuo atveju <math>A = \{0,1\}^4</math>. Baitų lygio perturbacijos (1.2.1.):</p> <ul style="list-style-type: none"> <li>• <math>ARBE</math></li> <li>• <math>ARI</math></li> <li>• <math>ARS</math></li> <li>• <math>RS</math></li> </ul>

#### 1.4.2. *MalInfo*

*MalInfo* remiasi *MalFox* (1.3.3.). *MalInfo* karkaso [ZHZ<sup>+</sup>22] tikslas ir apibrėžimas pateikiami 5-oje lentelėje.

**5 lentelė.** *MalInfo* karkasas

<b>Tikslas</b>	Surasti optimalią obfuskacijos strategiją konkrečiai programai, pagal kurią sukurtas AE nebūtų aptiktas komercinių kenkėjiškų programų detektorių.
<b>Surogatinis modelis</b>	RL surogatinis modelis nenaudojamas. „Juodos dėžės“ detektoriumi pasirinkti komerciniai detektoriai ( <i>VirusTotal</i> ).
<b>ML modelis</b>	Agentas implementuotas kaip klasikiniai ML algoritmai (konkrečiai dinaminis programavimas ir skirtumų laike ( <i>angl. temporal difference</i> ) algoritmas).
<b>Požymiai</b>	Agentas nėra neuroninis tinklas ir požymių iš programos netraukia. Agentas mokosi tik iš perėjimų, o būsenų erdvė $S$ yra originali programa ir perturbuoti jos variantai. Teoriškai perturbuotų programos variantų galėtų būti be galo daug, tuomet $S = A^\infty$ , $ S  = \aleph_0$ , tačiau autoriai nurodo, jog daugiau nei 3 sluoksniai kompleksinių perturbacijų reikšmingai paveikia programos veikimo laiką, o tai gali „sukelti įtarimų“ komerciniams detektoriams. Todėl pasirinkta $S = A^3$ .
<b>Perturbacijos</b>	<p><math>A = \{0,1,2,3\}</math></p> <ul style="list-style-type: none"> <li>• <i>Null</i> perturbacija – naudinga tik formaliam pilnumui (atitinka nulinį <math>A</math> veiksmą).</li> <li>• Visos kompleksinės perturbacijos (1.2.3.), t. y. tokios pačios, kaip ir <i>MalFox</i> (1.3.3.) karkaso.</li> </ul>

## 1.5. Genetinių algoritmų tipo modelių karkasai

Genetiniai algoritmai (GA) yra viena seniausių mašininio mokymosi ML apraiškų; jų veikimas paremtas evoliucija [CSD19]. Kenkėjiškų programų obfuskacijai AE generavimas taikant GA yra tokia seka [YPT22]:

1. Sukuriama pradinė populiacija (perturbacijos metodai pradinei populiacijai priklauso nuo karkaso).
2. Atliekamas tinkamumo (*angl. fitness*) vertinimas.
3. Atliekama selekcija – dažniausiai pasirenkami geriausiai įvertinti populiacijos AE, tačiau galimos ir kitos selekcijos strategijos.
4. Atliekamas selekcijos atrinktų AE kryžminimas (po 2) taip sukuriant naują AE, turintį po dalį genų iš abiejų kryžmintų AE.
5. Tam tikrai daliai AE atliekama dalies genų mutacija.
6. Vertinama, ar sugeneruoti AE atitinka kriterijus (vertina detektorius).
7. Jei kriterijai nėra tenkinami, seka kartojama nuo 2-o žingsnio.

### 1.5.1. AIMED

AIMED karkaso [CSD19] tikslas ir apibrėžimas pateikiami 6-oje lentelėje.

**6 lentelė. AIMED karkasas**

<b>Tikslas</b>	AE generavimo greičio padidinimas ir modelių kompleksiskumo sumažinimas, lyginant su GAN ir RL tipo modelių karkasais.
<b>Surogatinis modelis</b>	Surogatinis modelis nenaudojamas. Naudojami „juodos dėžės“ detektoriai yra <b>3 komerciniai</b> ( <i>Kaspersky, ESET, Sophos</i> ) ir vienas ML modelis – GBDT.
<b>ML modelis</b>	Klasikinis GA modelis – veikimas visiškai atitinką bendrą seką. Tinkamumo ( <i>angl. fitness</i> ) vertinamas remiasi AE požymių vektoriaus panašumu į originalios programos požymių vektorius (kuo mažiau panašūs, tuo tinkamumo įvertinimas didesnis).
<b>Požymiai</b>	Baitų lygio požymiai (1.1.2.) – atskiras $n$ -gramų atvejis, kai $n = 1$ .
<b>Perturbacijos</b>	Baitų lygio perturbacijos <sup>4</sup> (1.2.1.).

<sup>4</sup> autoriai rėmėsi perturbacijomis, aprašytais [AKF<sup>+</sup>18]

### 1.5.2. GAMMA

GAMMA karkaso [DBL<sup>+</sup>21] tikslas ir apibrėžimas pateikiami 7-oje lentelėje.

**7 lentelė. GAMMA karkasas**

<b>Tikslas</b>	Efektyvus (neaptikimo šansų didinimas naudojant perturbacijas, paremtas nekenksmingomis programomis) varžymosi principais pagrįstų atakų kūrimas.
<b>Surogatinis modelis</b>	Surogatinis modelis nenaudojamas. GBDT ir <i>MalConv</i> pasirinkti kaip „juodos dėžės“ detektoriai.
<b>ML modelis</b>	Pagrindinė modelio idėja yra požymių ištraukimas iš nekenksmingų programų ir jų pridėjimas, naudojant tam pritaikytas perturbacijas, į kenksmingas programas kiekvienos populiacijos generavimo metu. Tinkamumo ( <i>angl. fitness</i> ) ir kriterijų vertinimas atliekamas naudojant detektorių ir pridėtų požymių dydį baitais (norima pridėti kuo mažiau požymių).
<b>Požymiai</b>	<ul style="list-style-type: none"><li>• PE formato programų požymiai (1.1.1.).</li><li>• Kodas sekcijose (nestandartinis požymis).</li></ul>
<b>Perturbacijos</b>	<ul style="list-style-type: none"><li>• Visos baitų lygio perturbacijos (1.2.1.), gebančios pridėti baitus.</li><li>• Autoriai pažymi, jog gali būti naudojama ir DLL / API vardų pridėjimo semantinė perturbacija (1.2.2.).</li></ul>

## 1.6. Nevalidaus PE formato problema

Andersonas ir kt. [AKF<sup>+</sup>18], atlikdami eksperimentus su funkcionalumą išlaikančiomis perturbacijomis PE formato failams, pastebėjo, jog ne visais atvejais perturbuotos programos veikia teisingai. Dėl *Windows* operacinės sistemos PE formato failų interpretavimo ir paleidimo specifikos, programas įmanoma parašyti tokiu būdu, jog pakeitus kodo ar kitų sekcijų turinį nekeičiant originalių mašininio kodo instrukcijų, programa neveiktų. Techniškai, programų rašymas tokiu būdu pažeidžia patį PE formato standartą, tačiau šią praktiką neretai naudoja kenkėjiškų programų autoriai.

