



ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

ÓBUDAI EGYETEM
NEUMANN JÁNOS INFORMATIKAI KAR
TUDOMÁNYOS DIÁKKÖRI DOLGOZAT

GRÁF NEURÁLIS HÁLÓZATOK VIZSGÁLATA ELOSZTOTT KÖRNYEZETBEN

Szerző(k):

Póra Krisztián

mérnökinformatikus MSc. szak, II. évf.

Konzulens(ek):

Dr. habil. Lovas Róbert

egyetemi docens

Farkas Attila

egyetemi tanársegéd

Budapest, 2024. május 24.

Absztrakt

Az elmúlt évtizedben jelentősen megnőtt az igény olyan az Euklideszi téren kívül eső adatokon történő mély gépi tanulásra, mint például a gráfok. Erre a célra dedikálva úgynevezett gráf neurális hálózatok kerültek kifejelesztésre, amelyek hatékonyan képesek gráf alapú gépi tanulási feladatok megoldására. Ezen modellek tanítása azonban hagyományos társaikéhoz hasonlóan egy rendkívül idő- és erőforrásigényes folyamat, amelyet többek között elosztott környezetben történő végrehajtással gyorsíthatunk fel. A dolgozat célja a gráf neurális hálózatok elosztott tanítását támogató keretrendszerek vizsgálata, illetve egy kiválasztott keretrendszer teljesítményének javítása volt. Az elosztott végrehajtás átfogó vizsgálata után a számítási állomások közötti adatátvitel optimalizálását tűztem ki célul, amelynek érdekében egy partíció tudatos csúcs gyorsítótárat dolgoztam ki. A PANCA-nak keresztelt megoldás az azt alkalmazó állomások bejövő hálózati forgalmát több mint 85%-al csökkentette, és ezáltal egyes esetekben az elosztott tanítási klaszter összesített forgalmát kevesebb mint felére. A gyorsítótár a tanított modell pontosságát nem befolyásolja, azonban három és négy gépes elosztott konfiguráció esetén közel 15% csökkenést ért a tanítás teljes időtartamában.

Abstract

In the past decade, there has been a significant increase in demand for deep learning on data beyond Euclidean spaces, such as graphs. To address this need, dedicated graph neural networks have been developed, which are capable of effectively solving graph-based machine learning tasks. However, training these models remains an extremely time- and resource-intensive process, similar to their traditional counterparts. This process can be accelerated by execution in a distributed environment. The aim of this thesis was to investigate frameworks supporting the distributed training of graph neural networks and to enhance the performance of a selected framework. Following a comprehensive examination of distributed execution, I focused on optimizing data transfer between compute nodes by developing a partition-aware node cache. The solution, named PANCA, was able to reduce the incoming network traffic of workstations that utilized it by more than 85%, which in some cases resulted in the total traffic of the distributed training cluster being reduced by more than half. The cache does not affect the accuracy of the trained model, however, it has achieved a nearly 15% decrease in the total training time for three and four machine distributed configurations.

Tartalomjegyzék

1. Bevezető	3
2. Gráf Neurális Hálózatok	4
2.1. A gráfelmélet alapvető koncepciói	4
2.2. Neurális hálózatok	5
2.3. Motivációk a gráf alapú gépi tanulásra	7
2.4. Gráf neurális hálózatok alapjai	8
2.4.1. Műveletek	8
2.4.2. Felépítés	9
2.5. Gráf neurális hálózatok típusai	10
2.6. Gráf neurális hálózatok alkalmazása	11
2.6.1. Gráf adatstruktúrán végrehajtandó feladattípusok	11
2.6.2. Alkalmazási és kutatási területek	12
2.7. Gráf neurális hálózatok tanítása	13
2.7.1. Elosztott tanítás	14
2.7.2. Felhő alapú tanítási környezet	15
3. Gráf Neurális Hálózatok Eszközkészlete	16
3.1. Elosztott tanítást támogató keretrendszerek	16
3.1.1. PyTorch Geometric	17
3.1.2. Deep Graph Library	18
3.1.3. Roc	19
3.1.4. P ³	20
3.2. Teljesítménymérési keretrendszerek	21
3.2.1. Open Graph Benchmark	21
3.2.2. GNNMark	21
4. Mérési Környezet Megvalósítása	23
4.1. Eszközkészlet meghatározása	23
4.2. Elosztott tanítási architektúra	24
4.3. Elosztott tanítás paraméterei	28
4.3.1. Adathalmazok	28
4.3.2. Tanítási kód	31
4.3.3. Modell	35
4.3.4. Tanítási paraméterek	37
4.4. Teljesítménymérés automatizálása	39
5. Kiinduló Állapot Kiértékelése	42
5.1. Mérési eredmények	42

5.2. Kiinduló állapot előnyei és korlátai	46
5.3. Csúcs jellemző és címke átvitel elosztott környezetben.....	46
6. Partíció Tudatos Csúcs Gyorsítótárazás (PANCA)	49
6.1. Implementáció.....	50
6.2. Tesztelés és értékelés.....	52
6.3. Kapcsolódó munkák	55
7. Továbbfejlesztési lehetőségek és további kutatási irányok	57
8. Összefoglalás.....	58
9. Summary	59
Ábrajegyzék	60
Irodalomjegyzék.....	61

1. Bevezető

Napjainkra a mesterséges intelligencia és annak ágazatai, úgymint a mély gépi tanulás kiforrott módszertanná és életünk szerves részévé váltak. Megjelentek okos készülékeinkben, járműveinkben, és számos népszerű internetes szolgáltatásban. Mindemellett széleskörű felhasználásnak örvendenek az iparban, és aktív kutatási közösségeknek hála az akadémiai szférában is.

Az idő előrehaladtával a terület elsőrendű eszközeivé váltak a neurális hálózatok, amelyek az emberi agy és annak neuronjait szimuláló számítási modellek. A mély neurális hálózatok megjelenése és a felhasználható erőforrások fejlődése lehetővé tette, hogy óriási mennyiségű adatot dolgozzunk fel és rendkívül összetett problémákat oldjunk meg gépi tanulás segítségével. A különböző típusú neurális hálózatok forradalmasították olyan problémakörök megoldását, mint a gépi látás, a beszéd felismerés, és az intelligens becslés és ajánlás.

Hagyományosan a gépi tanulási alkalmazások az Euklideszi térbe tartozó, reguláris adatokra fókuszáltak, azonban az elmúlt évtizedben egyre nagyobb igény jelent meg az úgynevezett geometriai mély gépi tanulásra. Ennek keretein belül olyan adatokat dolgozunk fel, amelyek az Euklideszi téren kívül esnek. A módszertan elsőszámú alanyai a gráfok, melyek segítségével objektumok és azok kapcsolatainak bonyolult hálózatát, például közösségi és szenzor hálózatokat, vásárlók és termékek közötti kapcsolatokat vagy biológiai és kémiai struktúrákat ábrázolhatunk. Az ilyen adathalmazok feldolgozása komoly kihívást okozott az eddigi módszerek számára, amely gráf neurális hálózatok kifejlesztését motiválta.

Számos különböző megoldás született, azonban a gráf neurális hálózatok tanítása korántsem triviális, hiszen a bemeneti adathalmazaik, a gráfok, életszerű példáknál akár több millió, kiegyensúlyozatlan koncentrációjú csúcsot és élt tartalmazhatnak. Az irregularitás mellett a gráfokkal ábrázolt adatok dinamikusak is lehetnek, hiszen például egy közösségi hálózat folyamatosan új felhasználókkal bővíthet, ami új felhasználók közötti interakciókat is jelent.

A fejlesztési lépések megkönnyítésének és a tanítási folyamat hatékony végrehajtásának érdekében tanítási keretrendszereket alkalmazhatunk, amelyek számos a célnak megfelelő eszközt és módszert magukban hordozó könyvtárak. A kutatásom célja a gráf neurális hálózatok elosztott tanítását támogató keretrendszerek vizsgálata, valamint egy kiválasztott keretrendszer hatékonyabbá tétele (valamely meghatározott aspektust tekintve) az általa alkalmazott, elosztott tanítást támogató eljárás(ok) módosításával.

2. Gráf Neurális Hálózatok

Az elmúlt néhány évben a gráf neurális hálózatok (GNN) felemelkedtek a mély gépi tanulás világában, hála az általuk felmutatott sikereknek gráf alapú adatok és gyakran rendkívül komplex hálózatok modellezésében és feldolgozásában. Széleskörű felhasználásnak örvendenek különböző területeken, mint például kereső és ajánló rendszerekben, klasszikus gépi tanulási problémákban mint a gépi látásban vagy beszédfelismerésben, forgalom előrejelzésben, és kémiai modellezésben. Ahogy nevük is sugallja, két alapvető pillérük a gráfok és neurális hálózatok koncepciója, így a gráf neurális hálózatok részleteinek interpretálásához elengedhetetlen ezek megértése. Az elméleti bevezetés után a gráf neurális hálózatok céljait, típusait, és alkalmazási területeit mutatom be. Egy GNN módfelett hasznos eszköz lehet a kezünkben, azonban ezen hálózatok tanítása korántsem triviális. A fejezet utolsó érintett témái a neurális hálózatok tanítása, annak kihívásai, és hatékony tanításra vezető technikák.

2.1. A gráfelmélet alapvető koncepciói

A gráf neurális hálózatok esszenciális alanyai a gráfok, amelyek a matematika és számítástudomány területén használt alapvető struktúrák. A különböző informatikai ágakban, úgy mint a gépi tanulásban alkalmazott gráfok a matematikai struktúra koncepciójaira építenek, így a gráf neurális hálózatok átfogó megértéséhez elengedhetetlen a gráfelmélet alapjainak ismerete, kezdve a gráf definíciójával.

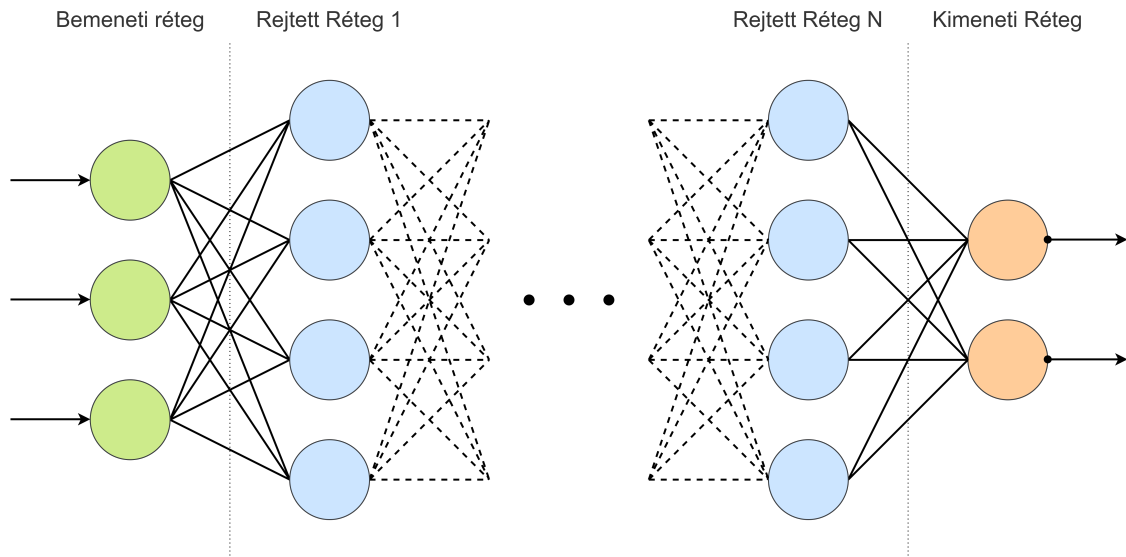
Egy $G = (V, E)$ **gráf** egy matematikai struktúra, ahol V csúcsok, E pedig élek véges halmazát jelöli. A $H = (V', E')$ gráfot $G = (V, E)$ gráf **részgráfjának** nevezzük, amennyiben V' részhalmaza V -nek, E' pedig részhalmaza az E halmaznak és végpontjai V' -hez tartoznak.

Egy $e = (u, v)$ él két végpont, u és v összekötése, amely esetben u és v végpontok szomszédosak. Amennyiben egy él irányított, az általa összekapcsolt csúcspár rendezett. Egy olyan gráfot, amelynek minden éle irányított, **irányított gráfnak** nevezzük. **Súlyozott gráfnak** azokat a gráfokat nevezzük, amelyeknek minden éléhez egy nemnegatív számot, úgynevezett él-súlyt rendelünk. Egy él súlya számos mennyiséget szimbolizálhat, úgymint valószínűség, úthossz, út költsége, vagy vonal kapacitás.

A gráf mint adatstruktúra alkalmas arra, hogy számos dolgot leírjunk segítségével. A leggyakoribb és kézenfekvőbb példa adott objektumok és a köztük lévő kapcsolatok halmaza, de használhatunk gráfot egy kép vagy akár szöveg reprezentációjára is. A számítástudomány és a gépi tanulási technikák folyamatos fejlődése azt is lehetővé tette, hogy gráfok neurális hálózatok bemeneti adathalmazaként szolgálhassanak.

2.2. Neurális hálózatok

A neurális hálózatok olyan számítási modellek, amelyek az emberi agy és annak neuronjainak felépítésére és karakterisztikáira alapozva lettek létrehozva. Ezen struktúrák legfontosabb alkotóelemeiként nagyszámú összekapcsolt egység szolgál, amelyek adott bemenetekhez kimeneteket kötnek, és információt terjesztenek a hálózaton. A számos egyszerű neuron közötti interakciók tették lehetővé a neurális hálók számára, hogy komplex kapcsolatokat modellezzenek.

