**VILNIAUS UNIVERSITETAS**

**MATEMATIKOS IR INFORMATIKOS FAKULTETAS**

TAUTVALDAS GUMBIS

*Informatikos specialybė, II kursas, II grupė*

# UŽKLAUSŲ SISTEMA „GREMLIN“

**Darbo vadovas: doc. dr. Haroldas Giedra**

**Vilnius, 2023**

# Turinys

# Įvadas

Duomenų bazės neretai būna grafinio pavidalo. Duomenis tokiose duomenų bazėse dažniausiai suprantame per viršūnes bei ryšius – viršūnės yra kažkokie duomenys, o jos turi briaunas (ryšius), jungiančias jas su kitomis viršūnėmis. Kartais taip pat galime sakyti, kad viršūnės atvaizduoja esybę (lentelę), o viršūnių savybės (angl. properties) yra lentelės atributai (stulpeliai). Pavyzdžiui, viršūnę galėtume paimti kaip žmogų, ir jis gali turėti kokį nors ryšį, tokį kaip draugystė, su kitu žmogumi. Grafinės duomenų bazės taip pat leidžia lengvai pridėti naujas viršūnes ir briaunas, kas leidžia greitai prisitaikyti prie kintančių duomenų reikalavimų. Taip pat grafinės duomenų bazės turi gerą našumą, efektyvius perėjimus, algoritmus, ieškant informacijos duomenų bazėje, ir yra dažnai naudojamos aplikacijose, kuriose svarbu greitai atnaujinti informaciją realiu laiku.

Be jau išvardintų bruožų, grafinėms duomenų bazėms būdinga ir užklausų kalbos, specializuotos ir optimizuotos dirbti būtent su grafais. Dažnai naudojamos tokios kalbos yra „SPARQL“ – kalba skirta dirbti su RDF duomenų bazėmis, „Cypher“ – deklaratyvi grafų užklausų sistema, suprojektuota „Neo4j“ grafinei duomenų bazei, „GQL“ – standartizuota grafų užklausų kalba ir “Gremlin“ – grafo užklausų bei perėjimų sistema. Panagrinėsime pastarąją užklausų sistemą. “Gremlin“ svarbi savybė yra ta, kad ji nėra pritaikyta būtent kokiai nors specifinei grafinei duomenų bazei, o yra dalis „Apache TinkerPop“ karkaso. Tai reiškia, kad „Gremlin“ užklausos gali būti naudojamos bet kokioje duomenų bazėje, kuri palaiko “TinkerPop“, suteikiant galimybę tomis pačiomis užklausomis veikti skirtingose grafų duomenų bazėse.

## Grafo kūrimo taisyklės, sintaksė

Grafo struktūra yra topologiškai formuojama aiškių nuorodų tarp viršūnių, briaunų bei savybių. Su „Gremlin“ ir „Apache TinkerPop“ karkasu galime dirbti su duomenų bazėmis bei vykdyti įvairias operacijas susijusias su grafais. Pavyzdžiui, galime sukurti grafą. Norint sukurti viršūnę, galime pridėti „addV“ žingsnelį - g.addV(label). Realus pavyzdys su žmogumi galėtų būti toks:

*g.addV(label).property(raktas1, reikšmė1).....*

*g.addV('žmogus').property('vardas', 'Lukas').property('metai', 34) – Sukūrėme grafe viršūnę žmogus, kuris turi kaip savybes vardą – Lukas bei metus – 34. Taip pat turi žmogaus etiketę.*

Norint pridėti briauną tarp dviejų viršūnių, naudojama „addE“ funkcija.

*addE(label).from(šaltinis).to(tikslas)*

Pavyzdžiui kompanija kuria programą, tokį ryšį tarp dviejų viršūnių galėtume pridėti:
*addE('kuria').from('kompanija').to('programa')*

V() funkcija grąžins mums visas viršūnes grafe, o E() visas briaunas. Galime grąžinti viršūnę ir pagal indeksą g.V(1) mums grąžintų v[1]. Taip pat galime patikslinti paiešką su etiketėmis. Pavyzdžiui, g.V().hasLabel("žmogus", "vyras") ras visas viršūnes turinčias žmogaus ir vyro etiketes. Arba galime naudoti g.V().has('raktas1', 'reikšmė1'), jeigu naudojame savybes. Toks sakinys kaip g.V().has('vardas', 'Lukas') mums parinks visas viršūnes, turinčias savybes, pavadintas 'vardas' bei 'Lukas'.