Norint visiškai išvengti nevalidaus PE formato problemos tenka taikyti perturbacijas, nekeičiančias originalių programų, o taikančias kitokius obfuskacijos metodus. Iš 1.2. poskyryje aptartų perturbacijų, tokias sąlygas atitinka tik 2 kompleksinės perturbacijos (1.2.3.) – *Stealmal* ir *Hollowmal*.

## 1.7. AE perkeliamumas

Perkeliamumas DI modeliuose dažniausiai suprantamas kaip žinių perkeliamumas (*angl. knowledge transferability*). Tačiau tiriant varžymosi principais pagrįstas atakas buvo pastebėtas ir AE perkeliamumas (*angl. adversarial sample transferability*). Tai reiškia, jog gebant sukurti AE, kuriuos neteisingai klasifikuoja vienas modelis, tikėtina, jog kitas (tos pačios paskirties) modelis taip pat klasifikuos AE neteisingai. Be to, nustatyta, jog AE perkeliamumo savybė galioja skirtingų architektūrų modeliams [DCB<sup>+</sup>21]. Šia savybe remiasi visos „juodos dėžės“ atvejams pritaikytos atakos.

## 1.8. AE aptikimo strategijos

### 1.8.1. Varžymosi principais pagrįstas pertreniravimas

Varžymosi principais pagrįstas pertreniravimas (*angl. adversarial retraining*) – tai papildomas ML modelio treniravimo etapas. Tarkime pradinis ištreniruotas ML modelis yra  $M_1$ . Tuomet Varžymosi principais pagrįstas pertreniravimas yra AE generavimas atakuojant  $M_1$  ir šio modelio papildomas treniravimas su sugeneruotais AE. Po treniravimo gauname  $M_2$  ( $M_1 \xrightarrow{AE} M_2$ ), kuris bus atsparesnis varžymosi principais pagrįstoms atakoms. Ši strategija nėra atspari „juodos dėžės“ atakoms [CAD<sup>+</sup>21].

### 1.8.2. Gradientų slėpimas

Gradientų slėpimas (*angl. gradient hiding*) – tai metodas, kai pasirenkama klasifikatoriaus architektūra, kurioje nefigūruoja gradientai, pavyzdžiui, nediferencijuojami modeliai, tokie kaip atsitiktinis miškas (*angl. random forest*). Dėl AE perkeliamumo (1.7.) ši strategija yra neveiksminga prieš „juodos dėžės“ atakas [CAD<sup>+</sup>21].

### 1.8.3. Kategorijų švelninimas

Kategorijų švelninimas (*angl. label smoothing*) – metodas, leidžiantis geriau klasifikuoti nežinomus duomenis ir taip apsisaugoti nuo varžymosi principais pagrįstų atakų. Kategorijų švelninimas reiškia tiksliai apibrėžtų kategorijų (*angl. hard labels*) pavertimą tikimybiniais vektoriais (*angl. soft labels*). Tuomet ML modelio išvestis (taip pat tikimybinis vektorius) gali būti palyginama su kategorijų vektoriais taikant norimas euristicas, pavyzdžiui, kosinusų panašumą (*angl. cosine similarity*). Dėl AE perkeliamumo (1.7.) ši strategija neveiksminga prieš „juodos dėžės“ atakas [CAD<sup>+</sup>21].

### 1.8.4. Perkeliamumo blokavimas

Perkeliamumo blokavimas (*angl. transferability blocking*) – tai varžymosi principais pagrįsto pertreniravimo (1.8.1.) praplėtimas, įtraukiant naują (*NULL*) kategoriją į galimas klasifikatoriaus išvestis. Ši strategija praplečia klasifikatoriaus treniravimą į šiuos etapus:

1. Įprastas klasifikatoriaus treniravimas.



2. *NULL* tikimybių funkcijos  $f$  skaičiavimas.  $p_{NULL}$  suprantama kaip varžymosi principais pagrįstos atakos tikimybė. *NULL* tikimybių ir jų skaičiavimo funkcijos  $f$  ryšys pateiktas 1-oje lygtyje.
3. Varžymosi principais pagrįstas pertreniravimas, įtraukiant tiek originalias įvestis, tiek perturbuotas ( $\delta X$ ).

Ši strategija sprendžia pagrindinį daugelio kitų strategijų trūkumą – AE perkeliamumą (1.7.), todėl yra pakankamai efektyvi [CAD<sup>+</sup>21].

$$p_{NULL} = f\left(\frac{\|\delta X\|_0}{\|X\|}\right), \quad (1)$$

$$\text{čia } \|\delta X\|_0 \sim U[1, N_{max}], \quad N_{max} = \min n \ni f\left(\frac{n}{\|X\|}\right) = 1$$

### 1.8.5. Bazės keitimas, transformacijos

Bazės keitimas, transformacijos (*angl. change of basis, transformations*) – tai strategija transformuojanti duomenis juos projektuojant kitoje koordinačių sistemoje. Pavyzdžiui, PCA metodas keičia duomenų koordinačių sistemos bazę taip, jog pirma komponentė turėtų didžiausią inerciją. Tam tikrais atvejais taikant šią strategiją gali būti prarandama informacija (jei keičiama į mažesnės dimensijos bazę), pavyzdžiui, naudojant *JPEG lossy compression* algoritmą nuotraukų transformacijai. Ši strategija padeda efektyviai apsaugoti nuo visų tipų varžymosi principais pagrįstų atakų [CAD<sup>+</sup>21].

## 1.9. LIME metodas

*LIME* (*angl. Local Interpretable Model-agnostic Explanations*) – tai lokalūs, interpretuojami ML modelių išvesčių paaiškinimai. ML modeliai dažnai turi būti sudėtingi (dėl to ir sunkiai, ar visiškai neinterpretuojami), nes jie aproksimuoja sudėtingus skirstinius visoje (globalioje) srityje. *LIME* metodo idėja yra aproksimuoti skirstinį taip pat, kaip ML modelis žymiai mažesnėje – **lokalioje** – srityje (*angl. locally faithful*). Tai leidžia *LIME* naudojamam aproksimacijos modeliui būti ženkliai paprastesniam, daugeliu atvejų – tiesiniam (pvz., tiesinė regresija). *LIME* veikimo principas yra vieno pavyzdžio (įvesties  $x_0$ ) perturbavimas taip sukuriant lokalių  $x_0$  artimų įvesčių aibę  $\hat{X}$ . Kiekvienas  $\hat{X}$  elementas pateikiamas originalaus modelio ( $M$ ) klasifikacijai ir taip gaunama aibė  $\hat{Y}$  ( $\hat{X} \xrightarrow{M} \hat{Y}$ ). Abi aibės naudojamos surogatinio (dažniausiai tiesinio) modelio  $\hat{M}$  mokymui. Tuomet **interpretuojamas paaiškinimas** yra surikiuotas pagal įtaką galutiniam  $\hat{M}$  sprendimui  $x_0$  komponentių (požymių) sąrašas (pilnas arba dalinis) [RSG16].