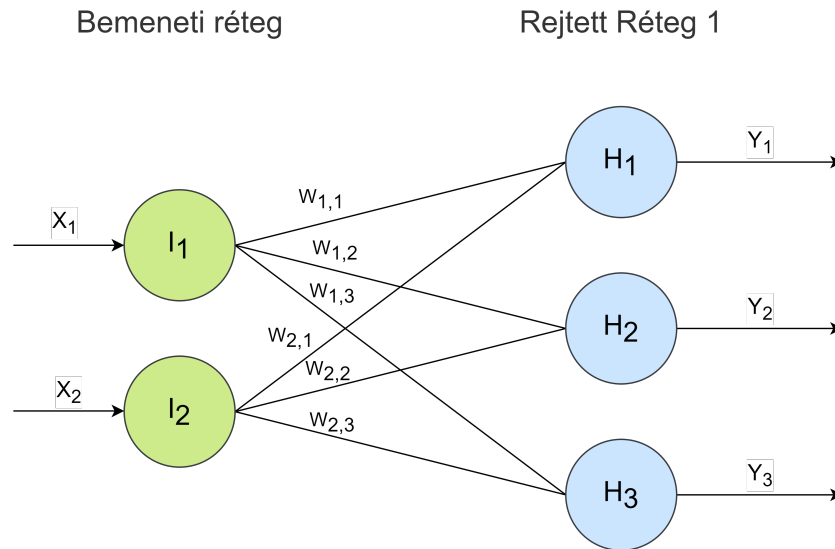


2.1. ábra. Neurális hálózatok általános felépítése

A 2.1 ábrán látható a neurális hálózatok általános felépítése. Rögtön észrevehetjük, hogy a magukat neurális hálózatokat gráfokkal vizualizálhatjuk, azaz csúcsok (neuronok) és élek (kapcsolatok) halmazával. Rendszerint a neuronok három különböző, összekapcsolt rétegbe szerveződnek, amelyeket rendre bemeneti, rejtett és kimeneti rétegnek hívunk. A rejtett réteget (vagy rétegeket) középső rétegnek is nevezzük, hiszen a bemeneti és kimeneti réteg között helyezkedik el. A különböző rétegben elhelyezkedő egység neuronokat kapcsolatok kötik össze, amelyek egyirányú információfolyamot jelképeznek, erősségüket pedig hozzájuk rendelt súly értékekkel írhatjuk le. Ezáltal a neurális hálózatokat **irányított, súlyozott gráfokkal** modellezhetjük.

A hierarchikus, többrétegű felépítés szolgál a neurális hálózatok problémamegoldó képességei alapjaként. A rejtett rétegek számát figyelembe véve megkülönböztethetünk sekély és mély neurális hálózatokat. **Sekély neurális hálózatok** azok, amelyekben egyetlen rejtett réteg van jelen. Amennyiben több rejtett réteg van a struktúrában (általában ez áll fenn), **mély neurális hálózatról** beszélünk.

Egy neurális hálózat minden egysége valós értékű aktivációk sorozatát produkálja, kezdve a bemeneti réteg neuronjaival, amelyek a bemeneti paraméterek alapján aktiválódnak [1]. A további neuronok sorra az előző réteg már aktivált neuronjaival összekötő súlyozott kapcsolaton keresztül aktiválódnak. A neurális hálózatok tanításának célja olyan súlyértékek megtalálása, amelyekkel a kívánt működés érhető el. Ennek érdekében a súlyok (és egyéb tanítási paraméterek) optimalizálásra kerülnek a tanítási folyamat alatt.



2.2. ábra. Neurális hálózat belső paraméterei

Egy neurális hálózat rétegei között történő információáramlás paramétereit láthatjuk részletes lebontásban a 2.2 ábrán. Az ábrán I_i jelöli a bemeneti egységeket, x_i a bemeneti értékeket, $w_{i,j}$ az egységek közötti kapcsolatok súly értékeit, H_j a rejtett réteg egységeit, y_j pedig azok kimeneti értékeit. Ezen kimeneti értékek csupán az adott rejtett réteg kimenetei, nem az egész hálózaté, azaz bemenetként szolgálhatnak egy további rétegnek. A teljes hálózat kimeneteit általában egy erre hivatott kimeneti rétegből kapjuk meg. Egy adott kimeneti értéket az előző réteg súlyozott bemeneteinek az összegeként kaphatunk meg. Az ábrán látható példa y_1 kimeneti értékét az alábbiak szerint számolhatjuk ki:

$$y_1 = x_1 \cdot w_{1,1} + x_2 \cdot w_{2,1} \quad (2.1)$$

Könnyen felismerhetjük, hogy a folyamathoz szükséges számításokat effektíven elvégezhetjük egyesítve mátrixszorzás segítségével. Ennek óriási jelentősége van a számítási kapacitásaink tekintetében, hiszen a mátrixszorzás egy jelentősen optimalizálható folyamat, különösen párhuzamos architektúrákon. Amennyiben a bemeneti és kimeneti értékeket vektorokba, a súlyértékeket pedig mátrixba rendezzük, akkor az ábrán látható információ

áramlási folyamat leírható a következő mátrix szorzással:

$$\begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} w_{1,1} & w_{2,1} \\ w_{1,2} & w_{2,2} \\ w_{1,3} & w_{2,3} \end{pmatrix} \times \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} w_{1,1} \cdot x_1 + w_{2,1} \cdot x_2 \\ w_{1,2} \cdot x_1 + w_{2,2} \cdot x_2 \\ w_{1,3} \cdot x_1 + w_{2,3} \cdot x_2 \end{pmatrix} \quad (2.2)$$

Nem elfelejtendő, hogy a következő neuron számára továbbított végső kimenetet a mátrixszorzásban megkapott y_j értékre alkalmazott **aktivációs függvény** fogja megadni. Az aktivációs függvények célja hogy egy adott réteg bemenetéből előállítsa annak kimenetét, ami a következő réteg bemeneteként szolgál. A megfelelő típusú aktivációs függvény alkalmazása rendkívül fontos tényező, ugyanis még a rétegek számánál is nagyobb mértékben befolyásolhatja a neurális hálózat pontosságát [2].

Annak köszönhetően, hogy **sekély** társaikhoz képest a **mély neurális hálózatok** magasabb számú rejtett réteget tartalmaznak, képesek nagyobb komplexitás mellett megtanulni magasszintű karakterisztikákat. A rétegek (és ezáltal paraméterek) számának növelésének azonban megvan az ára, hiszen a nagyobb használati potenciál mellett a mély neurális hálózatok tanításának erőforrás szükségletei is jelentősen magasabbak. A mély neurális hálózatok nagy pontosságú, effektív alkalmazásánál számos tényező, úgy mint a hálózat felépítése, a tanítási adathalmaz, algoritmus és paraméterek, valamint az alkalmazott infrastruktúra jelentős befolyást gyakorolhatnak. A neurális hálózatok tanítását és annak kihívásait a 2.7 fejezetben részletesebben taglalom.

2.3. Motivációk a gráf alapú gépi tanulásra

A mély gépi tanulás és általa a neurális hálózatok elterjedése számos problémakörben forradalmasította a gépi tanulási feladatok megoldását. Az imént bemutatott konvolúciós és rekurrens neurális hálózatok rendkívül eredményes innovációt jelentettek többek között az gépi látás, fordítás, és beszédfelismerés világában. A neurális hálózatok sikere nagyban hozzájárult a mintafelismerés és adatbányászat témakörökben történő kutatások virágzásához, valamint a számítási kapacitások gyors fejlődéséhez, és a nagy volumenű adathalmazok egyre szélesebb körű elérhetőségéhez [3]. Napjainkra a mély gépi tanulás egy olyan technológiává nőtte ki magát, amely beférkőzött mindennapi életünkbe és mára számos eszközünk esszenciális részévé vált, például a telefonunkon lévő személyes asszisztens (beszéd felismerés) vagy különböző internetes kereső eszközök (szövegértelmezés, fordítás) formájában [4].

Közel sem mondhatjuk lezártnak azonban a neurális hálózatok alkalmazásáról szóló kutatásokat, hiszen egyrésztől egészen újkeletű területeken, mint például az önvezető autók rendszereiben is előszeretettel alkalmazzuk őket, másrésztől pedig a mély gépi tanulás egész új ágazatait is aktívan vizsgáljuk. Ilyen irányzat például az úgynevezett "geometriai" mély gépi tanulás, amely túlmutat a hagyományos neurális hálózatok rácsszerűen

strukturált adatra vonatkozó szükségletein, és olyan feltörekvő technikákat hordoz magában amelyek képesek nem-Euklideszi, "geometriai" adatok alapján tanulni [4]. Az adatok ezen csoportjába tartoznak a gráfok, amelyek számos területen alkalmazhatóak, hiszen segítségükkel modellezhetünk többek között közösségi és szenzor hálózatokat, vásárlók és termékek közötti kapcsolatokat, biológiai és kémiai struktúrákat, publikációk közötti hivatkozás hálózatot, vagy éppen az agy anatómiai és funkcionális építőelemeit [3, 4].

Ahogy az előző bekezdésben láthattuk, számos terület forrása lehet gráf alapú adathalmazoknak. Mindemellett egyre nagyobb az érdeklődés az iránt, hogy a hagyományos gépi tanulási feladatokat, mint a klasszifikálást vagy predikciót ezeken a területeken is kivitelezzük. A célra korábban alkalmazott merev technikáink azonban statikus, rácsszerű struktúrájú adatokon dolgoztak, és nem alkalmazhatóak (könnyen) a nem-Euklideszi adatok terében, ugyanis azok struktúrája dinamikus, így rendezetlen egységeik közötti kapcsolatok gyorsan változhatnak az idő előrehaladtával. Szükségessé vált tehát olyan típusú neurális hálózatok megalkotása, amelyek képesek ilyen adatok, így például gráfok feldolgozására [5].

2.4. Gráf neurális hálózatok alapjai

Ebben a fejezetben ismertetem a gráf neurális hálózatok alapvető funkcionalitását és felépítését. Ezek leírása során nagyban támaszkodom a 2.1 és 2.2 fejezetekben kifejtett alapfogalmakra, amelyek a gráf neurális hálózatok elméleti hátterét képezik.

2.4.1. Műveletek

Vegyünk például egy $G = (V, E)$ gráfot. Ekkor azt mondhatjuk, hogy minden $v \in V$ csúcsonak létezik egy $N(v)$ szomszédsága, amelyet az őt más csúcsokkal összekötő élek határoznak meg. Mindemellett minden v csúcs tartalmaz egy h_v csúcs jellemző reprezentációt, és minden $e \in E$ egy g_e él jellemző reprezentációt. Csúcsok vagy élek jellemző reprezentációi általánosságban őket körülíró skaláris jellemvonások. Egyes esetekben a gráfhoz egy y globális jellemző reprezentáció is kapcsolódik, amely az egész gráfra vonatkozó jellemzőket tartalmaz [6].

Hogy könnyebben megértsük a jellemző reprezentációk szerepét, gondoljunk bele minek is felelhetnek meg ezek a gráfok egyik tipikus alanyai, a közösségi hálózatok esetén. Mivel ebben a felállásban a csúcsok felhasználókat képviselnek, úgy ezek jellemzői felhasználók attribútumai lehetnek, úgymint név, kor, vagy cím. Ehhez hasonlóan, a felhasználók közötti kapcsolatokat megtestesítő élek például a felhasználók közötti interakciókra vonatkozó jellemzőket hordozhatnak. A gráf szintű jellemző reprezentáció magasabb szintű attribútumokat ír le, mint például a felhasználók nemeinek aránya, vagy éppen azok átlag

életkora. A gráf neurális hálózatok elemi feladata, hogy kimeneti jellemző reprezentációkat számoljon ki a csúcsokra, élekre, és a teljes gráfra. Az előző példa gondolatmenetét folytatva, egy csúcs kimeneti reprezentációja információt tartalmazhat az adott felhasználó érdeklődési köréről és vásárlási szokásairól, amely felhasználható például célzott hirdetési rendszerekben. A gráf neurális hálózatok általában több rétegből épülnek fel, és minden réteg csúcs és él összesítő és kombinációs függvények, valamint súlymátrixok alapján dolgozik. A folyamat végén a csúcs és él jellemző vektorok mellett globális egyesítő és kombinációs függvények vannak, amelyek a végső kimeneti vektort nyújtják [6].

Fontos megjegyezni, hogy a gráfok által reprezentált adatok gyakran változnak az idő elteltével. Előző példánkhoz visszatérve, egy közösségi háló új felhasználókkal gyarapodhat, akik új kapcsolatokat és interakciókat hoznak magukkal. Az ilyen adatstruktúrákat dinamikus gráfoknak nevezzük, és a hozzájuk való adaptáció érdekében a gráf neurális hálózatokban végzett számításokat többször megismételhetjük folyamatosan változó súlymátrixokkal [7].

2.4.2. Felépítés

Alapjában véve egy gráf neurális hálózat egy olyan algoritmus, amely a gráfban lévő csúcsok összeköttetéseit használva tanul azok kapcsolatairól, amelyet aztán modellezni is képes. Ugyan a pontos folyamat persze nagyban függ az aktuális struktúrától, általánosságban a gráf neurális hálózatok egy ciklikus bemenetként fogadnak csúcs, él, illetve gráf szintű jellemző vektorokat, és azokból kimeneti jellemző vektorokat állít elő. A bemeneti jellemző vektorok rendre a csúcsok, élek, és a teljes gráfot jellemző attribútumokat, a kimeneti jellemző vektorok pedig a célzott predikciót képviseli. Az általános folyamat a következő főbb lépésekre bontható [6]:

- **Előfeldolgozás:** Az előfeldolgozás egy opcionális lépés, amelyet a folyamat kezdő lépéseként alkalmazhatunk. Az eljárás lényege, hogy a bemeneti jellemző vektorokat, vagy a teljes gráf felépítésének reprezentációját átalakítjuk annak érdekében, hogy a későbbi algoritmusok komplexitását és feldolgozási nehézségét csökkentjük.
- **Ismétlődő frissítések:** Az előfeldolgozást követő második lépésben ismétlődő, avagy ciklikus frissítéseket végzünk a csúcsok és élek jellemző vektorain. Mindezt a korábban bemutatott összesítő és kombinációs függvények segítségével végezzük el. A folyamat során a csúcsokat a szomszédaik összesített jellemző vektorainak kombinációjával, az éleket pedig az adott él, az ahhoz kapcsolódó csúcsok, és a teljes gráf összesített attribútumainak kombinációjának segítségével frissítjük. A folyamat kimeneteleként új csúcs és él jellemző vektorokat kapunk. Miután az

egyes lépések vagy rétegek az aktuális elem környezetének függvényében végzik el a frissítéseket, a folyamat előrehaladtával egyre távolabbi csúcsok és élek kapcsolata lesz beszámítva.

- **Kiolvasás:** A folyamat végső kimenete egy jellemző vektor, amely csúcsokkal vagy élekkel kapcsolatos információt hordoz, vagy a teljes kimeneti gráfra vonatkozó vonásokat. A globális jellemző vektor (ha van ilyen) frissítése a csúcs és él frissítések után, egyszeri alkalommal történik meg.