## Klausimų sudarymo sintaksė

„Apache TinkerPop“ leis mums naudoti “Gremlin Query Language“, su kuria galėsime pasiekti atsakymus į sąvokų grafe iškeliamus klausimus. “Gremlin“ vartoja traversal-based būdą grafo užklausoms. Užklausos sudaromos žingsniais, o žingsniai gali būti tokie kaip filtravimas, rikiavimas, transformavimas, agregavimas, jungimas grafo elementų. “Gremlin“ užklausa prasideda nuo tam tikros viršūnės ar viršūnių aibės, pradinio taško. Tarkime, ‚g‘ yra mūsų grafo šaltinis. Panaudoję funkciją ‘V()‘ , pasiekiame visas grafo viršūnes. Yra skirtingos operacijos, nusakančios, kaip bus vykdomi perėjimai grafe, dažnu atveju tokios operacijos yra:

- ‘out()‘: Keliautume nuo esamos viršūnes į išeinančias viršūnes;
- ‘in() ‘: Keliauti nuo esamos viršūnės į įeinančias viršūnes;
- ‘both()‘: Keliauti nuo esamos viršūnės į išeinančias ir įeinančias viršūnes.

Taip pat galime užklausai pridėti filtravimo bei atitikimo funkcijas. Filtruoti galime su tokiomis funkcijomis kaip ‘has(raktas, reikšmė)‘ – filtruoja viršūnes pagal nurodytą raktą ir reikšmę ir ‘hasLabel(etiketė)‘ – filtruoja viršūnes pagal etiketę. ‘Where()‘ funkcija leis taip pat filtruoti pagal savybes ar reikšmes gretimų viršūnių ar ryšių. ‚Filter()‘ funkcija skirta pritaikyti filtravimo logiką su “Gremlin“ predikatais. Taip pat galima naudoti loginius operatorius ‘and()‘ ar ‘or()‘ , siekiant pridėti daugiau funkcijų. “Gremlin“ taip pat turi funkciją neigimui ‘not()‘ ir paprastam filtravimui funkciją ‘is()‘. Pavyzdžiui, norint rasti viršūnes, kurioje nusakoma kokia nors lytis. g.V().hasLabel(‘žmogus‘).is(‘lytis‘, ‘moteris‘) grąžins mums viršūnių grafe viršūnes, kurios turi etiketę ‘žmogus‘ bei pagal savybės raktą ‘lytis‘ ieškos reikšmės, lygios reikšmei ‘moteris‘.

“Gremlin“ užklausų sistema taip pat palaiko agregatinius ir transformacinius žingsnius kaip ‘count()‘, ‘group()‘, ‘valueMap()‘, ‘sum()‘, ‘max()‘ duomenų apibendrinimui. Pavyzdžiui, galėtume surasti, kiek grafe yra viršūnių, turinčių etiketę žmogus paprasčiausiai pridedant ‘count()‘ funkciją prie mūsų užklausos.

*g.V().hasLabel(‘žmogus‘).count()*

Arba galime pritaikyti viršūnėms grupavimą pagal jų savybes

*g.V().group().by(‘metai‘) // Sugrupuos viršūnes pagal savybę ‘metai‘*

Taip pat vertėtų paminėti ir funkciją ‘fold()‘ ‘unfold()‘, šios funkcijos leidžia mums sudėti viršūnes į sąrašą ar jį išskaidyti.

*g.V().hasLabel(‘žmogus‘).values(‘metai‘).fold() // Agreguoja ‘metai‘ savybės reikšmes viršūnių, su etiketę ‘žmogus‘, į sąrašą.*

Galiausiai, galime pritaikyti ir rikiavimo bei tam tikras limito funkcijas mūsų užklausose. Dažnai ‘order().by(raktas, tvarka)‘ bei ‘limit(kiekis)‘ funkcijos yra naudojamos kartu užklausose. ‘order()‘ funkcija gali rikiuoti elementus pagal vieną ar daugiau savybių didėjimo ar mažėjimo tvarka.