Svarbus *LIME* parametras yra branduolio plotis (*angl. kernel width*)  $\omega$ . *LIME* mokymosi etape kiekvienai perturbuotai įvesčiai  $\hat{x}_i \in \hat{X}$  priskiria svorį  $a_i \propto \exp\left(-\frac{D(x_0, \hat{x}_i)}{\omega^2}\right)$ . Taigi,  $\omega$  turi būti parenkamas atsižvelgiant į skalę, kurioje klasifikuojami klasteriai gali būti atskiriami tiesiškai.

*LIME* metodas yra paaiškinamo dirbtinio intelekto (XAI) pavyzdys. XAI gali būti naudojamas

kaip AE aptikimo strategija, jei paaiškinimai geba pastebimai atskirti AE nuo tikrų įvesčių (pavyzdžiui, klasifikacijos paaiškinimas statistiškai reikšmingai skiriasi nuo tikrų duomenų paaiškinimų).

### 1.9.1. Varžymosi principais pagrįstų atakų aptikimas IDS naudojant *LIME*

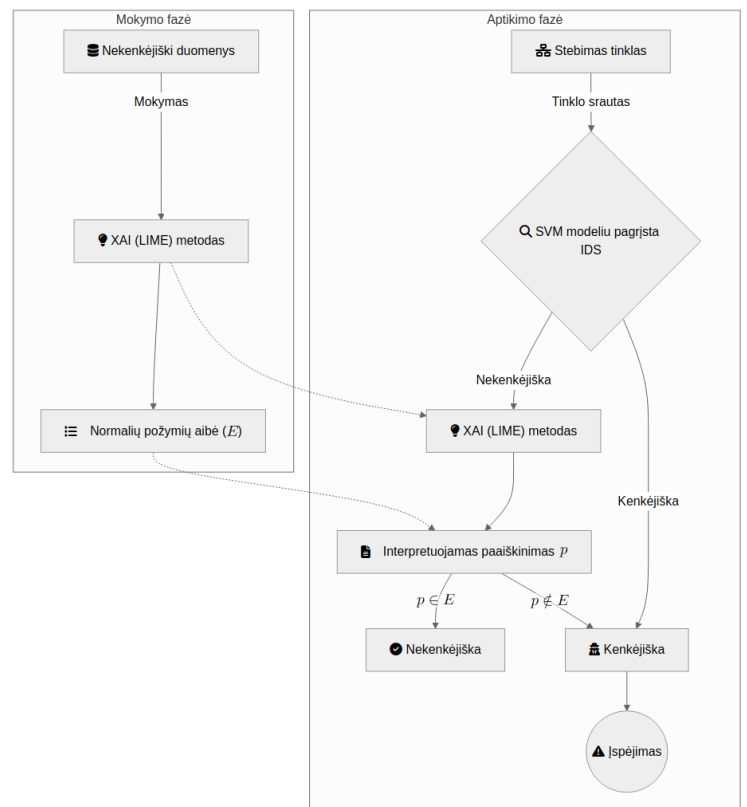
IDS (*angl. Intrusion Detection System*) – tai sistema, veikianti uždareme tinkle ir nuolat analizuojanti tinklo srautą. Vienas iš IDS įgyvendinimo būdų yra pasitelkti ML modelius, tad šios sistemos taip pat yra pažeidžiamos varžymosi principais pagrįstoms atakoms. Šiame kontekste Tcydenova ir kt. pritaikė *LIME* kaip AE aptikimo strategiją. Jų siūlomas metodas (žr. 2 pav.) AE aptikimui susideda iš dviejų pagrindinių dalių [TKL<sup>+</sup>21]:

1. Mokymo fazė.

- Mokomas ML modelis tinklo srauto klasifikavimui.
- Kiekvienam nekenkėjiškų mokymo duomenų pavyzdžiui pritaikomas *LIME* metodas – gaunama paaiškinimų aibė  $E$ , kur kiekvienas elementas yra  $n$ -matis vektorius, turintis  $n$  svarbiausių **interpretuojamų paaiškinimų**.
- $E$  laikoma normalių požymių aibe.

2. Aptikimo fazė.

- Išmokytas ML modelis klasifikuoja tinklo srautą.
- Jei ML modelis nustato, jog įvestis yra kenksminga – ji tokia ir laikoma.
- Jei ML modelis nustato, jog įvestis nėra kenksminga – taikomas *LIME* metodas ir gaunamas klasifikavimo paaiškinimas.
- Jei klasifikavimo paaiškinyje figūruoja bent vienas požymis, nepriklausantis  $E$  aibe – laikoma, jog įvestis yra kenksminga.
- Jei visi klasifikavimo paaiškinyje figūruojantys požymiai priklauso aibei  $E$  – laikoma, jog įvestis nekenksminga.



**2 pav.** *LIME* pritaikymas AE aptikimui IDS (adaptuota iš [TKL<sup>+</sup>21])

*LIME* metodo taikymas varžymosi principais pagrįstų atakų aptikimui yra artimas **perkeliamumo blokavimo** (1.8.4.) strategijai, tačiau šiuo atveju *LIME* veikia kaip atskiras nuo pagrindinio klasifikatoriaus komponentas, tad pagrindinis ML modelis lieka pažeidžiamas varžymosi principais pagrįstoms atakoms.

## 1.10. Dimensijų mažinimo metodai

### 1.10.1. Principinių komponentių analizė (PCA)

PCA – tai daugiamatės statistinės analizės metodas, naudojamas išskirti svarbiausią informaciją iš turimų duomenų. Svarbiausios informacijos apibrėžimui naudojama inercijos metrika. Inercija – tai dalis variacijos, kuri yra paaiškinama viena komponente. Iš esmės PCA yra bazės keitimo metodas (1.8.5.), kurio tikslas yra parinkti bazę, tenkinančią nelygybę  $\forall n : I_n \geq I_{n+1}$ , kur  $I_n$  –  $n$ -osios komponentės inercija [AW10]. Tai leidžia sumažinti naudojamų komponentių skaičių pasirenkant pakankamą inercijos vertę. Pavyzdžiui, jei pirmosios komponentės inercija yra 50%, antrosios – 25% ( $\rightarrow$  likusių komponentių inercija lygi 25%) ir pakankama inercijos vertė yra 70%, tuomet užtenka palikti pirmas dvi komponentes, kurių bendra inercija (*angl. cumulative inertia*) yra 75%.