Ahogy azt az ismétlődő frissítések leírásában említettem, a gráf neurális hálózatok rétegei a frissítési folyamat egyes lépéseinek felelnek meg, és minden réteg a csúcsoktól egyre távolabbra terjeszti a belőlük származó információt. Ezek alapján belátható, hogy a távoli csúcsok közti kapcsolatok fontossága egy befolyásoló tényező egy gráf neurális hálózat rétegszámának megválasztásánál. A rétegek számát érdemes alacsonyan tartani, miután a túl sok (hét vagy több) rétegből álló hálózatoknál túlsimulási vagy túlilleszkedési problémák léphetnek fel [8].

2.5. Gráf neurális hálózatok típusai

A gráf neurális hálózatok iránti nagymértékű érdeklődésnek hála számos különböző algoritmus és eszköz kifejlesztéséhezre került. Ezen eszközök közül sok a már korábban, a hagyományos mély gépi tanulási eseteknél bevált módszerekre épít, átszabva és bővítve azokat a nem-Euklideszi térben történő felhasználásra. A következő néhány bekezdésben a gráf neurális hálózatok különböző elterjedt típusait kategorizálom és mutatom be.

A gráf neurális hálózatok első modelljét 2004-ben mutatták be [9], majd 2008-ban revizionálták [10] Scarselli és társai. Ezek a korai módszerek a **gráf rekurrens neurális hálózatok (RecGNN)** kategóriájába esnek, amelyek a terület úttörői voltak. Ezen hálózatok célja rekurrens neurális hálózatok segítségével megtanulni a csúcs reprezentációkat. Arra alapoznak, hogy a gráf csúcsai folyamatos információ avagy üzenetváltást folytatnak egészen addig, amíg egy stabil egyensúly helyzet elő nem áll. A gráf rekurrens neurális hálózatok fontos fogalmakat hoztak magukkal, és sok későbbi, a témakörrel kapcsolatos kutatást ihletett meg. Az itt definiált üzenetváltási gondolatot például a térbeli gráf konvolúciós neurális hálózatok örökölték meg [3].

A **gráf konvolúciós neurális hálózatok (ConvGNN)** általánosítják a rácsszerű felépítésű adatokon végzett konvolúciós folyamatot gráf adatokon való végrehajtásra. A mögöttük lévő fő koncepció, hogy egy csúcs reprezentációját a saját és szomszédai jellemzőinek összesítéséből állítjuk elő. A rekurrens gráf neurális hálózatoktól eltérően a magas-szintű csúcs reprezentációkat több egymásra halmozott gráf konvolúciós réteg segítségével nyelik ki. A konvolúciós gráf neurális hálózatok rekurrens társaikhoz hasonlóan sok más

koncepciót ihlettek meg, és mindemellett sok összetett gráf neurális hálózati modell fontos alkotóelemei [3].

A **gráf autóenkóderek (GAE)** a felügyelet nélküli tanítási algoritmusok közé tartoznak. Ezek a gráfokat vagy csúcsokat egy úgynevezett rejtett vektortérbe kódolja le, majd a kódolt információ alapján építi újra a gráf adatokat. A gráf autóenkóderek két fő felhasználási területe a reprezentációk megismerése és a gráf generálás. A reprezentációk esetén csúcs jellemzőket tanulhat meg a gráf struktúrájával kapcsolatos információ, például a gráf szomszédsági mátrix újraépítésével. A generációra különböző eljárások elérhetőek. Léteznek olyan generálási algoritmusok amik lépésről lépésre generálnak csúcsokat és éleket, illetve olyanok amelyek egyszeri kimenet formájában adják meg az új gráfot [3].

2.6. Gráf neurális hálózatok alkalmazása

A gráf neurális hálózatok gráf alapú adatokra vonatkozó feldolgozó képességének hála rendkívül szélessávú használatnak örvendenek tudományos, és egyre inkább ipari és üzleti területeken. Ebben a fejezetben bemutatom ezen algoritmusok segítségével végrehajtott feladatok alapvető típusait és szintjeit, valamint az aktuális alkalmazási és kapcsolódó kutatási területeket.

2.6.1. Gráf adatstruktúrán végrehajtandó feladattípusok

A gráfok lehetséges adatforrásairól és a gráf neurális hálózatokról is sok szó esett már az eddigi fejezetekben, így most szeretném azt felvezetni, hogy pontosan milyen feladatokra is alkalmazhatjuk a megismert eszközöket.

Alapvetően három különböző típusú feladatról beszélhetünk, amelyek a gráf adatstruktúra különböző szintjeinek felelnek meg. Ezek nevezetesen a **gráf-szintű**, **csúcs-szintű**, és **él-szintű** feladatok. Egy gráf-szintű feladatban egy olyan jellemzőt szeretnénk megismerni vagy prediktálni, amely a gráf teljes egészét jellemzi. A csúcs-szinten valamilyen a gráf minden csúcsára vonatkozó jellemzőt prediktálunk. Az él-szintű feladatok esetében élek jellemzőjét, vagy azok jelenlétét szeretnénk prediktálni. Egy gráf neurális hálózat erejét mi sem szimbolizálja jobban, minthogy egyetlen modell formájában olyan eszközt kapunk, amelyet használva képesek vagyunk megoldani mind a három fent bemutatott feladat típust [11].

Hogy könnyebben kössük ezen feladat szinteket az eddigi gépi tanulási ismereteinkhez, lássunk néhány összehasonlító példát. A teljes gráfhoz köthető tulajdonság megtalálása nagyon hasonló például a képfeldolgozási klasszifikációs problémákra, ahol egy adott képre szeretnénk valamilyen címkét akasztani. Hasonló lehet a gépi szövegfeldolgozás,

ahol elemzést végzünk egy teljes mondat hangulatának megértése érdekében. A csúcs-szintű predikciót olyan feladatoknak feleltethetjük meg, ahol például egy kép minden pixelét, vagy szöveg egyes betűit szeretnénk megcímkézni azok adott tulajdonságai alapján. Végezetül, az él-szintű feldolgozás során csúcsok közötti összekapcsolásokat vizsgálunk. A képfeldolgozási példát továbbvive, a csúcsokat a képen látható objektumoknak, a közöttük lévő éleket pedig az objektumok közötti kapcsolatoknak feleltethetjük meg. Az ilyen predikciós feladatokban a cél információ lehet két csúcs közötti él jelenléte, vagy ha jelen van, annak egy attribútuma [11].

2.6.2. Alkalmazási és kutatási területek

A gráf neurális hálózatokat számos olyan kutatási területen, amelyben objektumok és azok kapcsolatain alapuló feladatok vannak, előszeretettel használják. Ebben a fejezetben felsorolok és rövid betekintést nyújtok néhány alkalmazási és kutatási területbe, a teljesség igénye nélkül.

A **gépi látás** területén a gráf neurális hálózatok alkalmazhatjuk jelenetleíró gráfok generálására, pontfelhők osztályozására, vagy (objektumok közötti) cselekvés felismerésre. Természetesen különböző objektumok közötti kapcsolatok felismerése nagyban elősegíti a kép által ábrázolt jelenet jelentését vagy kontextusát (amely jó példa egy gráf-szintű kimeneti jellemzőre). A gráf neurális hálózatokat (elsősorban a konvolúciós modelleket) előszeretettel és kitűnő hatékonysággal alkalmazzák például objektumok és azok kapcsolatainak detektálására, és a képen látható körülményeket leíró információ kinyerésére [12]. Gráf konvolúciót használhatunk továbbá arra, hogy egy bemeneti gráfból jelenet és kontextus információt számítsunk ki (például objektumok határait), majd ezt az információt felhasználva összetett, realisztikus képeket generáljunk [13].

Ugyan a **természetes nyelvek feldolgozásánál** a bemenő adatra mint sorozatra gondolkunk, az belsőleg tartalmazhat gráf struktúrájú információt is, mint például a szavak közötti szintaktikai függőségeket definiáló fát. Ezen információk kinyerése, és arra alapozva szavak jellemző reprezentációinak előállítása hatékonyan elvégezhető gráf konvolúciós neurális hálózatokkal [14]. A nyelvi feldolgozáshoz kapcsolódó, gyakori felhasználási terület továbbá az úgynevezett hivatkozási hálózatok ábrázoló gráfok feldolgozása. Gráf neurális hálózatok segítségével a gráfban szereplő objektumok (úgy mint publikációk) között lévő kapcsolatokat (esetünkben hivatkozásokat) kihasználva megbélyegezhetjük az egyes dokumentumokat [8].

Egy **okos közlekedési rendszer** kiépítéséhez elengedhetetlenek a **forgalmi előjelzések**, amelyek olyan tényezőkre képesek intelligens becslést adni, mint a forgalom sebessége, sűrűsége, vagy az abban résztvevők mennyisége. Gráfként foghatjuk fel az okos közle-

kedési rendszer elemeit, ahol a csúcsok az utakra telepített szenzorokat, az élek pedig a köztük lévő távolságokat szimbolizálják. A csúcsok bemeneti jellemzője lehet az egyes szakaszokon meghatározott forgalmi sebesség. Ekkor egy gráf neurális hálózat segítségével dolgozhatjuk fel az elérhető adatokat, és állíthatunk elő forgalmi predikciókat [15].

2.7. Gráf neurális hálózatok tanítása

Az előző fejezetben megismerkedhettünk számos olyan feladat típussal és területtel ahol a (mély) gráf neurális hálózatok kiváló eredményeket értek el akár több millió csúcsból és élből álló gráf adathalmazokat feldolgozva. Az adathalmazok számottevő mérete hozzájárul komplex és valóélet-beli feladatok hatékony megoldásához, azonban jelentősen hátráltathatja a mély neurális hálózatok tanítását fizikai korlátok (úgy mint memória) súrolásával vagy túllépésével, amelyek új technikák bevezetésére kényszerítenek minket és a tanítási idő drasztikus növekedéséhez vezethetnek [16].

A gráf neurális hálózatok tanításának optimalizálása érdekében gyakran alkalmazott technika például az úgynevezett **mintavétel**. Ha elképzelünk egy komplex adathalmazt, úgy mint egy közösségi hálózatot ábrázoló gráfot, könnyen beláthatjuk, hogy az egyes csúcsok (amelyek esetünkben felhasználókat képviselnek) szomszédsága (azaz szomszédos csúcsok összessége) nagy mértékben eltérő lehet. Mindemellett könnyen belátható, hogy ahogy egyre nagyobb körben vizsgáljuk a szomszédságot (először csak egy csúcs szomszédai, aztán szomszédainak szomszédai is, és így tovább), az érintett csúcsok száma exponenciálisan növekszik, amely egy sokmillió csúcsot és élet tartalmazó gráf esetén könnyen kimerítheti számítási erőforrásainkat. Annak érdekében hogy a szomszédsággal kapcsolatos számítások idő és erőforrás igényeit korlátozzuk és egységes szinten tartsuk nagy méretű, dinamikus, vagy több különböző gráf esetén, a teljes szomszédság helyett egy abból vett, meghatározott méretű mintát használunk [17].

Egy másik módszer a szomszédság számítási feladatok erőforrás igényének kezelésére a gráfok **klaszterezése**. Az eljárás lényege hogy a teljes gráfot egy gráf klaszterező algoritmus segítségével részgráfokra bontjuk, és az egyes lépésekben egy ilyen csúcs csoportosításra korlátozzuk le a szomszédsági számításokat. Amellett hogy a számítási és memória követelményeket jelentősen csökkenti, ez az elviekben egyszerű technika korábbi módszerekhez hasonló modell pontosságot ér el [18].

Fontos felismerni, hogy miután a tanítási adathalmaz és a rétegek számának növelése serkenti a predikciós pontosságot, így az itt bemutatott és egyéb optimalizációs eljárások az erőforrás korlátok leküzdése mellett a betanított modellre is jelentős pozitív hatással lehetnek. A következő két alfejezetben további, azonban a problémát sokkal inkább az infrastruktúra irányából megközelítő módszertanokról esik szó, amelyek komoly előrelé-

péseket tettek lehetővé a mély neurális hálózatok tanításában, alkalmazásában, és kutatásában.

2.7.1. Elosztott tanítás

Ugyan a neurális hálózatok koncepciója korántsem újkeletű, kigondolásuk után a mély neurális hálózatok felhasználása számottevő ideig erősen korlátozott volt a hatékony tanításukra alkalmas módszerek hiánya végett. Helyettük az egyszerűbb felépítésű, kevesebb rétegből álló sekély hálózatokat alkalmaztunk. Ezek ugyan jeleskedtek alapvető klasszifikációs és regressziós feladatok megoldásában, alul maradtak magasszintű, komplex problémákkal szemben. A mély neurális hálózatok ilyen feladatokkal is képesek dolgozni, azonban erre egy jelentősen megnövelt réteg és neuron szám árán képesek. Ennek következményeként újfajta aktivációs függvények, optimalizációs eljárások, hálózati architektúrák lettek kifejlesztve és alkalmazva a gépi tanulásban. Ezek lehetővé tették a mély neurális hálózatok használatát, azonban a folyamat további optimalizálása szükséges maradt, hiszen a nagy volumenű tanítási adatszükséglet jelentős negatív hatással volt ezen modellek tanítási idejére [19].

Az idő előrehaladtával a számítástechnikában történt fejlődések, úgymint a grafikus feldolgozó egységek (GPU) megjelenése és elterjedése új utat nyitottak meg a neurális hálózatok alkalmazására. Ezek az eszközök monumentális számítási kapacitással rendelkeznek, amelynek köszönhetően figyelemre méltó teljesítményt képesek elérni (mély) neurális hálózatok tanítása során. A hagyományos számítási kapacitások, úgy mint processzorok, elfogadható időn belül voltak képesek sekély hálózatok tanítására, azonban a mély neurális hálózatok tanításához szükséges nagymértékű párhuzamos végrehajtást nem tudták elérni. Ugyan egy GPU kártya végrehajtó egységei egy CPU magnál kevesebb funkcionalitást nyújtanak, azok masszív darabszáma lehetővé teszi temérdek a gépi tanuláshoz szükséges alapvető művelet gyors elvégzését. Későbbi, többek között eszközök közötti kommunikációs technológiák azt is lehetővé tették, hogy egy gépen belül akár több GPU kártyát összekötve, együttesen alkalmazzuk erőforrásaikat [19].