*order().by(savybėsRaktas, asc/desc) // Vietoj asc/desc galime taip pat naudoti ir incr/decr raktažodžius.*

Pavyzdžiui, jeigu mūsų klausimas būtų, kaip surikiuoti žmones pagal amžių mažėjimo tvarka, galėtume parašyti štai tokią užklausą:

*g.V().hasLabel(‘žmogus‘).order().by(‘metai‘, decr)*

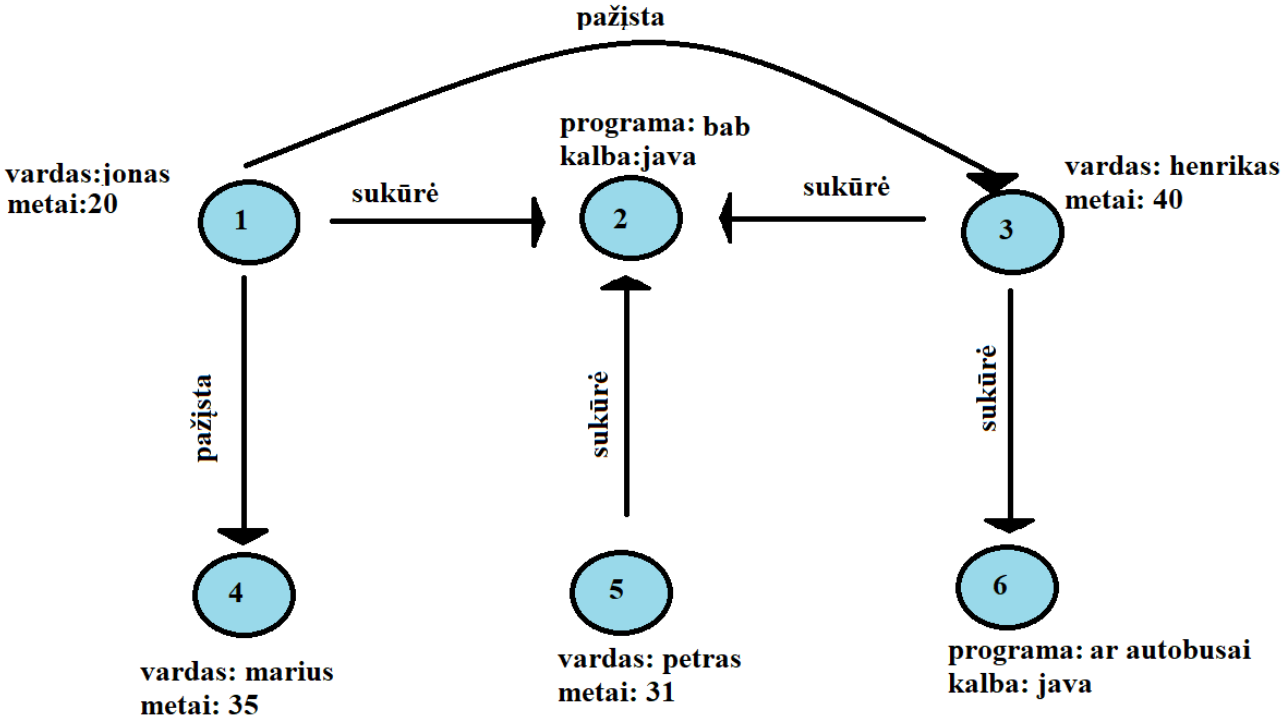
Ir štai čia, jeigu turėtume didelį sąraša žmonių, mes galėtume pridėti ‘limit(kiekis)‘ funkciją, kad į mūsų klausimą rezultatas būtų mažesnis.

*g.V().hasLabel(‘žmogus‘).order().by(‘metai‘, decr).limit(10) // Grąžins 10 seniausių žmonių*

## Algoritmas, kurio pradiniai duomenys yra sąvokų grafas ir klausimas, o išvesties duomenys – atsakymas į klausimą

“Gremlin“ grafo perėjimų algoritmas yra toks: “Gremlin“ užklausa prasideda nuo viršūnių aibės ar viršūnės tam, kad turėtume pradinį tašką perejimams pradėti. “Gremlin“ leidžia vykdyti kelis perėjimo žingsnius keliauti grafu, ir kiekvienas žingsnis filtruoja ar transformuoja esamą viršūnių aibę siaurinant rezultatų aibę. Taip pat yra svarbu kokia tvarka bus vykdomi perėjimai, ją nusako jau apžvelgtos operacijos „out“, „in“ ir kt. “Gremlin“ užklausa užbaigiama terminuojamu žingsneliu, kuris grąžina norima klausimo rezultatą – kokį nors viršūnių sąrašą, elementą ar tam tikrų reikšmių sumą.