### 1.10.2. Daugialypės korespondencijos analizė (MCA)

MCA (*angl. Multiple Correspondence Analysis*) – tai dar vienas daugiamatės statistinės analizės metodas, tik šis skirtas kategoriniams duomenims. Šio metodo veikimo principas pagrįstas standartinė korespondencijos analize (*angl. correspondence analysis*) iš kategorinių duomenų sukonstruotai indikatorių matricai. MCA gali būti laikomas PCA praplėtimu kategoriniams duomenims [AV07], tad ir dimensijų mažinimo procesas toks pat – pasirenkama pakankama inercijos vertė  $\varepsilon$  ir pasirenkami pirmi  $m$  stulpelių taip, kad  $\sum_{i=1}^m I_i \geq \varepsilon$

MCA metodas gali būti naudojamas kaip išmokomas statistinis modelis. Kadangi nauja bazė, į kurią transformuojami duomenys, yra glaudžiai susijusi su duomenų rinkinio statistikomis, galima naudoti mokymo duomenų aibę šios bazės nustatymui, o tolimesnius duomenis transformuoti kaip projekcijas į šią bazę. Toks MCA panaudojimas ir bus pasirenkamas tyrime (3.).

## 2. MCA ir *LIME* AE aptikimo metodų sintezė

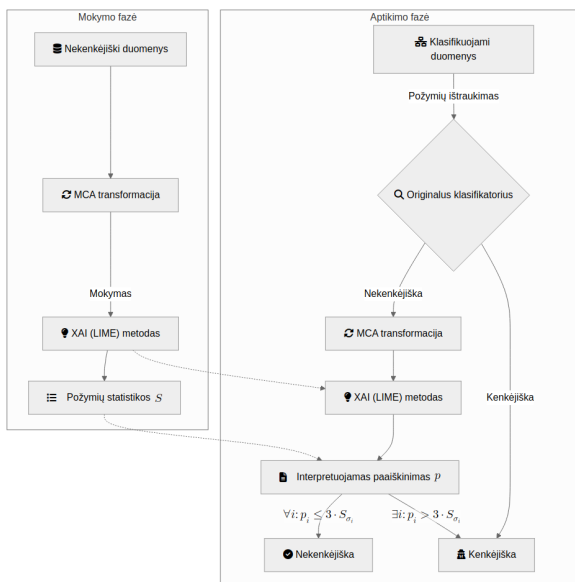
Autoriaus siūlomas AE aptikimo metodas yra apjungti MCA dimensijų mažinimo metodą bei *LIME* pritaikymą AE aptikimui su tam tikromis modifikacijomis (2.1.). Kadangi MCA ir *LIME* atitinka bazės keitimo (1.8.5.) ir perkeliavimo blokavimo (1.8.4.) AE aptikimo strategijas, kurios iš aptartų yra perspektyviausios – jų sintezė tikimasi gauti dar tikslesnį AE aptikimo metodą (žr. 3 pav.).

### 2.1. *LIME* pritaikymo AE aptikimui metodo modifikacijos

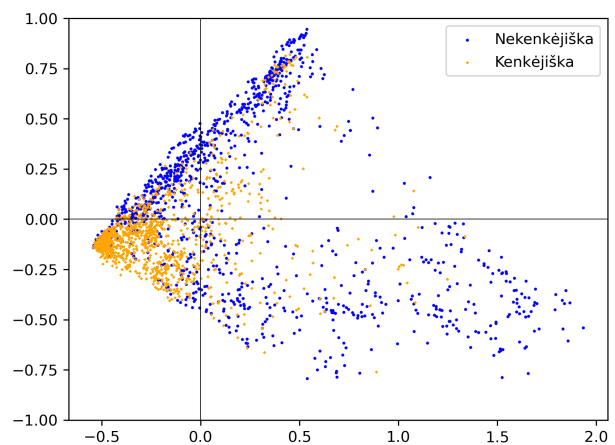
1.9.1. skyriuje siūlomas *LIME* pritaikymas remiasi svarbiausių (didžiausią įtaką ML modelio sprendimų priėmimui turinčių) požymių analize. Svarbiausi požymiai laikomi pirmieji 10 [TKL<sup>+</sup>21]. Nors autoriai neaprašo kaip pasirenkama tokia konstanta, akivaizdu, jog ji nėra tinkama visiems atvejams. Pavyzdžiui, turint žymiai daugiau požymių, ši konstanta gali būti per maža. Tokia problema ypač aktuali kai analizuojami kategoriniai požymiai, turintys daug kategorijų (dažniausiai koduojami kaip dvejetainiai vektoriai – *angl. one-hot encoding*). Dėl šios priežasties, autoriaus siūlymas yra „svarbiais“ laikyti visus požymius ir mokymo fazės pabaigoje apskaičiuoti šių požymių įtakos ML modelio sprendimo priėmimui statistikas (vidurkį ir standartinį nuokrypį). Tuomet aptikimo fazėje naudoti  $3\sigma$  taisyklę [Puk94] t. y. jei bent vieno požymio įtaka ML sprendimo priėmimui nukrypsta per 3 standartinius nuokrypius nuo duotojo požymio vidurkio – laikyti, jog analizuojama įvestis buvo obfusuota.

### 2.2. *LIME* branduolio pločio pasirinkimas

*LIME* branduolio plotį (1.9.) gali padėti nustatyti duomenų vizualizacijos ar kita duomenų analizė. Tai atlikti nėra sudėtinga, kai požymių vektoriaus dimensija nedidelė arba požymiai yra skaitiniai, tačiau turint kategorinius požymius ir atsižvelgiant į jų kodavimą dvejetainiais vektoriais, parinkti tinkamą *LIME* branduolio plotį gali būti sudėtinga. Taikant MCA transformaciją neretai užtenka branduolio pločio vertę parinkti pagal pirmų dviejų komponentų skalę, kadangi šios dvi komponentės apibūdina didžiausią inerciją (žr. 4 pav.).