Bizonyos (többnyire) fizikai korlátok elérése után egyetlen gép további vertikális bővítése már nem volt kivitelezhető. Ilyen korlátok lehetnek például a GPU kártyák memória mérete, és a GPU és CPU közötti kommunikációs sebesség. Miután egy gép felskálázott erőforrásai továbbra is gyakran kevésnek bizonyultak a mély neurális hálózatok tanításánál, szükségessé vált a horizontálisan skálázott, azaz elosztott rendszerek bevetése. Az ilyen architektúrákban a számítások további gyorsításának érdekében a folyamatot több hálózati alapon összekötött gépre terjesztjük ki, több fél között felosztva a teljes feladatot. Ennek természetesen elengedhetetlenül fontos tényezői a gépeket összekötő hálózat egyes metrikái, úgy mint a sávszélesség, késleltetés, és ráta. Az elosztott rendszerek megvaló-

sítására különböző hálózati technológiákat használhatunk fel. Sok múlik a technológia választáson, ugyanis amíg mindkét modern megoldás, nevezetesen az Ethernet és Infiniband magas sávszélességet nyújtanak, az Infiniband jelentősen több üzenetet képes azonos idő alatt átküldeni, kisebb késleltetések mellett. A magas teljesítményű számításokra dedikált hálózati technológiák is megjelentek, amelyek képesek mindhárom metrikában felülmúlni az eddig említetteket, de ennek ellenére a hálózati kommunikáció továbbra is gyakran szűk keresztmetszetet képez a folyamatban [20].

Az elosztott mély gépi tanulási módszertan lehetővé teszi a tanítási folyamat nagymértékű felgyorsítását több gép erőforrásainak együttes használatával. Két fő tanítási módszer alkalmazható, nevezetesen adat-párhuzamos és modell-párhuzamos végrehajtás. Míg az adat-párhuzamos tanítás során az egyes gépek a bemeneti adathalmaz különböző részeit dolgozzák fel, a modell-párhuzamos esetben ugyanazon modell különböző részeit tanítjuk párhuzamosan. A mély gépi tanulásban legnépszerűbb és fejlett keretrendszerek általában dedikált könyvtárakat biztosítanak az elosztott, GPU erőforrásokat felhasználó tanításra, beleértve az alkalmazott kommunikációs protokollt [21].

2.7.2. Felhő alapú tanítási környezet

Az előző fejezetben szó esett az elosztott rendszerekhez vezető motivációkról, illetve azok előnyeiről, nevezetesen a tanítási folyamat gyorsításáról. Egy elosztott rendszer, és különösképpen GPU erőforrások beszerzése és felállítása azonban közel sem triviális feladat. Míg egy üzleti cég könnyedén rendelkezhet elegendő erőforrással komoly gépi tanulási architektúrák felállításához, az akadémiai világban és magánszemélyek esetén ez sokszor sokkal nagyobb fejtörést okozhat. Az infrastrukturális nehézségek leküzdésére segítséget nyújthatnak a felhők. Ezek olyan, publikus, privát, vagy közösségi rendszerek, amelyek rendkívül jelentős számítási kapacitásokhoz nyújthatnak hálózati alapú, könnyed, gyors, és költséghatékony hozzáférést.

A kutatóknál bevált módszer felhőszolgáltatások, úgy mint számítási-felhő kapacitások alkalmazása, hiszen a mély gépi tanuláshoz szükséges erőforrásokkal rendelkező infrastruktúrák elérhetősége meglehetősen alacsony. Az erőforrás igények mellett azonban további fontos tényezőket kell számításba venni, hogy a "megszerzett" infrastruktúrát hatékonyan ki is tudjuk használni. Ilyen tényezők persze a tanítási adathalmaz és algoritmus, azonban elengedhetetlen a megfelelő elosztott mély gépi tanulási keretrendszer alkalmazása is. Egy ideális keretrendszer lehetővé teszi akár olyan felhő alapú erőforrások hatékony kihasználását is mély gépi tanulásra, amelyek általános felhasználásra lettek tervezve [21]. A kutatásomhoz alkalmazott digitális környezet megvalósítását a HUN-REN Cloud [22] szolgáltatásainak segítségével hajtottam végre.

3. Gráf Neurális Hálózatok Eszközkészlete

A gráf neurális hálózatok tanítása egy a mai napig aktív kutatási téma, hiszen a folyamat (hatékony) végrehajtása jelentős kihívásokat jelenthet. A korábban bemutatott elosztott számítási infrastruktúrák közelebb vihetnek minket a célhoz, hiszen az általuk összefogott erőforrásokat együttesen alkalmazhatjuk gráf neurális hálózatok tanítására. Fontos azonban megjegyezni, hogy az elosztott környezetek skálázási hatékonysága nagyban függ számos tényezőtől, például az egyes gépeket összekötő hálózati kapcsolattól, amelyen keresztül az egyes egységek kommunikálnak. Mindemellett, egy létesített elosztott tanítási infrastruktúra hatásos alkalmazása közel sem triviális, így célszerű a folyamatot támogató keretrendszereket használni. A gráf neurális hálózatok terjedésével folyamatosan új eszközök, többek között elosztott gráf tanulási keretrendszerek lettek kifejlesztve, amik támogathatják munkánkat a területen.

Hosszútávú célom hogy az ilyen keretrendszerek alapos vizsgálata során beazonosítsak javítható aspektusokat, majd megkíséreljem azok továbbfejlesztését egy választott keretrendszeren. Szükségem lesz továbbá valamilyen teljesítménymérő eszközre, amellyel kiértékelhetem a már létező keretrendszereket, illetve később a saját, valamely eszköz módosításából kapott elosztott tanítási eljárásomat. A mély gépi tanulási teljesítménymérések során olyan paramétereket szerint értékelhetünk mint a tanítási sebesség vagy a tanított modell pontossága. A következő fejezetekben néhány a felvázolt feladatokra alkalmas keretrendszert hasonlítok össze.

3.1. Elosztott tanítást támogató keretrendszerek

A gráf neurális hálózatok területe az elmúlt évtizedben kezdett kivirágozni, amely az elérhető eszközkészleten is érzékelhető, hiszen a hagyományos mély gépi tanuláshoz képest ezen terület eszközkészlete jóval szűkösebb. A két legelterjedtebb gráf tanítási keretrendszer a PyTorch Geometric (PyG) [23] és a Deep Graph Library (DGL) [24]. Ezeket a nyílt-forráskódú könyvtárakat előszeretettel alkalmazza fejlesztők és kutatók aktív közössége. Ugyan támogatják gráf neurális hálózatok elosztott tanítását, nem erre a célra dedikáltak, sokkal inkább általános eszközök.

Miután a PyG és DGL a legkiforrottabb eszközök, a velük kapcsolatos elérhető információmennyiség is feltehetőleg messze felülmúlja a kisebb fejlesztéseket, ami nagy előny lehet ezen rendszerek megértésénél. Mindemellett ezen projektek kódbázisa is igen jelentős, és rengeteg különböző fejlesztő munkájából áll össze, amely nagy komplexitást eredményezhet. Sok friss, kisebb eszköz is van, amelyek az általános PyG vagy DGL alapjaira építve, elosztott tanításra fókuszálva lettek létrehozva. Ilyen például a Roc [25] és a P³ [26], amelyeket szintén röviden ismertetek.

3.1.1. PyTorch Geometric

A PyTorch Geometric [23] egy a rendkívül népszerű PyTorch keretrendszeren alapuló mély gépi tanulási könyvtár, amely a nem-Euklideszi adatokon, úgy mint gráfokon vagy pontfelhőkön alapuló tanítást tesz lehetővé. Az általános gráf adatstruktúra és az azt feldolgozó eljárások mellett tartalmaz számos különböző, újkeletű módszert a relációs tanulás és a három dimenziós adatfeldolgozás területén. A keretrendszer képes magas teljesítményű adatfeldolgozásra, hála a GPU erőforrások kihasználásának és a változó méretű bemeneti adatok hatékonyan történő kezelésének.

A magas teljesítmény elérése érdekében (amely irreguláris, változó volumenű adatok esetében különösen nehéz) a PyTorch Geometric dedikált CUDA kerneleket alkalmaz, amelyek tulajdonképpen GPU-n végrehajtandó függvények. A fejlesztők mindemellett egy egyszerű üzenet-továbbító interfészt követve egy egyesített keretrendszerbe integrálták a nemrégiben megjelenő legtöbb konvolúciós illetve összesítő réteget. A PyTorch Geometric minden megvalósított eljárása egy állandó adatfolyam mintát követ, lehetővé téve a gráf struktúrák dinamikus természetéből adódó folyamatos változások kezelését. Ugyan nagyméretű feladatokat GPU kártyák segítségével végezhetünk érdemben, a keretrendszer minden metódusa támogatja a CPU alapú végrehajtást is [23].

A fejlesztők a keretrendszer egyes részeit és eljárásait, úgy mint a felhasználói interfészeket, adatbetöltő eljárásokat, és a modell inicializációt igyekeztek a lehető legközelebb tartani a PyTorch bevált módszereihez, hogy megkönnyítsék azok megismerését. Mindemellett a felhasználók számára általános interfészeket nyújtanak, amelyek segítségével gyorsan és letisztultan lehet új kutatási ötleteket próbára tenni, amely a az én munkám szempontjából is rendkívül előnyös lehet. Az interfész részeként definiálhatjuk a saját üzenetküldési és frissítési metódusainkat, illetve választhatunk az elérhető összesítő sé mák közül (például összeadás, átlag, maximum) [23].

Nyílt-forráskódú eszköz révén a PyTorch Geometric kódbázisa elérhető a GitHub-on [27], mindemellett pedig egy az elindulást segítő példákat és útmutatókat tartalmazó, részletes dokumentációval rendelkezik [28]. A keretrendszer beépítve támogat számos gyakori teljesítményvizsgálati adathalmazt, és azok automatizált letöltését és feldolgozását is lehetővé teszi. A több mint hatvan adathalmaz között találhatóak például biológiai, hivatkozás-hálózati, és képfeldolgozási területek alá tartozó gráfok [23]. A rendkívül elterjedt PyTorch keretrendszerre való alapozás, az imént felsorolt eszközök, a számottevő elérhető információ, és a beépített teljesítményértékelési támogatás mind erős érvek a PyTorch Geometric alkalmazása mellett.

3.1.2. Deep Graph Library

A Deep Graph Library (DGL) [24] egy nyílt-forráskódú könyvtár, amelyet kifejezetten arra terveztek, hogy rugalmas és erős legyen kutatók kezében, valamint hatékony és magas teljesítményű valóélet-beli, ipari felhasználásoknál. Mindkét oldal követelményeinek kielégítése persze nehéz feladat, hiszen nagy a szakadék a hagyományos, már szoftver és hardver oldalról is jól optimalizált mély gépi tanulás és az irreguláris gráfok világa között.

A DGL által alkalmazott egyik alapvető elv a gráf neurális hálózatokhoz tartozó számítási sémák egyszerűsítése. Ezeket néhány, a felhasználó által beállítható üzenet-továbbítási elemre bontja, amely általánosítja a számítási műveleteket, a neurális hálózaton mindkét irányban (előre és hátrafelé) átfutó számítási utakat lefedve. Amellett hogy ezen elemek optimalizálva vannak a jelenleg elérhető hardver komponensekre, a jövőbeli optimalizációk alapjait is igyekeznek lefektetni, és ezáltal a gráf neurális hálózatok tanítási folyamatának építő elemeiként szolgálni. Ezen felül a DGL a számítások felgyorsítása és az erőforrások hatékony kihasználása érdekében számos párhuzamosítási módszert is alkalmaz [24].

A felhasználók dolgának könnyítése érdekében a DGL központi elemeként kezeli a gráfokat. Ez ugyan természetesnek tűnhet révén hogy a gráf neurális hálózatok területére tervezték, a DGL bizonyos pontokon a többi könyvtárnál tovább viszi ezt az elvet. A keretrendszer kezeli és elfedi a gráf adatstruktúrájának módosításával járó bonyodalmakat, és egy egyszerűsített, átlátható, a gráf analitikát végzők számára már ismerős programozási felületet nyújt. A gráfok szintjén egy teljesen objektum-orientált programozási stílust képvisel, azonban a haladó felhasználók számára alacsony szintű szerkezeteket is elérhetővé tesz [24].

A DGL a PyG-hez hasonlóan tartalmazza a PyTorch alapjait, azonban mellette a TensorFlow [29] és MXNet [30] keretrendszerekre is épült. A DGL egy minél inkább keretrendszer semleges megoldás hívatott lenni, amely az alapját képező keretrendszerek képességeinek minél széleskörűbb kihasználása mellett az egyes keretrendszerek közötti modell átvitelt igyekszik megkönnyíteni. Ennek az oka többek között az, hogy a gráf neurális hálózatokkal foglalkozó összetett folyamatok a DGL-en kívül eső komponenseket is tartalmaznak (úgy mint az adatok előfeldolgozása), amely esetben természetesen előnyös, ha keretrendszer képes más könyvtárakkal együtt dolgozni. A DGL forráskódja elérhető a GitHub-on [31], az alapvető gráf alapú gépi tanulási példáktól haladó anyagokig terjedő dokumentációja pedig a keretrendszer hivatalos weboldalán [32].

3.1.3. Roc

Köszönhetően a PyG és DGL könyvtárak nyílt-forráskódúságának és könnyed használatának, egyre több új, rájuk épülő tanítási keretrendszer jön létre, köztük olyanok, amelyek dedikáltak a gráf neurális hálózatok elosztott tanítását hivatottak támogatni. Ilyen például a Roc [25], amely egy elosztott több GPU-alapú keretrendszer, gráf neurális hálózatok nagy méretű gráfon történő, gyors tanítására. Annak érdekében hogy remek teljesítményt érjen el nagy gráf neurális hálózati modellek életszerű-méretű gráfokon történő tanításánál, együttesen használ fel több számítási állomást és a bennük rejlő GPU erőforrásokat. A tanítás során a Roc a gráfok teljes egészét használja, és a feldolgozási sebesség és tanítási pontosság szempontjából is remek teljesítményt nyújt létező megoldásokhoz hasonlítva. A nagymértékű skálázhatóság elősegíti, hogy több rétegből álló, komplexebb gráf neurális hálózatokat is bevezessünk, amelyek felfedezése erősen korlátozott volt az elosztott keretrendszerek előtt.

A tanítási folyamat érdekében a bemeneti adatokat gyakran több részre osztjuk, és valamilyen terhelés-elosztó algoritmussal igyekszünk optimalizálni a folyamatot az egyes részek között. A gráf alapú gépi tanulás esetében ezt gráf partícionálásnak nevezzük. A probléma azonban az, hogy az adatstruktúra irreguláris természete végett, különösen életszerű gráfok esetében a gráfok mérete és az egyes csúcsokhoz szükséges számítási kapacitás mennyiségének változatossága megnehezíti a kiegyensúlyozott partícionálást. A probléma leküzdésének érdekében a Roc egy úgynevezett költség modellel dolgozik, amely GNN műveletek végrehajtási idejét becsüli meg a bemeneti gráfon (vagy annak részgráfján). A költség modell tartalmaz gráf-szintű információt, úgy mint a csúcsok és élek száma, illetve hardver-szintű információt, mint a memória elérések száma a folyamat során. A Roc ezt a tudást felhasználva számítja ki a gráf partícionálását, majd minden iteráció után frissíti a költség modellt további optimalizáció érdekében [25].