Tarkime, turime grafą, kuriame yra žmonės bei programos. Žmogus gali sukurti programą, bet natūralu, jog programa negali sukurti žmogaus, todėl briaunos į programas gali būti tik įeinančios.




Paveikslas nr. 1 “TinkerPop“ Modern.

Tarkime, norime gauti rezultatą sąrašą vardų, kuriuos Jonas pažįsta. Tuomet, inicializave mūsų pradinę viršūnę grafe, žinome, kad iš pirmos viršūnės turėsime „išeiti“ ir pagal ryšį „pažįsta“ rasti viršūnes. Tai užrašyti galėtume taip:

*g.V(jonas).out(‘pažįsta‘)*

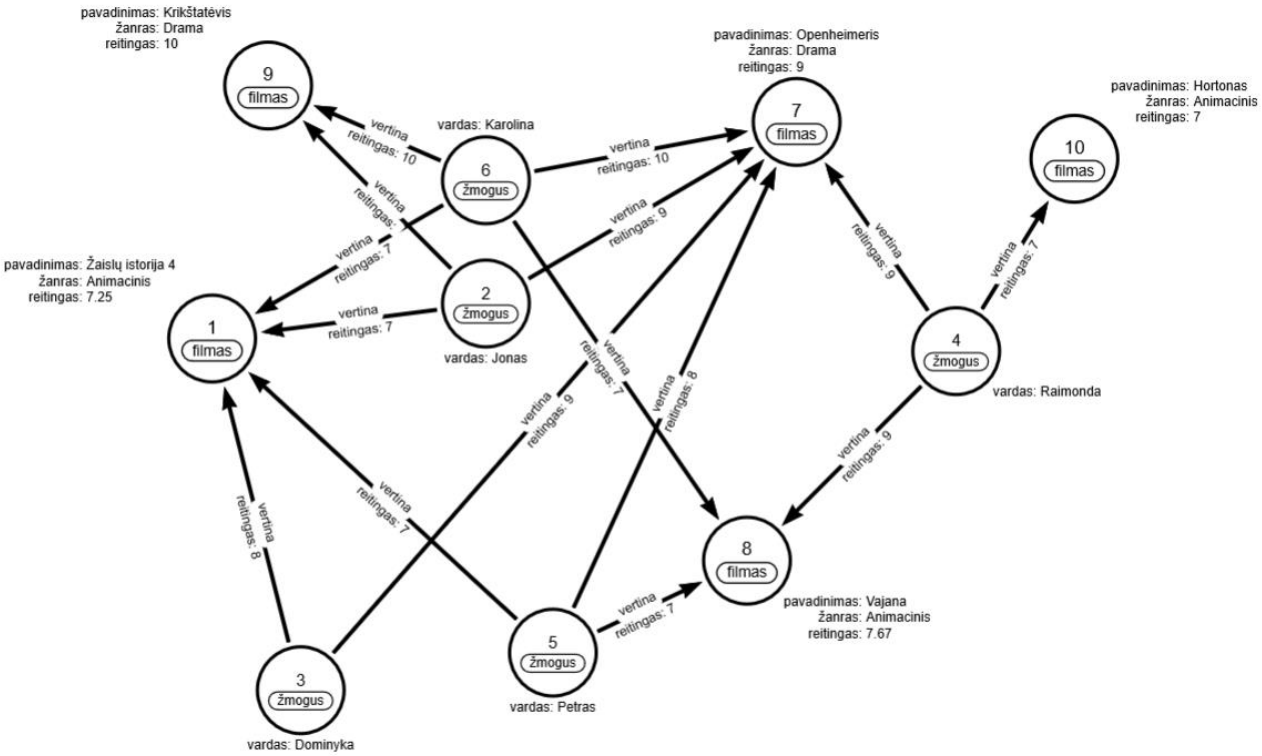
Tačiau toks užrašas grąžintu mums kaip atsakymą „v[3], v[4]“.