**3 pav.** LIME ir MCA sintezės AE aptikimo metodo iliustracija



**4 pav.** Pirmųjų dviejų MCA transformacijos komponentių pavyzdys

### 3. Klasifikavimo tikslumo TODO tyrimas

#### 3.1. Tyrimo metodika

Visa tyrimo metodika pavaizduota 5-ame pav. Tyrimo tikslas yra nustatyti, ar *LIME* ir MCA metodų sintezė geba efektyviai aptikti varžymosi principais pagrįstas atakas ir ar šio metodo tikslumas didesnis nei Tcydenovos ir kt. pasiūlyto *LIME* pritaikymo varžymosi principais pagrįstų atakų aptikimui. Tyrimui pasirinktas *SLEIPNIR* [AHH<sup>+</sup>18] duomenų rinkinys, kuriame yra 34995 kenkėjiškų programų ir 19696 nekenkėjiškų programų pavyzdžių, užkoduotų 22761-mačiais dvejetainiais vektoriais. Iš šio rinkinio pasirinkta po 1500 unikalių kenkėjiškų ir nekenkėjiškų programų pavyzdžių paliekant pirmas 200 komponentų – taip gaunamas subalansuotas duomenų rinkinys. Šis rinkinys toliau skeliamas į mokymo ir testavimo aibes santykiu 4 : 1. Eksperimentams atlikti paruošiamos šios priemonės:

1. Testavimo duomenų aibė, iš kurios sudaromas trijų tipų programų požymių srautas: nekenkėjiškos, kenkėjiškos, kenkėjiškos-obfusuotos.
2. *MalGAN* [HT17] AE generatorius.
3. Kenkėjiškų programų detektorius (klasifikatorius).
4. Varžymosi principais pagrįstų atakų aptikimo komponentas
  - Naudojantis originalius požymius.
  - Naudojantis MCA transformuotus požymius (2.).

Su šiais įrankiais atliekami 4 eksperimentai:

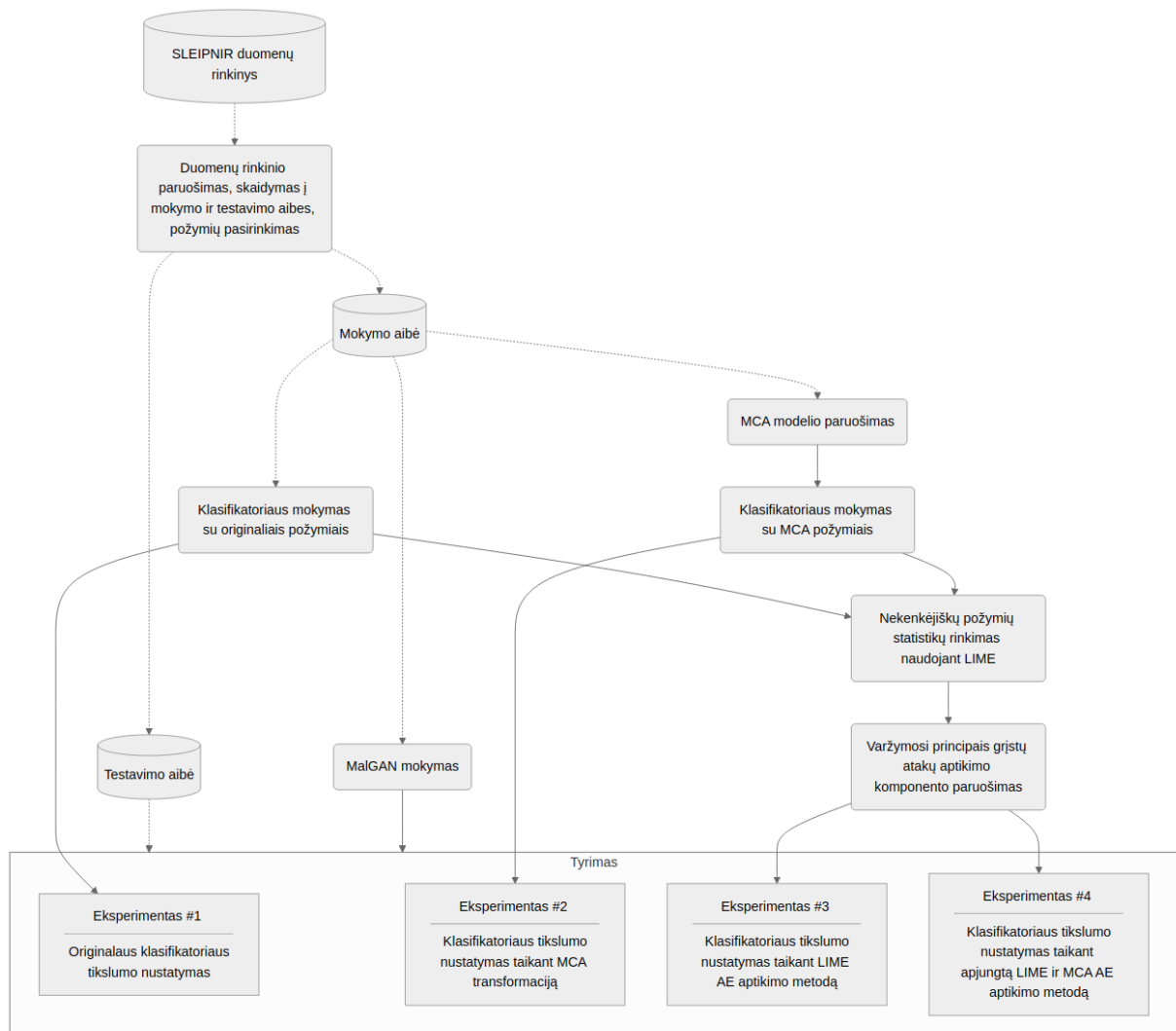
1. Bazinis atvejis – klasifikatoriaus be varžymosi principais pagrįstų atakų aptikimo metodų tikslumo nustatymas.
2. Klasifikatoriaus, naudojančio MCA transformuotus požymius, tikslumo nustatymas.
3. Klasifikatoriaus su *LIME* metodo pritaikymo varžymosi principais pagrįstų atakų aptikimui modifikacija tikslumo nustatymas.
4. Klasifikatoriaus su *LIME* ir MCA metodų sintezės varžymosi principais pagrįstų atakų aptikimui modifikacija tikslumo nustatymas.

#### 3.2. MCA komponentų pasirinkimas

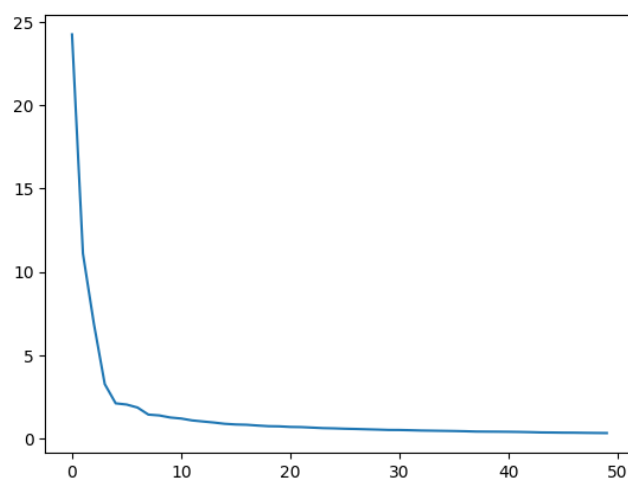
Komponentų pasirinkimas yra 5-ame pav. minimo MCA modelio paruošimo dalis. Komponentų pasirinkimui t. y. jų kiekio pasirinkimui nėra apibrėžto „teisingo“ metodo [AW10]. Dažniausiai siūlomi metodai yra tik didesnių už 1 tikrinių reikšmių pasirinkimas ir *alkūnės* (angl. *scree / elbow*) metodas. Šie metodai netiko, nes juos taikant paliktos MCA komponentės paaiškintų < 50%

visos inercijos: visos tikrinės reikšmės analizuojamuose mokymo duomenyse yra  $< 1$ , o alkūnės taške sukaupta inercija yra  $\sim 30\%$  (žr. 6 pav.). Taigi, pasirinkti 2 nestandartiniai MCA komponenčių kiekio kriterijai:

- sukaupta inercija  $\geq 75\%$ ,
- Išmokyto klasifikatoriaus tikslumas  $\hat{a}$  nemažesnis nei originalaus klasifikatoriaus tikslumas  $a$  5% intervale ( $\hat{a} \geq a - 5\%$ ), kai įvesčių srautas normalus (nėra obfusuotų programų požymių – AE) **TODO**.



5 pav. Tyrimo metodika



**6 pav.** *Alkūnės analizė MCA inercijai*



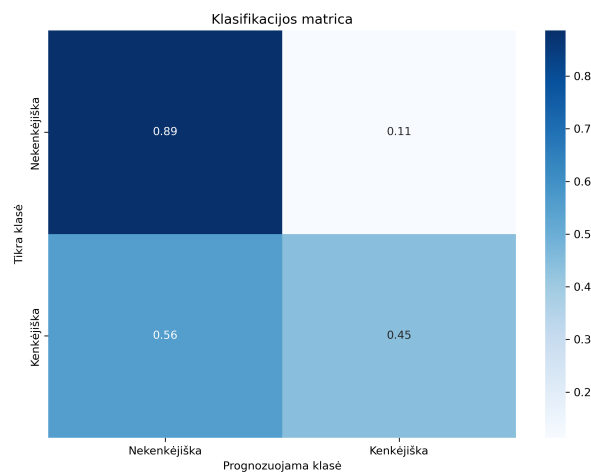
### 3.3. Eksperimentai

#### 3.3.1. Originalaus klasifikatoriaus tikslumo nustatymas

Kaip ir tolimesniuose eksperimentuose, testavimo duomenų aibė šiam eksperimentui susideda iš 300 kenkėjiškų ir 300 nekenkėjiškų programų požymių. Taip pat pridedami dar 300 obfuskuotų programų požymių (jie gaunami naudojant jau turimus kenkėjiškų programų požymius ir išmokytą *MalGan* modelį).

Kadangi originalus klasifikatorius negali diferencijuoti obfuskuotų programų, turima duomenų aibė nėra subalansuota – turime dvigubai daugiau duomenų, kuriuos klasifikatorius turėtų klasifikuoti kaip kenkėjiškus. Dėl to, klasifikacijos lentelėje (žr. 7 pav.) rodomas prognozuojamos klasės ir visų tai klasei priklausančių duomenų santykis (tokios matricos pagrindinė diagonalė nurodo klasės atkūrimo statistiką (*angl. recall*)).

Eksperimento rezultatai (klasifikavimo metrikos) pateikiami 8-oje lentelėje.



7 pav. Klasifikavimo matrica

8 lentelė. Originalaus klasifikatoriaus metrikos

Klasė / Svertinis vidurkis	Preciziškumas	Atkūrimas	F1
Nekenkėjiška	0,444	0,887	0,592
Kenkėjiška	0,887	0,445	0,593
Svertinis vidurkis	0,739	0,592	0,592

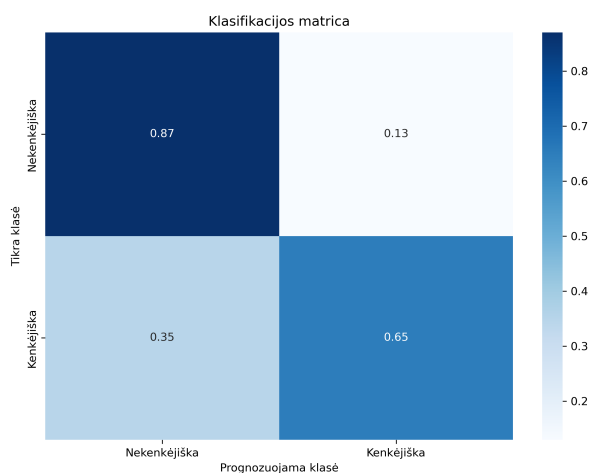
Tikslumas: 0,592

### 3.3.2. MCA transformacijos klasifikatoriaus tikslumo nustatymas

MCA klasifikatorius yra vienas iš standartinių klasifikatorių (šiuo atveju atsitiktinio miško (*angl. random forest*)), naudojantis pakeistos bazės požymius kaip įvestį. Šioje vietoje naudojamas klasifikatorius gali būti bet koks, tačiau turi būti suderinamas su MCA komponentų kiekiu reikalavimais (3.2.).

**TODO**

Eksperimento rezultatai (klasifikavimo metrikos) pateikiami 9-oje lentelėje.



**8 pav.** Klasifikavimo matrica

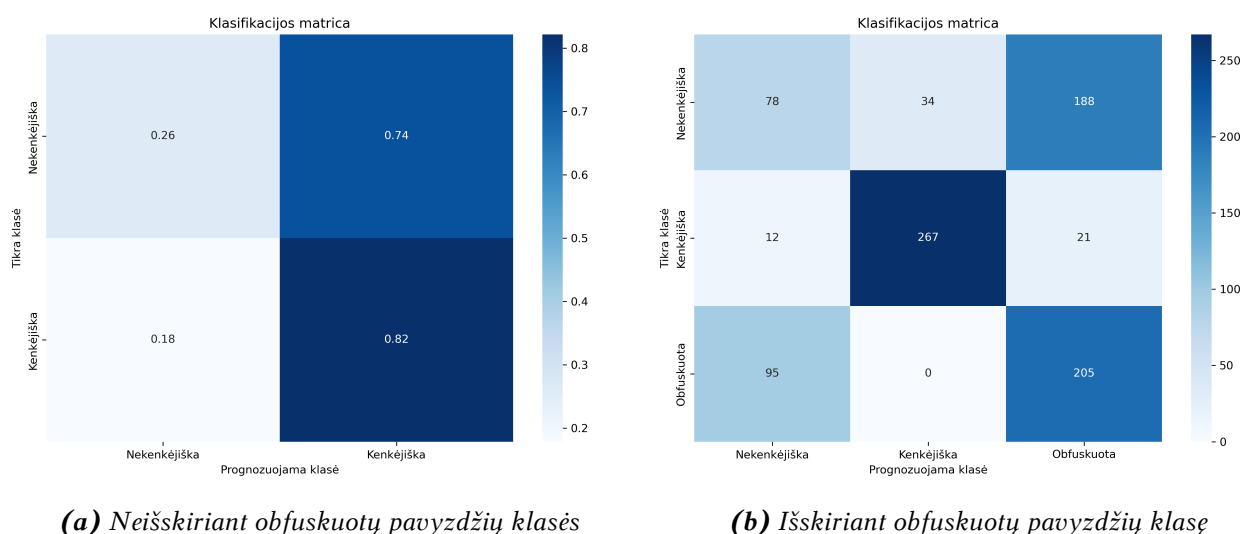
**9 lentelė.** MCA transformacijos klasifikatoriaus metrikos