A GNN tanításban alapvető kihívás, hogy az egyetlen csúcsra vonatkozó számítások elvégzéséhez számos más csúcsra is szükségünk lehet. Ez azért okoz fejtörést elosztott tanításnál, mert azok a csúcsok akár más GPU kártyán, vagy teljesen más számítási állomáson lehetnek tárolva. Ez a memóriamenedzsment problémaköre, amely igen jelentős, hiszen az ilyen erőforrás közötti adatátvitel komoly kihatással lehetnek a teljesítményre. A megoldás nehezen megvalósítható manuálisan, hiszen a megfelelő stratégia nagyban függ a bemeneti gráf méretétől és az erőforrásaink fizikai korlátaitól. A Roc ennek érdekében egy dinamikus programozási algoritmust vezetett be, amely képes gyorsan egy a CPU és GPU közötti adatátvitel számát minimalizáló, optimális stratégiát találni [25].

3.1.4. P³

A P³ [26] egy a gráf neurális hálózatok hatékony elosztott tanítását lehetővé tevő keretrendszer. A fejlesztőit három fő észrevétel motiválta a P³ megalkotására. Az első, hogy a gráf neurális hálózatok elosztott tanítása esetén az egyes erőforrásaink között adat függőség van, amelynek köszönhetően a tanítási időtartam egy jelentős részét hálózati kommunikáció teszi ki. A második, hogy ugyan a korábban is említett gráf partícionálási technikák elősegítik az adatstruktúra feldolgozását, nem feltétlenül előnyösek gráf neurális hálózatok tanításánál, sőt akár hátráltathatóak is lehetnek. A harmadik pont tulajdonképpen az első velejárója, nevezetesen a GPU erőforrások számítási kapacitásának kritikusán alacsony kihasználtsága, amelyet a hálózati kommunikációval töltött számottevő idő okoz. A P³ célja hogy ezen problémákat csökkentse vagy kiküszöbölje, és ezáltal javítsa a teljesítményt.

A P³ a Roc-hoz hasonlóan egy újabb megoldás, továbbá a két keretrendszer célja is közel megegyezik, azonban a problémát különböző irányokból közelítik meg. Míg a Roc jelentős optimalizációt hajt végre kifinomult partícionálási technikákkal és megköveteli a gráf jellemzőinek mozgatását az egyes gépek között, a P³ nem alkalmaz semmiféle intelligens partícionálási megoldást és soha nem továbbítja a jellemzőket a hálózaton keresztül [26].

A kitűzött célok elérése érdekében a P³ a gráf neurális hálózatok néhány alapvető tulajdonságát használja ki. Szemben ugyanis a hagyományos gépi tanulási modellekkel, melyeknél az egyes bemeneti adat minták (úgy mint képek) kicsit voltak, maga a modell (és annak paramétereinek száma) pedig óriási, a gráf neurális hálózatok esetében alacsony számú modell paraméterünk, ellenben terjedelmes bemeneti adat mintáink (gráfaink) vannak. A bemeneti gráfok számtalan csúcsot tartalmazhatnak, melyeknek megannyi jellemzőjük lehet. Természetes tehát, hogy a csúcsok alapján számolt jellemző reprezentációk hálózaton keresztüli mozgatása a kommunikáció nagy részét teheti ki. Annak érdekében hogy ezt megkerülje, a P³ a gráf struktúráját és a jellemzőket egymástól függetlenül osztja szét a gépek között [26].

Korábban szó esett olyan technikákról, amelyeknél a jellemző reprezentációk számításánál a csúcsok egy szomszédságával dolgoztunk. Ehhez képest a P³ csupán a gráf struktúrájára hagyatkozik, amely jelentősen kisebb adatmennyiséget jelent. Ezt a gráf struktúrát a modell legnagyobb számítás-igényű rétegében, az első rétegben minden gépre elhelyezi, majd ezen réteg műveleteit úgynevezett rétegen-belüli modell-párhuzamos módon hajtja végre. A hátrélvő rétegek végrehajtását adat-párhuzamos módon folytatja. Ez a művelet push-pull nevet kapta. A P³ a bemutatott műveletek folyamatos végrehajtását futószalagosítással (angolul *pipelining*) optimalizálja. Innen ered a keretrendszer neve, amely a *Pipelined Push-Pull* rövidítése [26].

3.2. Teljesítménymérési keretrendszerek

Ahogy az látható az ismertetett tanítási keretrendszerek bemutatásából, a gráf neurális hálózatok területén dolgozó fejlesztők és kutatók rengeteg erőfeszítést tettek a folyamat optimalizálására. A terület fejlődéséhez elengedhetetlen, hogy a hagyományos mély gépi tanuláshoz hasonlóan hatékonyan tudjunk teljesítményméréseket elvégezni. Ezen mérések során általában a tanítási (és vele adatfeldolgozási) sebességet, illetve a tanított modell pontosságát vizsgáljuk. Ebben a fejezetben két gráfokra szakosodott teljesítménymérési keretrendszert mutatok be.

3.2.1. Open Graph Benchmark

Az Open Graph Benchmark (OGB) [33] egy nyílt-forráskódú teljesítménymérő keretrendszer, amely számos különböző komplex, életszerű adathalmazt tartalmaz. A rendszer célja a skálázható, újraprodukálható, és hatékony gráf gépi tanulási kutatások támogatása. Az OGB adathalmazok három fő karakterisztika mentén lettek tervezve, nevezetesen nagy méret, tág tartomány, és többféle feladat típus.

Az OGB adathalmazok az előtte elérhető adathalmazoknál nagyságrendekkel nagyobb méretűek. Három különböző kategóriába csoportosíthatjuk őket, nevezetesen kicsi, közepes, és nagy. Nevük ellenére még a kis adathalmazok sem lebecsülendők, ugyanis ezek a gráfok is több mint százezer csúcsból, és akár több mint egymillió élből állnak. Ezek mérete még lehetővé teszi, hogy a teljes gráf beleférjen egyetlen GPU kártya memóriájába, amely ideális nagy számításigényű algoritmusok tesztelésére. Ezzel szemben a közepes és nagyméretű adathalmazokat skálázható modellek és elosztott tanítási taktikák fejlesztésekor vethetjük be [33].

A teljesítménymérési keretrendszer célja volt hogy a gráf gépi tanulás néhány leggyakoribb felhasználási területeit lefedje adathalmazával, úgy mint közösségi hálók, biológiai és kémiai struktúrák, és idézeti hálózatok. A széleskörű téma tartomány segíti az általános célú modellek fejlesztését, megkülönböztetve őket a tartomány-specifikus megoldásoktól. Az egyes tartományokhoz az OGB életszerűen felosztott adatokat nyújt, például idő vagy molekula struktúrán alapozott szeleteléssel. Az adatok sokfélesége mellett az OGB támogatja a 2.6.1 fejezetben bemutatott mindhárom feladattípust, nevezetesen gráf, csúcs, és él-szintűeket [33].

3.2.2. GNNMark

A GNNMark-ot [34] az első gráf neurális hálózatok tanítására fókuszáló teljesítménymérési eszközkészlet. Ez egy nyílt-forráskódú megoldás, amely arra lett tervezve hogy gráf neurális hálózatok viselkedését karakterizáljuk GPU erőforrásokon végrehajtott ta-

nítás során. Számos a gépi tanulási közösség által fejlesztett GNN modelt, és többféle alkalmazási területet lefedő adathalmazokat tartalmaz.

A tanítási folyamat karakterizálását az architektúrális szinten hajtja végre, figyelembe véve a tanítási folyamat kihatásait a GPU kártyákra. A végrehajtás során fellépő jelentős szűk keresztmetszetek felismerését a futási időtartam részletes elemzésével segíti elő. A korábban említett gondolatmenetekhez hasonlóan a GNNMark fejlesztői is megállapították, hogy a gráf neurális hálózatok esetében sokkal nagyobb változatosság, és a bemeneti adatoktól és modelltől való függőség figyelhető meg a hagyományos mély gépi tanuláshoz képest [34].

Hogy a GPU tanítás alacsony szintű karakterisztikáit kinyerje, a GNNMark a célra dedikáltan fejlesztett profilozó eszközöket és adatgyűjtő keretrendszereket alkalmaz. Ezek segítségével hardver teljesítményi jelzőket, úgy mint a gyorsítótárral vagy memóriával kapcsolatos információt gyűjthetünk. A több GPU-s munkamenet teljesítményvizsgálata érdekében a GNNMark tartalmaz ezekre célzott, a PyTorch elosztott könyvtára épített munkaterhelési tesztek, amelyekkel az elosztott tanítási rendszerek skálázhatóságát tehetjük próbára [34].

Összességében elmondható, hogy a gráf neurális hálózatok elosztott tanítását támogató keretrendszerekhez hasonlóan a teljesítménymérési megoldások is gyerekcipőben járnak. A kutatásom végrehajtásához elengedhetetlen különböző megoldások összehasonlító kiértékelése, így mindenképpen gondoskodnom kellett ennek megvalósításáról. A bemutatott teljesítménymérő keretrendszerek által nem támogatott, szükséges eszközöket és eljárásokat saját kezűleg kellett kifejlesztenem.

4. Mérési Környezet Megvalósítása

Annak érdekében hogy a gráf neurális hálózatok elosztott tanítását és annak folyamatait alaposan megvizsgáljam, szükségem volt egy a célra dedikált mérési környezetre. Ahogyan azt a 2.7.2. fejezetben taglaltam, ez alatt egy olyan digitális kutatási környezetre célok, amely számítási-felhő kapacitásokat alkalmaz. Alapvető elvárás volt, hogy a környezet lehetővé tegye a gyors és könnyed iterációt a forráskódon illetve a mérések végrehajtásában, valamint hogy hatékonyan használja ki a igénybe vett erőforrásokat.

4.1. Eszközkészlet meghatározása

Az első szükséges lépés a mérési környezet kialakítása felé az eszközkészlet meghatározása volt. A rendszer szíve természetesen a tanítási keretrendszer, amely támogatja gráf neurális hálózatok elosztott tanítását. Az elérhető keretrendszerek közül mutattam be néhányat a 3. fejezetben részletesen. Számos különböző tényező volt, amely alapján a keretrendszerek összehasonlíthatóak voltak, és amelyek vezényeltek a döntési folyamatban. A szempontokat két fő csoportra osztottam, nevezetesen alapvető követelményekre és az elosztott tanítással kapcsolatos tényezőkre. A 4.1 táblázat egy úgynevezett döntési mátrix, amely segít ábrázolni a keretrendszerek közötti eltéréseket az általam kitűzött szempontok alapján.

Keretrendszer	Alapvető Követelmények			Elosztott Tanítás		
	Nyílt forráskód	Python	Dokumentáció	Támogatás	Dokumentáció	Példa Kódok
DGL	Igen	Igen	Igen	Igen	Igen	Igen
PyG	Igen	Igen	Igen	Igen	Nem	Nem
ROC	Igen	Nem	Nem	Igen	Nem	Nem
P3	Nem	Igen	Nem	Igen	Nem	Nem

4.1. táblázat. Tanítási Keretrendszerek Döntési Mátrixa

Az alapvető követelmények közé tartozik, hogy a keretrendszer nyílt forráskódú legyen. Ez egyrészt lehetővé teszi, hogy könnyen kiigazodjak és esetleg módosítsak a keretrendszer által alkalmazott eljárásokon, továbbá általában az elérhető információ mennyisége is magasabb, hiszen a nyílt forráskódú eszközök mellett gyakran fejlesztők közössége áll. Az is szempont volt, hogy a keretrendszer a Python programozási nyelven alapszik-e. Ez egyrészt azért előnyös, mert a Python egy könnyen tanulható és érthető nyelv, amelynek gazdag közössége és kiegészítő csomagtárja van, és remek vizualizációs opciókat is tartalmaz. Másrészt a Python a leggyakrabban alkalmazott nyelv a gépi tanulásban (köszönhetően a korábban felsorolt előnyöknek), amelynek hála kivételes mennyiségű példa és segédanyag érhető el, így mindenképpen érdemes ezt választani ezen a területen. Az utolsó alapvető követelmény részletes dokumentáció elérhetősége, amely nagyban elősegíti a rendszer megértését és használatát. Ezen a téren nem meglepő módon a kiforrott

keretrendszerek, nevezetesen a DGL és PyG remekeltek. Ezek a rendszerek már éles környezetben használható könyvtárak, amelyek nagy és aktív közösséggel rendelkeznek, és ennek megfelelően a dokumentációjuk is alaposan kidolgozott.

Az alapvető követelmények után az elosztott tanításra fókuszálva is kiértékeltem a bemutatott keretrendszereket. Itt az elsődleges szempont persze hogy a keretrendszer támogatja az elosztott tanítást, azonban révén hogy célomnak megfelelően válogattam őket, ezt minden bemutatott megoldás teljesíti. Korántsem ilyen fényes azonban a helyzet a másik két szempontonál, nevezetesen az elosztott tanítással kapcsolatos dokumentáció és példa kódok esetében. Ezek óriási előnyt jelentenek, hiszen hozzájárulnak ahhoz, hogy könnyedén átláthatóak és megérthetőek legyenek a keretrendszer által alkalmazott elosztott tanítási eljárások és azok alkalmazási lehetőségei. A bemutatott keretrendszerek közül egyedül a DGL, azaz Deep Graph Library teljesíti ezt, amelynek az elosztott tanításra dedikált fejezete van mind a dokumentációban, mind a példakódok között. A DGL dokumentációban ismerteti az elosztott tanításban alkalmazható eljárásokat és programozási elemeket (osztályok, függvények), és magyarázattal ellátott minta kódokkal vezet le csúcs- és él-szintű elosztott tanítási feladatokat.

A felsorolt tényezőket és saját terveimet figyelembe véve a Deep Graph Library keretrendszert választottam a gráf neurális hálózatok elosztott tanításának támogatására. Tovább erősített választásomban, hogy a szakirodalmat böngészve számos tudományos publikációt találtam, amely egy erre a keretrendszerre épülő, és azt valamilyen formában továbbfejlesztő megoldást mutat be, amely irányvonal szorosan egyezik a kutatásom céljával.