Taip yra todėl, kadangi nenurodėme specifines funkcijos „values“ kuris leistų išspausdinti viršūnių reikšmes. Pridėję šią funkciją, turime:

*g.V(jonas).out('pažįsta').values('vardas')*

Tokia užklausa mūsų grafe jau išspausdintų visus Jono pažįstamų žmonių vardus, taigi atsakymas būtų: „henrikas, marius“.

Panagrinėkime šiek tiek platesnį grafą:




Paveikslas Nr. 2 Filmų ivertinimai.

Tokia galėtų būti grafinio pavidalo duomenų bazė, sauganti įvairiausių žmonių įvertinimus skirtingiems filmams. Kaip matome, net ir naudojant nedaug ryšių ar vieną ryšį (šiuo atveju „vertina“ su tam tikru svoriu, kuris nurodo filmo reitingą) didėjant duomenų kiekiui skaityti informacija darosi sunku, ir todėl mums yra reikalingos užklausų sistemos, kalbos, galinčios surinkti duomenis, atsakymus, į mūsų norimus klausimus.

Pavyzdžiui, galime rasti, kokie trys filmai turi aukščiausią reitingą.

Panagrinėkime, kaip tokią užklausą turėtume užrašyti. Žinant, kad norime filmų, galime iš karto suteikti sąlygą hasLabel(‘etiketė‘), kad nereiktų tikrinti kitų duomenų kaip „žmonės“.

g.V().hasLabel(‘filmas‘) užklausa pateiktame pavyzdyje iš karto mūsų nagrinėjamus duomenis sumažina perpus – kadangi turime 5 viršūnes, atvaizduojančias žmones bei 5 viršūnes, atvaizduojančius filmus. Tada galime pridėti rikiavimo funkciją order().by(raktas, tvarka), ir pasirinkus mažėjimo tvarka, gautume kaip atsakymą viršūnių sąrašą: v[9], v[7], v[8], v[1], v[10]. Bet tai vis dar nėra mūsų norimas atsakymas, kadangi norime filmų pavadinimą, o jeigu duomenų būtų labai daug, tada grąžinamas viršūnių sąrašas taip pat galėtų būti be galo ilgas. Taigi, prie užklausos pridedame dar dvi jau mums žinomas funkcijas - limit(n) bei values(raktas, ..) ir turime mums reikalingą užklausą:

g.V().hasLabel(‘filmas‘).order().by(‘reitingas‘, desc).limit(3).values(‘pavadinimas‘, ‘reitingas‘)

Toks “Gremlin“ užklausos rezultatas mums grafinėje duomenų bazėje grąžintų trijų filmų, su aukščiausiais reitingais sąrašą. Šiuo atveju:

- Pavadinimas: Krikštatėvis, reitingas: 10
- Pavadinimas: Openheimeris, reitingas: 9
- Pavadinimas: Vajana, reitingas: 7.67

Dar vienas klausimas, kurį galėtume panagrinėti, yra kokie žmonės peržiūrėjo tam tikrą filmo žanrą, pavyzdžiui, dramą. Tokią užklausą pradėtume panašiai kaip ir praeitą, susiaurinant duomenis su ‚hasLabel‘ etikete, tik šį kartą mes ieškosime žmonių vardų, todėl etiketė bus žmogus. Turime g.V().hasLabel(‘žmogus‘). Šį kartą ieškosime iš viršūnės išeinančių ryšių, kadangi mūsų duomenų bazėje žmogus, peržiūrėjęs filmą, yra toks, kuris jį įvertino. Taigi galime pridėti sąlygą ‘where‘, jai pridėti sąlygas ‘out‘, kadangi žmogus turi ryšį, ‘vertina‘ filmą, ir specifikuoti filmo žanrą su ‘has‘ sąlyga. Taigi turėtume taip atrodančią užklausą:

*g.V().hasLabel(‘žmogus‘).where(out(‘vertina‘).has(‘žanras‘, ‘Drama‘)).*

Vėlgi, tokia užklausa mums dar tik grąžintų viršūnių sąrašą su indeksais, taigi pridėję galutinį žingsnelį values(‘vardas‘), patikslintume, jog norime žmonių vardų sąrašo ir gautume atsakymą, jog Karolina, Dominyka, Raimonda, Jonas bei Petras yra peržiūrėję bent vieną dramos filmą. Pažvelgę į grafą galime įsitikinti, jog taip ir yra, kadangi visi žmonės turi ryšį su kokia nors drama.