Klasė / Svertinis vidurkis	Preciziškumas	Atkūrimas	F1
Nekenkėjiška	0,555	0,870	0,678
Kenkėjiška	0,909	0,652	0,759
Svertinis vidurkis	0,791	0,724	0,732
Tikslumas:			0,724

### 3.3.3. Modifikuoto *LIME* pritaikymo AE aptikimui tikslumo nustatymas

Šio eksperimento tikslas yra nustatyti modifikuoto *LIME* pritaikymo AE aptikimui (2.1.) tikslumą. *LIME* branduolio plotis paliekamas pagal numatytus nustatymus ( $\omega = 0,75 \cdot \sqrt{n}$ , čia  $n$  – požymių skaičius, taigi,  $\omega = 0,75 \cdot \sqrt{200} \approx 10,61$ ) [RSG16].

Kadangi šis metodas geba aptikti AE – obfusuotas programas – 9-ame pav. pateikiamos dvi klasifikacijos lentelės. Pirmoje (9a) – klasifikacijos rezultatai, kai obfusuota programa laikoma kenkėjiška. Antroje (9b) – išskiriama *obfusuota* klasė. Kadangi išskyrus šia klasę duomenų rinkinys tampa subalansuotas, klasifikavimo lentelėje pateikiami sveiki skaičiai, atitinkantys modelio prognozuojamų duomenų tai klasei kiekį.



9 pav. *LIME* pritaikymo AE aptikimui klasifikavimo lentelės

10 lentelė. TODO

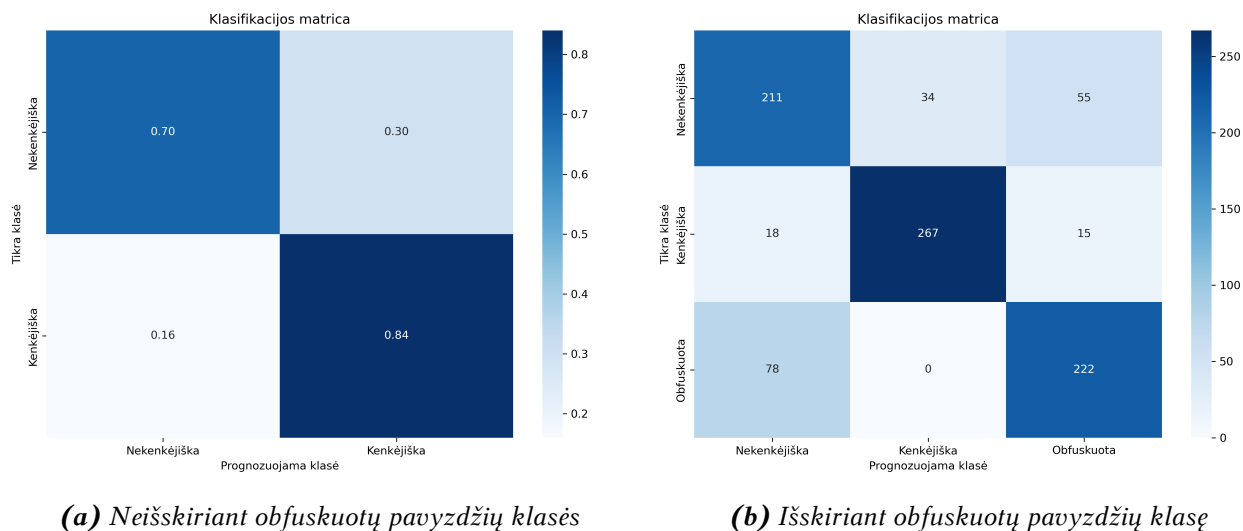
Klasė / Svertinis vidurkis	Preciziškumas	Atkūrimas	F1
Nekenkėjiška	0,422	0,260	0,322
Kenkėjiška	0,690	0,822	0,750
Svertinis vidurkis	0,600	0,634	0,607
Tikslumas:			0,634

11 lentelė. TODO

Klasė / Svertinis vidurkis	Preciziškumas	Atkūrimas	F1
Nekenkėjiška	0,422	0,260	0,322
Kenkėjiška	0,887	0,890	0,889
Obfusuota	0,495	0,683	0,574
Svertinis vidurkis	0,601	0,611	0,595
Tikslumas:			0,611

### 3.3.4. LIME ir MCA metodų sintezės tikslumo nustatymas

Šio eksperimento tikslas yra nustatyti autoriaus siūlomos modifikuoto *LIME* ir *MCA* metodų sintezės varžymosi principais pagrįstų atakų aptikimui tikslumą. Kaip ir praeitame eksperimente, 10-ame pav. pateikiamos dvi klasifikavimo lentelės – išskiriant ir neišskiriant *obfusuotą* klasę.



10 pav. LIME ir MCA metodų sintezės klasifikavimo lentelės

12 lentelė. TODO

Klasė / Svertinis vidurkis	Preciziškumas	Atkūrimas	F1
Nekenkėjiška	0,687	0,703	0,695
Kenkėjiška	0,850	0,840	0,845
Svertinis vidurkis	0,796	0,794	0,795
Tikslumas:			0,794

13 lentelė. TODO

Klasė / Svertinis vidurkis	Preciziškumas	Atkūrimas	F1
Nekenkėjiška	0,687	0,703	0,695
Kenkėjiška	0,887	0,890	0,889
Obfusuota	0,760	0,740	0,750
Svertinis vidurkis	0,778	0,778	0,778
Tikslumas:			0,778