A tanítási keretrendszer mellett egy teljesítménymérési keretrendszer kiválasztása is szükséges volt. Hála annak, hogy a tanítási keretrendszer már kiválasztásra került, nem alapulatul kellett döntenem. A 3.2. fejezetben két teljesítménymérési keretrendszer került bemutatásra, nevezetesen az Open Graph Benchmark és a GNNMark. Ugyan mindkét keretrendszer ígéretes volt a kutatómunkám céljainak végrehajtásához, a DGL nagyon erősen kötődik az Open Graph Benchmarkhoz, és példa kódjaiban is előszeretettel alkalmazták, amely persze azt jelentette, hogy könnyebben elkezdhettem vele a munkát. Mindemellett további előny az OGB mellett, hogy rendkívül részletes dokumentációja van az elérhető adathalmazokról és azok alkalmazásáról, valamint egy nyilvános rangsorra, amelyen az aktív kutatói közösség igyekszik új modellek és eljárások fejlesztésével minél jobb predikciós pontosságot elérni.

4.2. Elosztott tanítási architektúra

A tanítási és teljesítménymérési keretrendszerek kiválasztása után elkezdődhetett az elosztott tanítási környezet tervezése és kivitelezése. Egy felhő erőforrásokon alapuló di-

gitális infrastruktúráról beszélünk, amely - elosztott tanítás révén - több összekötött számítási állomásból áll. Ahogyan azt korábban említettem, a kutatásomhoz szükséges környezetet a HUN-REN Cloud erőforrásainak használatával valósítottam meg, amely egy OpenStack [35] alapú felhő. Egy ilyen felhőn a számítási állomásainkat virtuális gépek formájában hozhatjuk létre, amelyek hálózati kapcsolattal vannak összekötve.

A HUN-REN Cloud, a felhőkre jellemző általános előnyökön (úgy mint a gyors és könnyed hozzáférés jelentős számítási kapacitásokhoz) túl GPU erőforrásokat is, nevezetesen NVIDIA Tesla V100 [36] és NVIDIA Tesla A100 [37] kártyákat is nyújt. Ahogy azt a 2.7.1. fejezetben felleltem, a grafikus gyorsító kártyák forradalmasították a mély gépi tanulási alkalmazásokat, azonban rendkívül költséges erőforrások, így ezek elérhetősége óriási lehetőséget jelent ezen területen végzett kutatások során. Ezek tudatában természetesen elvárás volt a rendszerrel szemben, hogy támogassa GPU erőforrások használatát.

Elvárás volt továbbá, hogy a mérések és maga a mérési környezet reprodukálható legyen. Napjainkban, az adat korában, ez egy egyre fontosabb tényező olyan kutatások során, amelyek az Open Science (nyílt tudomány) elveinek igyekeznek megfelelni. Így mindamelllett, hogy nyílt forráskódú technológiákkal dolgozunk, egyre inkább előtérbe kerülnek olyan tényezők mint a kutatási eredmények elérhetősége, átláthatósága, és reprodukálhatósága. Annak érdekében hogy a kidolgozott rendszer, azaz a tanítási és teljesítménymérési eszközök és környezetet, valamint a digitális infrastruktúra amelyre mindez felépül megfeleljen az irányelveknek, nyílt forráskódú eszközöket, verziókezelést és programozható infrastruktúrát alkalmaztam.

A korábbi szakmai tapasztalataimat [21] és szakirodalmi ajánlásokat figyelembe véve a kitűzött célok elérése érdekében a mérési környezetet egy úgynevezett referencia architektúra formájában hoztam létre. Referencia architektúrákat sokféleképpen értelmeznek a szakirodalomban, azonban a fogalom esszenciája egy olyan rendszer vagy rendszerterv, amely egy adott célra lett tervezve és implementálva tetszőleges technológiák felhasználásával, és kiindulási pontként vagy akár kész megoldásként felhasználható.

A dolgozatom kontextusában olyan referencia architektúráról beszélek, amely lehetővé teszi egy digitális kutatási infrastruktúra gyors és könnyen kiépítését felhő erőforrásokon. Ezek - a felvázolt általános koncepciónak megfelelően - célzottan fejlesztettek az adott kutatási vagy alkalmazási területre, legyen az gépi tanulás, konténer orkesztráció, vagy épp erőforrás ütemezés. Annak érdekében hogy a célterületet magas hatékonysággal támogassák, a referencia architektúrák naprakész és széles körben elterjedt nyílt forráskódú technológiákat és módszertanokat alkalmaznak. A HUN-REN Cloud számos ilyen referencia architektúrát nyújt [38], és tervezett, hogy a jövőben az ebben a munkában bemutatott, gráf neurális hálózatok tanítására készített megoldás is publikálásra kerül a

felhő szolgáltatásai között.

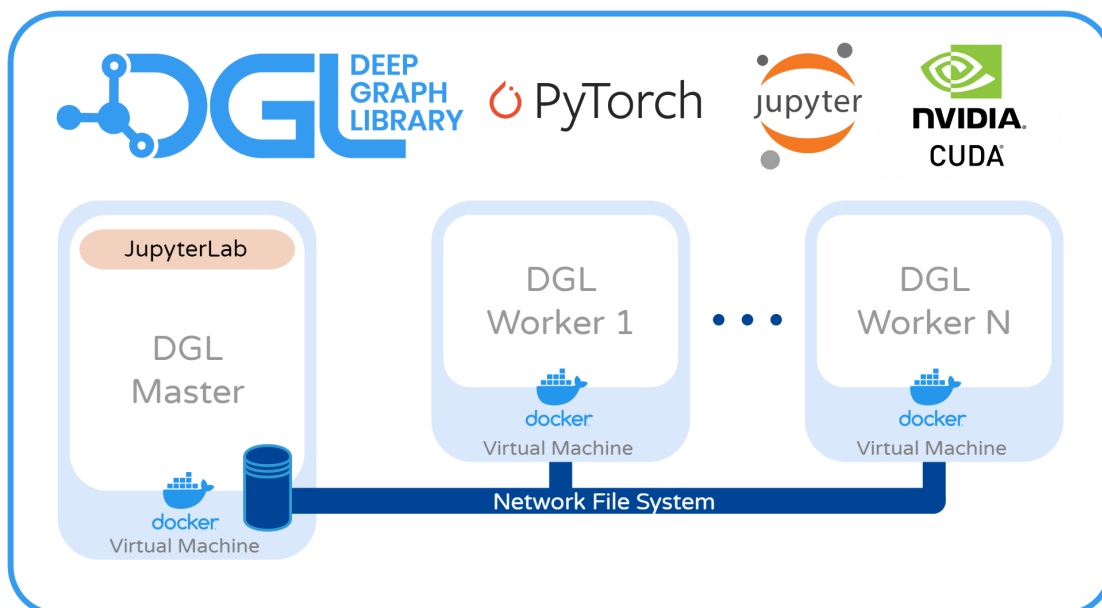
Technológiai szempontból a referencia architektúrák alapját két fő eszköz alkotja, nevezetesen a Terraform [39] és az Ansible [40]. Mind a két eszköz nyílt forráskódú, és az infrastruktúra kódként (IaC) módszertant alkalmaznak, az az deklaratív kód alapján hozzák létre a kívánt rendszerállapotot:

- A **Terraform** egy felhő orkesztrációs eszköz, amelynek feladata hogy a felhő API interfészén keresztül létrehozzon, kezeljen, és termináljon erőforrásokat a felhasználó igényei szerint. Felhő alapú környezetek esetén ilyen erőforrások lehetnek például a virtuális gépek, a tárhelyként szolgáló kötetek, az gépeket összekötő hálózat, vagy a gépeken élesített tűzfal szabályok. A referencia architektúrák a Terraformot használják továbbá az Ansible indítására.
- Az **Ansible** a konfiguráció menedzsmentért felel, azaz feladata hogy a Terraform által felépített infrastruktúrára felhúzza a kívánt szoftver réteget. Ez alatt olyan lépéseket értünk, mint különböző csomagok vagy könyvtárak telepítése, szolgáltatások elindítása, vagy Docker konténerek indítása.

Ezeket az eszközöket használva a kívánt infrastruktúrát tulajdonképpen úgynevezett leíró fájlokban határozzuk meg kód segítségével. Ez azt is lehetővé teszi, hogy testreszabhatóvá tegyük az infrastruktúrát. Mindezt paraméterek segítségével érjük el, amelyeket a kiépítés előtt a felhasználó igényei és elérhető erőforrásai szerint testreszabhat, hiszen a Terraform futási időben olvassa ki majd alkalmazza ezeket. Ezáltal lehetőség adódik többek között a virtuális gépek méretének és számának, szoftvercsomagok verziójának, vagy felületek jelszavának módosítására.

A létrehozott digitális kutatási környezet, azaz a Deep Graph Library referencia architektúra felépítését és főbb technológiáit láthatjuk a 4.3 ábrán. Ahogyan azt korábban felleveztem, a referencia architektúra virtuális gépek klaszteréből áll, amelyben a kommunikációt hálózati kapcsolat biztosítja. A virtuális gépeken Docker konténereket indítunk, amelyek tartalmazzák a szükséges szoftver csomagokat és szolgáltatásokat. A konténerizált megoldásnak számos előnye van, úgy mint magasabb szintű hordozhatóság, erőforrás-hatékonyság, és biztonság.

A gépek között kijelölünk egy elsődleges avagy mester gépet, amely a DGL Master nevet viseli. A kijelölt gép mindössze annyiban tér el a klaszter további gépeitől, más néven a DGL Worker gépektől, hogy két további szolgáltatást nyújt. Az egyik ilyen szolgáltatás egy JupyterLab web-alapú fejlesztői környezet, amelyet előszeretettel alkalmaznak gépi tanulási kutatás lebonyolításához. A második szolgáltatás a Network File System (NFS), amely elengedhetetlen az elosztott végrehajtáshoz. Ehhez ugyanis szükséges, hogy a



4.3. ábra. Deep Graph Library referencia architektúra

klaszter összes állomása elérje a tanítási adathalmazt és kódot. Hagyományos és gráf alapú neurális hálózatok esetén is gyakran előfordul, hogy a tanítási adathalmaz rendkívül nagy méretű (akár több tíz vagy száz GB), amely esetben roppant idő- és erőforrás igényes lenne azt minden gépre átmásolni. Ennek elkerülése érdekében a referencia architektúrában egyedül a DGL Master gép tárolja fizikailag a tanítási adatokat és kódot, míg a többi gép azt az NFS segítségével, hálózati alapon érheti el.

Miután az (elosztott) mély gépi tanulás egy rendkívül erőforrás igényes folyamat, fontos szempont volt hogy a számítási állomások kihasználtságát nyomon lehessen követni a tanítás során. Miután gépek klaszteréről beszélünk, egy központi megoldás implementációja volt célszerű, amelynek használatával könnyedén figyelhetjük az egyes gépek erőforrásait. A korábbi irányelveknek megfelelően a monitorozási rendszer nyílt forráskódú eszközökkel, konténerizált formában került implementációra. A kihasználtsági adatok összegyűjtésére és tárolására a Prometheus [41] nevű monitorozó rendszer és idősor adatbázis szolgál. Ezen adatokat a felhasználó a Grafana [42] analitikai és vizualizációs eszköz által nyújtott felületen érheti el. Olyan metrikákat követhetnek nyomon vagy nézhetnek vissza, amelyek a tanítás szempontjából rendkívül fontosak. Ide tartoznak például olyan alapvető metrikák, mint a CPU magok száma, a memória mérete, és ezek kihasználtsága. Látható továbbá a hálózati forgalom mértéke, amely különösen jelentős tényező elosztott mély gépi tanulás esetén, hiszen a folyamat során a számítási állomások folyamatosan kommunikálnak. Láthatóak továbbá a felhasznált GPU kártyák paraméterei, úgy mint annak típusa, kihasználtsága, illetve memóriájának foglaltsága. Ez mindössze

néhány példa a rengeteg különféle metrikára amelyet a monitorozási rendszer képes eltárolni és vizualizálni.

4.3. Elosztott tanítás paraméterei

A Deep Graph Library referencia architektúra tervezése, implementációja és felépítése után kutatásom alanyát, azaz gráf neurális hálózatok elosztott tanítását kellett megvalósítani. Ennek természetesen nincsen konkrét, behatárolható formája, hiszen a tanítási folyamat számos eszközből, módszertanból, lépésből és azok paramétereiből áll össze, ezáltal rendkívül variálható. A téma nyújtott azonban egy támpontot, amelynek segítségével az elérhető szakirodalmat, esettanulmányokat és megvalósított megoldásokat böngészve tájékozódhattam és felmérhettem lehetőségeimet. Ebben a fejezetben részletekbe menően mutatom be az implementált elosztott tanítási környezet legfontosabb elemeit, kiválasztásukhoz vezető okokat, valamint tulajdonságaikat és működésüket.

4.3.1. Adathalmazok

Általánosságban a mély gépi tanulási alkalmazások, így a gráf neurális hálózatok tanítása is az adathalmaz(ok) előkészítésével kezdődik. Korábban kifejtettem, hogy köszönhetően a részletes dokumentációjának, széleskörű probléma lefedettségének, és a DGL tanítási keretrendszerrel való összefűzöttségének az Open Graph Benchmark keretrendszert választottam az elosztott tanítási folyamatok teljesítményértékeléséhez.

Ahogy az bemutatásakor említettem, az OGB adathalmazai lefedik a gráf alapú gépi tanulás mindhárom szintjét, nevezetesen a csúcs, él, és gráf szintű feladatokat. Dolgozatomban az első problémacsoportra fókuszálok, azaz csúcsok tulajdonságainak predikciójával foglalkozom. Ezen a téren az OGB ezen munka írásakor öt különböző adathalmazt kínált, amelyek legszembetűnőbben méretükben térnek el. Találhatóak kis, közepes, és nagy címkével ellátott adathalmazok is. A címkék között lévő méretbeli eltérést három adathalmaz konkrét adataival szemlélteti a 4.2 táblázat. Ahogy az táblázatból könnyen leolvasható, az adathalmazok között drasztikus eltérések vannak csúcsaik és éleik számában, és ezáltal a tárhely szükségletükben is. Meglepő lehet, hogy mindhárom táblázatban szereplő adathalmaz valójában egy-egy gráf, amely jól mutatja milyen rendkívüli mennyiségű adatot ábrázolhatunk akár egyetlen gráf segítségével. Több gráfból álló adathalmazok általában a gráf-szintű feladatoknál fordulnak elő. A kutatásaim végrehajtásához az elérhető adathalmazok méretét és a rendelkezésemre álló erőforrásokat figyelembe véve a *products* gráfot választottam.