Paskutiniu atveju padarykime sunkesnę užklausą, ieškokime filmų, turinčių aukščiausius įvertinimus, kurių tam tikras žmogus dar nėra matęs. Pradėkime susiaurinant paieška iki žiūrovo viršūnės, nurodant jo vardą.

*g.V().has(‘vardas‘, ‘Dominyka‘).*

Tada pridėkime užklausą ieškoti visų viršūnių, turinčių etiketę filmas.

*g.V().has(‘vardas‘, ‘Dominyka‘).V().hasLabel(‘filmas‘).*

Toliau ieškome pagal įeinančią briauną ‚vertina‘, jog viršūnės, kuri turi tą įeinantį ryšį, sutaptų su mūsų pradine viršūne. Pridėkime neigimą pradžioje, jog rastume dar neįvertintus vartotojo filmus.

*g.V().has(‘vardas‘, ‘Dominyka‘).V().hasLabel(‘filmas‘).not(where(inE('vertina').outV().has('vardas', 'Dominyka'))).*

Galime pridėti rikiavimą pagal filmo reitingą mažėjančia tvarka.

*g.V().has(‘vardas‘, ‘Dominyka‘).V().hasLabel(‘filmas‘).not(where(inE('vertina').outV().has('vardas', 'Dominyka'))). order().by('reitingas', desc)*

Jeigu mūsų duomenų bazė būtų didesnė, galime pridėti limitą.

*g.V().has(‘vardas‘, ‘Dominyka‘).V().hasLabel(‘filmas‘).not(where(inE('vertina').outV().has('vardas', 'Dominyka'))). order().by('reitingas', desc).limit(5)*

Galiausiai, turime pridėti funkciją, kuri leistų mums išspausdinti ne viršūnių indeksus, o norimas reikšmes, šiuo atveju, filmų pavadinimus bei reitingus. Turime galutinę užklausą:

*g.V().has(‘vardas‘, ‘Dominyka‘).V().hasLabel(‘filmas‘).not(where(inE('vertina').outV().has('vardas', 'Dominyka'))). order().by('reitingas', desc). limit(5). values('pavadinimas', 'reitingas')*

Ši užklausa mums grąžintų kaip rezultatą, sąrašą: {10 Krikštatėvis, 7,67 Vajana, 7 Hortonas}.

Šis sąrašas, kaip mes ir norėjome, grąžina geriausius nematytų filmų pavadinimus bei reitingus.

# Apibendrinimas ir išvados

Taigi, „“Gremlin““ užklausų sistema yra gana lanksti, suteikianti galimybę užrašyti įvairiausių sunkumų užklausas ieškant atsakymų į klausimus.

- „Gremlin“ užklausų sistema suteikia galimybes sukurti grafą, viršūnes ir jų etiketes, savybes ir briaunas (ryšius), kurios taip pat gali turėti savo tam tikrus bruožus;
- „Gremlin“ kalba galime užrašyti kuo tikslesnes užklausas dėl įvairių filtravimo, rikiavimo, agregatinių funkcijų bei siekiant rezultato galime patikslinti atsakymo tipą;
- Algoritmus galime rašyti palaipsniui, su kiekvienu žingsneliu (funkcija) priartėjant prie klausimo sprendimo.

Norint išrinkti duomenis grafinėse duomenų bazėse su daug duomenų, galime pasitelkti būtent „Gremlin“ užklausų kalbą dėl jos universalumo, paprastos sintaksės bei puikios dokumentacijos.

# Literatūra ir šaltiniai

[1] D. Kuppitz, „Gremlin Cheat Sheet 101,“ [Tinkle]. Available: https://dkuppitz.github.io/Gremlin-cheat-sheet/101.html. [Kreiptasi 30 7 2023].

[2] „Apache TinkerPop“, „“TinkerPop“ Documentation,“ [Tinkle]. Available: https://TinkerPop.apache.org/docs/current/reference/#graph-computing. [Kreiptasi 30 7 2023].

[3] „Apache TinkerPop“, „Apache TinkerPop: Gremlin,“ [Tinkle]. Available: https://TinkerPop.apache.org/Gremlin.html. [Kreiptasi 30 7 2023].