## Literatūra ir šaltiniai

- [AHH<sup>+</sup>18] A. Al-Dujaili, A. Huang, E. Hemberg, U.-M. O'Reilly. *Adversarial Deep Learning for Robust Detection of Binary Encoded Malware*. 2018. <https://doi.org/10.48550/arXiv.1801.02950>. (Žiūrėta 2025-02-28).
- [AKF<sup>+</sup>18] H. S. Anderson, A. Kharkar, B. Filar, D. Evans, P. Roth. *Learning to Evade Static PE Machine Learning Malware Models via Reinforcement Learning*. 2018. <https://doi.org/10.48550/arXiv.1801.08917>. (Žiūrėta 2024-09-30).
- [AV07] H. Abdi, D. Valentin. „Multiple Correspondence Analysis“. Iš: *Encyclopedia of measurement and statistics* 2.4 (2007), puslapiai 651–657.
- [AW10] H. Abdi, L. J. Williams. „Principal Component Analysis“. Iš: *WIREs Computational Statistics* 2.4 (2010), puslapiai 433–459. ISSN: 1939-0068. <https://doi.org/10.1002/wics.101>. (Žiūrėta 2025-04-17).
- [CAD<sup>+</sup>21] A. Chakraborty, M. Alam, V. Dey, A. Chattopadhyay, D. Mukhopadhyay. „A Survey on Adversarial Attacks and Defences“. Iš: *CAAI Transactions on Intelligence Technology* 6.1 (2021), puslapiai 25–45. ISSN: 2468-2322. <https://doi.org/10.1049/cit2.12028>. (Žiūrėta 2025-04-07).
- [CDH<sup>+</sup>16] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, P. Abbeel. *InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets*. 2016. <https://doi.org/10.48550/arXiv.1606.03657>. (Žiūrėta 2024-10-07).
- [CSD19] R. L. Castro, C. Schmitt, G. Dreo. „AIMED: Evolving Malware with Genetic Programming to Evade Detection“. Iš: *2019 18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*. 2019, puslapiai 240–247. <https://doi.org/10.1109/TrustCom/BigDataSE.2019.00040>. (Žiūrėta 2024-09-23).
- [DBL<sup>+</sup>21] L. Demetrio, B. Biggio, G. Lagorio, F. Roli, A. Armando. „Functionality-Preserving Black-Box Optimization of Adversarial Windows Malware“. Iš: *IEEE Transactions on Information Forensics and Security* 16 (2021), puslapiai 3469–3478. ISSN: 1556-6021. <https://doi.org/10.1109/TIFS.2021.3082330>. (Žiūrėta 2024-10-14).
- [DCB<sup>+</sup>21] L. Demetrio, S. E. Coull, B. Biggio, G. Lagorio, A. Armando, F. Roli. „Adversarial EXEmples: A Survey and Experimental Evaluation of Practical Attacks on Machine Learning for Windows Malware Detection“. Iš: *ACM Trans. Priv. Secur.* 24.4 (2021), 27:1–27:31. ISSN: 2471-2566. <https://doi.org/10.1145/3473039>. (Žiūrėta 2024-09-30).

- [FWL<sup>+</sup>19] Z. Fang, J. Wang, B. Li, S. Wu, Y. Zhou, H. Huang. „Evading Anti-Malware Engines With Deep Reinforcement Learning“. Iš: *IEEE Access* 7 (2019), puslapiai 48867–48879. ISSN: 2169-3536. <https://doi.org/10.1109/ACCESS.2019.2908033>. (Žiūrėta 2024-09-18).
- [HT17] W. Hu, Y. Tan. *Generating Adversarial Malware Examples for Black-Box Attacks Based on GAN*. 2017. (Žiūrėta 2024-09-18).
- [YPT22] J. Yuste, E. G. Pardo, J. Tapiador. „Optimization of Code Caves in Malware Binaries to Evade Machine Learning Detectors“. Iš: *Computers & Security* 116 (2022), puslapis 102643. ISSN: 0167-4048. <https://doi.org/10.1016/j.cose.2022.102643>. (Žiūrėta 2024-10-07).
- [Puk94] F. Pukelsheim. „The Three Sigma Rule“. Iš: *The American Statistician* 48.2 (1994), puslapiai 88–91. ISSN: 0003-1305. <https://doi.org/10.1080/00031305.1994.10476030>. (Žiūrėta 2025-04-18).
- [RSG16] M. T. Ribeiro, S. Singh, C. Guestrin. „”Why Should I Trust You?”: Explaining the Predictions of Any Classifier“. Iš: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. San Francisco California USA: ACM, 2016, puslapiai 1135–1144. ISBN: 978-1-4503-4232-2. <https://doi.org/10.1145/2939672.2939778>. (Žiūrėta 2025-02-28).
- [RSR<sup>+</sup>18] I. Rosenberg, A. Shabtai, L. Rokach, Y. Elovici. „Generic Black-Box End-to-End Attack Against State of the Art API Call Based Malware Classifiers“. Iš: *Research in Attacks, Intrusions, and Defenses*. Sudarė M. Bailey, T. Holz, M. Stamatogiannakis, S. Ioannidis. Cham: Springer International Publishing, 2018, puslapiai 490–510. ISBN: 978-3-030-00470-5. [https://doi.org/10.1007/978-3-030-00470-5\\_23](https://doi.org/10.1007/978-3-030-00470-5_23).
- [SB15] J. Saxe, K. Berlin. „Deep Neural Network Based Malware Detection Using Two Dimensional Binary Program Features“. Iš: *2015 10th International Conference on Malicious and Unwanted Software (MALWARE)*. 2015, puslapiai 11–20. <https://doi.org/10.1109/MALWARE.2015.7413680>. (Žiūrėta 2024-11-04).
- [TKL<sup>+</sup>21] E. Tcydenova, T. W. Kim, C. Lee, J. H. Park. „Detection of Adversarial Attacks in AI-Based Intrusion Detection Systems Using Explainable AI“. Iš: *Human-centric Computing and Information Sciences* 11.0 (2021), puslapiai 1–1. <https://doi.org/10.22967/HCIS.2021.11.035>. (Žiūrėta 2025-02-28).
- [ZCY<sup>+</sup>24] F. Zhong, X. Cheng, D. Yu, B. Gong, S. Song, J. Yu. „MalFox: Camouflaged Adversarial Malware Example Generation Based on Conv-GANs Against Black-Box Detectors“. Iš: *IEEE Transactions on Computers* 73.4 (2024), puslapiai 980–993. ISSN: 1557-9956. <https://doi.org/10.1109/TC.2023.3236901>. (Žiūrėta 2024-09-15).

- [ZH<sup>+</sup>Z22] F. Zhong, P. Hu, G. Zhang, H. Li, X. Cheng. „Reinforcement Learning Based Adversarial Malware Example Generation against Black-Box Detectors“. Iš: *Computers & Security* 121 (2022), puslapis 102869. ISSN: 0167-4048. <https://doi.org/10.1016/j.cose.2022.102869>. (Žiūrėta 2024-09-14).
- [ZZ<sup>+</sup>Y22] E. Zhu, J. Zhang, J. Yan, K. Chen, C. Gao. „N-Gram MalGAN: Evading Machine Learning Detection via Feature n-Gram“. Iš: *Digital Communications and Networks* 8.4 (2022), puslapiai 485–491. ISSN: 2352-8648. <https://doi.org/10.1016/j.dcan.2021.11.007>. (Žiūrėta 2024-09-23).