Az első adathalmazokkal kapcsolatos lépés természetesen azok letöltése, azonban ezzel nem érnek véget előfeldolgozási feladataink. A letöltött adathalmazt ugyanis a DGL által biztosított adatstruktúrákra kell konvertálnunk, mielőtt azt a tanításban felhasználhatnánk.

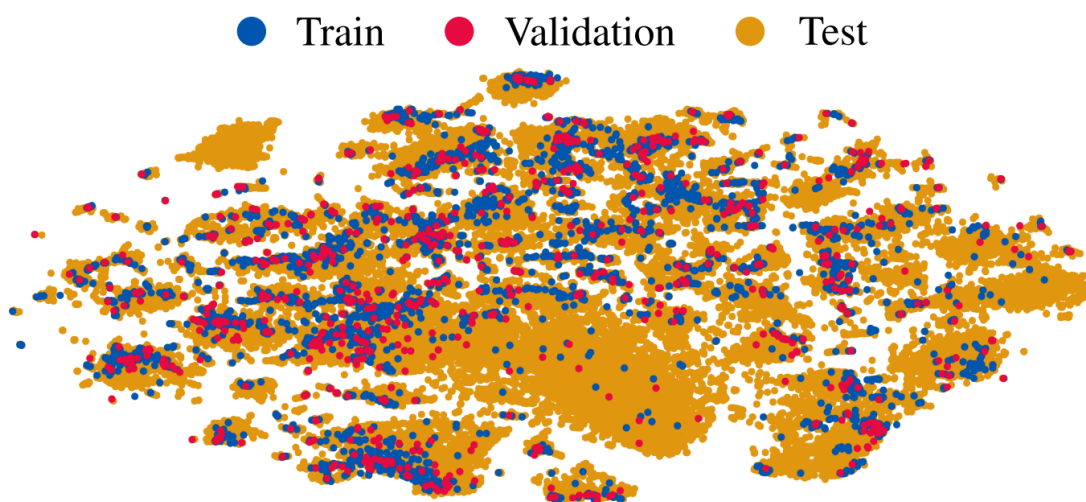
Név	Címke	Csúcsok	Élek	Méret
arxiv	Kicsi	169,343	1,166,243	79 MB
products	Közepes	2,449,029	61,859,140	1.4 GB
papers100M	Nagy	111,059,956	1,615,685,872	56 GB

4.2. táblázat. OGB csúcs tulajdonság predikciós adathalmazok

Csúcs-szintű feladatoknál, ahol adathalmazunk egyetlen gráf, az egész gráfot egy úgynevezett DGLGraph osztályba helyezzük, amely a gráf struktúra illetve csúcsok és élek adatainak tárolására szolgál. Gráf neurális hálózatok tanítása esetén az adathalmazt a hagyományos mély gépi tanuláshoz megszokott módon tanítási, validációs, és teszt részekre osztjuk. Csúcs-szintű feladatoknál ez azt jelenti, hogy a gráf csúcsai kerülnek beosztásra az egyes osztályokba. A DGL egy úgynevezett maszkolási technikát alkalmaz, amellyel megjelöli az adott csúcsok hovatartozását, ami így később könnyen felismerhető a tanítás alatt. A maszkolási folyamat a számítógép hálózatok esetén alkalmazott alhálózati maszkokhoz hasonlóan igaz-hamis értékekkel jelzi, hogy az adott csúcs melyik osztályba tartozik.

A 4.4. ábrán a *products* gráf felosztását láthatjuk, amely esetén a csúcsok 8-2-90 arányban kerülnek elosztásra rendre a tanítási, validációs és tesztelési halmazokba. A felosztás megértéséhez először szeretném a szükséges háttérinformációkat ismertetni az adathalmazról. A *products* adathalmaz az Amazon webáruház egyes termékeinek úgynevezett együtt-vásárlási hálózatát ábrázolja egy irányítatlan, súlyozatlan gráf formájában. Ennek értelmében a csúcsok természetesen termékeket jelképeznek, amelyeket akkor köt össze él, ha azokat együtt vásárolják meg. A számítások alapját képező csúcs tulajdonságok (vagy másnéven jellemzők) a termék leírásának szavai alapján jöttek létre. Ezen adathalmaz esetén a konkrét feladat termékek osztályozása a lehető legnagyobb pontossággal. A csúcsok korábban említett 8-2-90 arányú felosztása az OGB egyedi megvalósítása, amely során a termékeket eladásaik szerint rangsorolták, majd az első nyolc, második két, illetve maradék kilencven százalékot rendre a tanítási, validációs, és tesztelési osztályba sorolták. Mindezt azért tették, hogy a feladat élethű alkalmazást közelítsen meg, amely során a legkelendőbb termékeket címkézik kézzel, majd gépi tanulást alkalmaznak a kevésbé népszerűkre [33].

Annak köszönhetően, hogy a gráf neurális hálózatok elosztott tanítását (is) tervezem vizsgálni, még egy további fontos lépés szükséges az adatok előkészítésében. A tanítást ugyanis a 2.7.1. fejezetben ismertetett adat-párhuzamos módszertan szerint tervezem végrehajtani, amelynek lényege, hogy a tanítani kívánt modellt minden állomásra rámásoljuk, majd azokat az adathalmaz különböző partícióival tanítjuk az egyes állomásokon. Esetünkben az adathalmaz egyetlen gráfból áll, így ezt kell feldarabolnunk. A hagyomá-



4.4. ábra. *products* gráf felosztása [33]

nyos mély gépi tanulásban, például egy képekből álló adathalmaznál a dolgunk végtelenül egyszerű lenne, hiszen a képek függetlenek egymástól, azonban ez korántsem igaz gráfok esetén, ahol az adatok kapcsolatban állnak egymással. Miután összefüggő gráfról beszélünk, azaz bármely két csúcs között van út, a partícionálás során elkerülhetetlen, hogy éleket 'vágjunk el'. Ez egy korántsem triviális szükséglet, hiszen egyrésztől célszerű lenne a lehető legegyszerűbben felosztani a gráfot, másrészt pedig az élek elvágásával információt veszünk csúcsok kapcsolatairól. A gráfok feldarabolására a DGL véletlenszerű partícionálást és az úgynevezett Metis [43] algoritmust nyújtja. A Metis mindamellett, hogy a számítási állomások között igyekszik a lehető legegyszerűbben elosztani a gráf elemeit, a szükséges hálózati kommunikáció minimalizálására is törekszik, ugyanis kerüli szomszédos elemek különböző partícióba helyezését (másszóval a lehető legkevesebb élt próbálja elvágni).

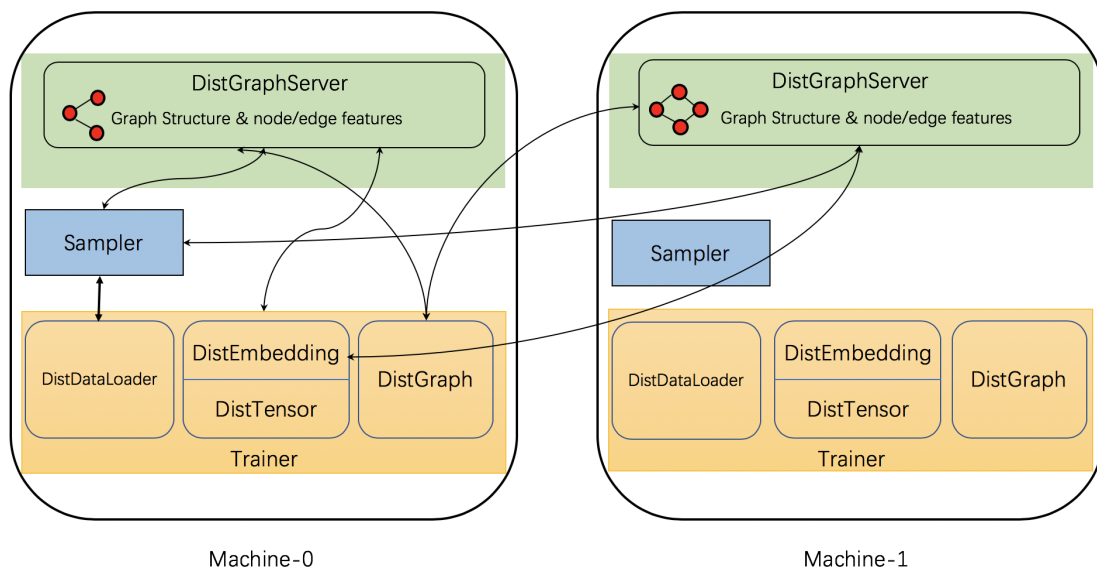
Köszönhetően annak, hogy az OGB egy dedikáltan a DGL keretrendszerhez tervezett előfeldolgozási osztályt, a DGL pedig egy gráf partícionálást végző beépített függvényt nyújt, az adathalmaz letöltése és teljes előfeldolgozása meglehetősen kevés (körülbelül harminc) sornyi kóddal kivitelezhető. A kódnak három nagyon fontos paramétere van, nevezetesen a feldolgozni kívánt adathalmaz neve, a partíciók száma, illetve a partícionálási algoritmus (véletlenszerű vagy Metis). A partíciók számát a számítási egységeink számával kell egyenlővé tenni, azaz ha négy állomáson tanítunk, négyfelé kell osztanunk a gráfot.

4.3.2. Tanítási kód

Az elosztott tanítás kivitelezéséhez a DGL által közétett példakódokat vettem alapul, majd a teljesítménymérési célok szerint módosítottam. A DGL elosztott tanításhoz a PyTorch keretrendszerre, és annak eszközeire hagyatkozik. Az elosztott tanítás során a DGL minden állomásnak kioszt egy partíciót, majd minden állomáson ugyan azt a tanítási kódot futtatja le. Három fajta folyamat indul a klaszter minden állomásán [44]:

- A **szerverek** feladata hogy az adott géphez kerülő gráf partíció struktúráját, illetve csúcsainak és éleinek jellemzőit tárolják.
- A **mintavételezők** feladata az, hogy a szervereken tárolt partíciókon mintavételezést hajtsanak végre, és kötegeket állítsanak elő az elosztott tanításhoz.
- A **tanítók** végzik a folyamat lényegi részét, ugyanis a neurális hálózatok kötegeken történő tanítását végzik. Mindehez a DGL által elosztott tanításra kidolgozott interfészeket használják, amelyek segítségével képesek akár állomásokon keresztül is hozzáférni a szerveren tárolt adatokhoz vagy a mintavételezők által készített kötegekhez. Az egyes állomásokon futó tanítók a számított gradienszeket a PyTorch elosztott adat-párhuzamos tanításra kidolgozott módszerével szinkronizálják.

A három komponens egymás közötti, gépen belüli és gépek közti kapcsolatait a 4.5 ábra szemlélteti. A felhasználó igényei és erőforrásai szerint bármely komponensből akár több is példányosítható a számítási állomásokon.



4.5. ábra. A DGL elosztott tanítási komponensei [44]

Az elosztott tanítás első lépése az **inicializáció**. Ebben a szakaszban olyan lépéseket hajtunk végre, amelyek biztosítják az állomásaink közötti folyamatos kommunikációt, amely

az elosztott tanítás szerves része. Gráf neurális hálózatok elosztott tanítása során az állomásoknak, pontosabban az állomásokon futó tanítóknak hozzáférésre van szüksége a partícionált gráf struktúrájához, illetve a csúcsok és élek jellemzőihez. Mindemellett miután az adott kötegeken minden tanító kiszámolta a gradienseket, azokat egymás között szinkronizálják, hogy a tanított modell konzisztens maradjon. Ezek okán az állomások közötti kommunikáció nem csak szükséges, de az azt biztosító hálózat a folyamat hatékonyságának egyik kulcs tényezője. Az inicializáció során először a DGL elosztott modulját aktiváljuk a klaszterünk állomásainak IP címeinek listájával, amely alapján a hálózati kapcsolat felépül. Másodjára a PyTorch elosztott modult kell inicializálnunk, a kívánt kommunikációs protokoll kiválasztásával. NVIDIA GPU kártyákkal gyorsított tanítás esetén az az NVIDIA által fejlesztett NCCL [45] protokollt érdemes használni, amely több GPU-s és elosztott tanításra optimalizált kommunikációt valósít meg.

A második fő lépésben először betöltjük a gráfot egy úgynevezett DistGraph (azaz ElosztottGráf) objektumba annak neve illetve a partícionálási adatok alapján. Az objektum számos interfészt nyújt, amelynek segítségével a betöltött gráfról adatokat, mint például csúcsainak és éleinek számát kérdezhetjük le. Másodjára a tanítási, validációs, és tesztelési halmazokat széthasítjuk olyan módon, hogy az adott tanítóhoz többnyire azok a csúcsok kerüljenek, amelyek a lokális partíció részei, így csökkentve a szükséges hálózati kommunikációt. A második lépésben valójában nem magukat a csúcsokat, hanem azok azonosítóját töltjük be változóba, amely lehetővé teszi hogy akár állomásokon keresztül megtaláljuk a gráf adott csúcsait.

Miután elérhetővé tettük az adott tanítóhoz rendelt adatokat, a harmadik lépésben egy **mintavételezőt** és egy **adatbetöltőt** hozunk létre. Ahogyan azt a 2.7. fejezetben részletesebben bemutattam, a mintavételezési folyamat lényege, hogy egy csúcs többretegű szomszédságának vizsgálata során minden rétegben az összes csúcs számítása helyett meghatározott méretű mintát veszünk, lecsökkentve ezzel az erőforrásigényt, és egységesítve a szomszédsági méreteket. A mintavételező bemenete egy úgy nevezett 'fanout' amely azt határozza meg, hogy az egyes rétegekben hány csúcsból álló mintát vegyünk. A mintavételező definiálása után egy úgynevezett csúcs-adatbetöltőt hozunk létre, amelynek feladata, hogy lehetővé tegye a **mintavételező segítségével létrehozott kötegeken történő végig-iterálást**. Minden köteg a következő három elemet tartalmazza: Bemeneti csúcsok listáját (amelyeknek bemeneti tulajdonságai szükségesek a kimenetek kiszámításához), kimeneti csúcsok listáját (amelyek kimeneti tulajdonságát számítjuk), és minden réteg számítási függőségeinek listáját (például adott H rétegben X csúcs kimenetének kiszámításához Y darab csúcs bemeneti jellemzői szükségesek). Külön adatbetöltőt hozunk létre a tanításra, validációra, és tesztelésre. Az adatbetöltő paraméterei többek között az elosztott gráf objektum, a létrehozott mintavételező, valamint az adott csúcs osztályba (tanítási, validációs, tesztelési) tartozó, jelenlegi tanítónak kiosztott csúcsok azonosítója.

A mintavételezés és az adatokon történő végig-iterálás biztosítása után negyedik lépésként a **modellt definiáljuk**, azaz a tanítani kívánt gráf neurális hálózatot. Az általam alkalmazott modell részleteit a 4.3.3. fejezetben ismertetem. A modell legfontosabb paraméterei a csúcsok jellemző vektorának mérete, a modell rétegeinek száma, a rejtett rétegekben található neuronok száma, a klasszifikációs osztályok száma, valamint az alkalmazott aktivációs függvény. A létrehozása után a PyTorch DistributedDataParallel moduljának segítségével a modellt **felkészítjük az elosztott tanításra** (amely a gyakorlatban gradiens szinkronizációt jelent a különböző állomásokon található modell replikák között), majd **felmásoljuk a GPU kártyára** (amennyiben GPU alapú tanítást hajtunk végre). A modell mellett egy **veszteségfüggvényt** és egy **optimalizációs eljárást** definiálunk. A veszteségfüggvény segítségével kiértékelhetjük a modell osztályozási hibájának mértékét, az optimalizáló pedig - ahogy azt neve sugallja - optimalizálni hívatott a folyamatot azáltal, hogy a veszteségfüggvényt igyekszik csökkenteni.

Az ötödik lépéssel elérkeztük a tanítási kód utolsó és egyben legfontosabb részéhez, a tanítási folyamat definiálásához. A gyakorlatban tulajdonképpen egy **tanítási ciklust** hozunk létre, amelyben minden lefutás egy epoch-nak, azaz a teljes adathalmazon történő egyszeri végig iterálásnak felel meg. A konkrét tanítási folyamat a következő négy fő lépésre bontható fel:

1. Elsőként minden tanítási ciklusban egy újabb ciklust alkalmazunk, amellyel végig-iterálunk az adatbetöltőn, minden lépésben kapva ezzel egy *mintavételezett köteget* (bemeneti és kimeneti csúcsok, illetve számítási függőségek listái). Ez a tanítási folyamat első lépése, amelyre a szakirodalomhoz hűen **Sampling** (mintavételezés) néven fogok hivatkozni.
2. A második lépés a **Fetching** (avagy lehívás/megragadás), amelynek lényege hogy a Sampling során megkapott köteg bemeneti csúcsaihoz tartozó jellemzőket és kimeneti csúcsaihoz tartozó címkéket betölti. A Samplinghez hasonlóan a Fetching is megkövetelhet különböző állomások közötti adatcserét.
3. Ezek után a harmadik lépés, azaz a **Data Transfer** (adatátvitel) következik, ami során a Fetchingben betöltött adatokat átmásoljuk a GPU memóriájába, ahol a számítás során elérhetjük majd őket (természetesen ez a lépés csak GPU tanítás esetén végrehajtandó).
4. A negyedik és egyben utolsó kötegenként végrehajtandó lépés a **Computation** (számítás). A folyamat gyakorlatilag azonos a hagyományos mély gépi tanulásban előszeretettel alkalmazott hibavisszaterjesztéses algoritmussal, amely során először előrefele (bemenettől kimenetig), majd hátrafele áramoltatjuk az információt. Az előrefele áramoltatás, avagy **forward pass** alatt kimeneteket számolunk ki, majd a

kívánt eredménytől való eltérés (egyszóval hiba) mértékét meghatározzuk a **vesztességfüggvény** segítségével. A hibavisszaterjesztés, avagy **backpropagation** során a vesztességfüggvény minimalizálásának érdekében újra számoljuk a hálózat neuronjainak súlyértékeit, amelyeket a modell után definiált optimalizáló fog **frissíteni**. A tanítás kezdetekor a súlyértékek véletlenszerű értéket vesznek fel 0 és 1 között, majd a hibavisszaterjesztés algoritmusával igyekszünk ezeket úgy beállítani, hogy a vesztesség (esetünkben klasszifikációs hiba) minimalizálódjon.

A tanítási algoritmus a fenti négy lépést az összes kötegen végrehajtja. A kötegek száma az adathalmaz méretétől és a beállított kötegméret függvénye. Miután végig iteráltunk az adatbetöltő összes kötegen, teljesítettünk egy epoch-ot, azaz a teljes adathalmazon való egyszeri végig-iterálást.

Amennyiben célunk egyszerűen a gráf neurális hálózat tanítása lenne, ezen a ponton véget is érne egy epoch. Miután azonban a kutatási munkámhoz alapvető szükséglet az elosztott tanítási folyamat átfogó vizsgálata és megértése, azon igyekeztem, hogy minden teljesített epochról a lehető legtöbb hasznos információt nyerhessem ki. Ennek legfontosabb lépése, hogy a tanított modell pontosságát minden epoch végén kiértékelem a validációs illetve teszt halmazokon. Az általam alkalmazott módszertan, nevezetesen az elosztott adat-párhuzamos tanítás esetén van azonban egy ezzel kapcsolatos nehézség. Ahogy azt korábban felvezettem, adat-párhuzamos tanítás révén ugyan azt a modellt minden állomásra átmásoljuk, feltanítjuk azt az adathalmaz különböző partícióin, majd a kiszámított gradienseket átlagoljuk. Amennyiben a tanítás után a modellt minden gépen kiértékeljük, azt fogjuk tapasztalni, hogy minden állomáson különböző pontosságot ér el. Ez elsőre furcsa lehet, hiszen a szinkronizációs lépés lényege éppen az, hogy a modell konzisztens maradjon az állomások között. Az eltérés oka azonban máshol keresendő. Annak köszönhetően, hogy ahogyan korábban említettem, mintavételezőt és adatbetöltőt alkalmazok a validációs és teszt adathalmazokra is, a modell kiértékelésekor az állomások ezen halmazok eltérő részeit fogják kapni, és különböző mintákon eltérő klasszifikációs pontosságot fog elérni ugyanaz a modell. Annak érdekében, hogy a modell pontosságáról egy minél reprezentatívabb adatot kapjak, egy olyan eljárást implementáltam, amely a gradiens szinkronizálásban használt módszerekkel átlagolja az állomások pontossági adatait, egy globális eredményt kinyerve. A pontosság mellett több jelentős adatot is kinyernek, úgy mint az eltelt idő, feldolgozási sebesség, valamint a tanítási folyamat lépéseinek időbeni lebontása.

A releváns adatok kinyerése után tovább lépünk a tanítási ciklusban következő epoch-ra, és újra végrehajtjuk a folyamatot. Természetesen a teljesített epoch Computation lépésének végén kapott súlyok a következő tanítási ciklus kezdeti súlyai lesznek, 'megtartva' ezáltal a megszerzett tudást. Hogy a program futása alatt pontosan hányszor is hajtjuk

vége a ciklust, másszóval hány epoch-ból áll a tanítás azt a felhasználó határozhatja meg az epochszám paraméter segítségével. Ez csupán egy a számos tanítási paraméter közül, amelyek nagyban befolyásolhatják az elért eredményeinket. A tanítás paramétereit részletesebben a 4.3.4. fejezetben ismertetem.

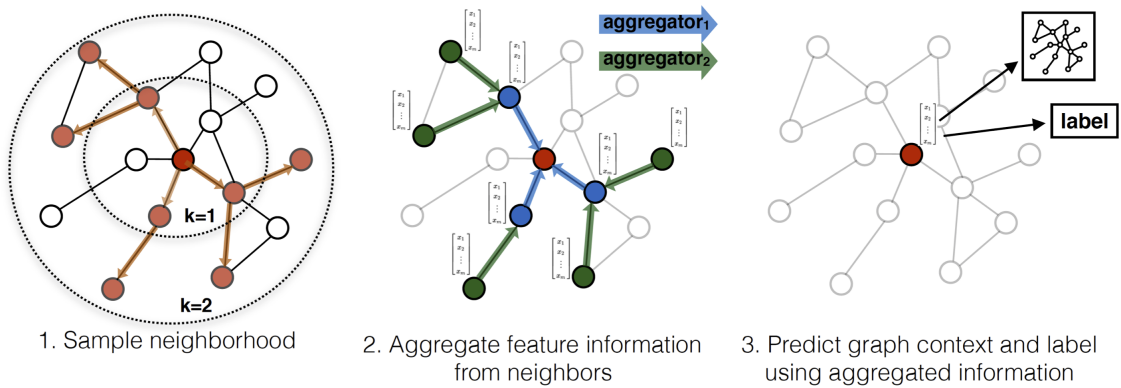
4.3.3. Modell

A választott modell az adathalmaz mellett a tanítás legfontosabb paramétere, így a döntésnek óriási kihatása van eredményeinkre. A megfelelő modell beazonosítása azonban egy rendkívül nehéz összetett feladat, hiszen számtalan tényezőt figyelembe kell vennünk. Különösen nehéz a döntés mély gépi tanulásban, ahol többek között a modell architektúrájának részletei (mint például a rétegek és neuronok száma) is jelentős kihatással lehet annak alkalmasságára. Hogy néhány meghatározó példát soroljak, a választáskor elengedhetetlen olyan tényezők átgondolása, mint a megoldandó probléma természete, a használni kívánt adathalmaz szerkezete és mérete, a teljesítménnyel kapcsolatos elvárásaink (feldolgozási sebesség és pontosság egyaránt szignifikáns tényező lehet), valamint a tanítási infrastruktúránk és erőforrásaink lehetőségei és korlátai.

A mély gépi tanulás különböző problémaköreiből seregnyi különböző modellt dolgoztak ki, és nincs ez másképp a gráf alapú mély gépi tanulás esetén sem, ahol számos gráf neurális hálózat között válogathatunk, és számuk rohamosan növekszik. A gráf neurális hálózatok különböző osztályokba sorolhatjuk az alkalmazott technikák alapján, amelyek közül bemutattam néhányat a 2.5. fejezetben. A kidolgozott GNN architektúrák többségére elmondható, hogy a rendkívül rossz a skálázhatósága két fő oknak köszönhetően. Elsőként sok modell teljes gráf alapú tanításra lett kidolgozva, másszóval az végrehajtandó számítások megkövetelik, hogy az egész gráfot töltsük be a számítási egység memóriájába. Az OGB adathalmazok méretét szemléltető 4.2 táblázatban szereplő *papers100M* gráfnak több mint egy milliárd éle van. Miután a gráfok memóriaszükséglete arányos csúcsaik és éleik számával, elképzelhetetlen hogy ilyen méretű gráfok elérjenek egyetlen GPU memóriájában. Másrészt, az általuk csúcs tulajdonságok kiszámítására használt műveletek az adott csúcs teljes szomszédságával működnek. Amennyiben két rétegű az alkalmazott architektúra, nem csak a csúcs szomszédjaira, de szomszédjainak szomszédjaira is szükség van. Könnyen belátható, hogy amennyiben teljes szomszédsággal dolgozunk, a számítási szükséglet exponenciálisan növekszik a rétegek számával, amely meggátolta sok kidolgozott megoldás alkalmazását nagyméretű gráfokon. További probléma, hogy rengeteg kezdeti GNN architektúra csak fix, merev gráfokra volt alkalmazható, azaz nem tudtak új, korábban nem látott adatokat kezelni. Ez egy jelentős limitáció ezen modellek alkalmazásában, hiszen a való-életbeli gráfok (mint például egy közösségi háló felhasználói és kapcsolataik), dinamikusak, rendszeresen új adattal bővülnek.

A felvezetett problémákra az elsők között nyújtott megoldásokat az általam választott modell, a GraphSAGE [17]. A döntésemet elősegítette, hogy a gráf neurális hálózatok skálázhatóságára és dinamikus adathalmazok kezelésére bemutatott úttörő megoldásai mellett a szakirodalom illetve az elérhető mintákban is gazdagon szerepel csúcs tulajdonság predikciós feladatoknál.

A GraphSAGE név a Graph és a **S**Ample and **A**ggre**G**at**E** tagokból tevődik össze, amely jól mutatja, hogy a gráf adathalmazokra készült modell két fő eleme a mintavételezés és az aggregáció. A folyamatot a 4.6 ábrán láthatjuk vizualizálva. Az első lépés a **szomszédság alapú mintavételezés** alkalmazása. Az eljárás minden csúcs szomszédságából, majd azok szomszédságából (és így tovább) adott számú csúcsból álló mintát vesz. Adott csúcs közvetlen szomszédságát első ugrásnak, szomszédjainak szomszédságát második ugrásnak nevezzük, és így tovább. Az eredeti csúcstól indulva több ugrásnyi mintát vehetünk, ahol az ugrás száma tulajdonképpen az eredeti csúcs és az adott ugrásba eső csúcsok közötti élek száma. Az, hogy pontosan hány ugrásnyi mintát veszünk, a modell rejtett rétegeinek számától függ (egyenlő vele). Ahogyan a 4.3.2. fejezetben említettem, a minták méretét az úgynevezett fanout bemeneti paraméterrel határozhatjuk meg. A GraphSAGE ezzel az eljárással korlátozza a számítási költséget (jelentősen növelve a skálázhatóságot), és egyenletesen tartja az egy csúcsra eső információ mennyiséget a gráfban [17].



4.6. ábra. GraphSAGE mintavételezés és aggregáció [17]

A második lépés az **aggregáció**, amelyet ugrásról ugrásra hajtunk végre a gráf (vagy kötegekre bontás esetén az adott köteg) csúcsain. Az aggregáció két fő műveletből áll, amelyek közül az első, nevezetesen a **szomszédsági jellemző vektor előállítását**, a 4.1 egyenlet írja le. Ez egyenlet paraméterei a következők: v jelöli az aktuális csúcsot, amelynek kimeneti reprezentációját számoljuk, $\mathcal{N}(v)$ annak szomszédságát, u ezen szomszédságba eső csúcsokat. $\mathbf{h}_v^k$ a v csúcs jellemző vektora a k -adik ugrásnál. Az *AGGREGATE* egy aggregációs függvény, amely lehet például átlagoláson vagy összeítéson alapuló. A művelet során az összes v csúcs szomszédságába eső u csúcs előző

iteráció során generált $\mathbf{h}_u^{k-1}$ jellemző reprezentációját aggregálja egyetlen vektorba, amelyet $\mathbf{h}_{\mathcal{N}(v)}^k$ jelöl.

$$\mathbf{h}_{\mathcal{N}(v)}^k \longleftarrow \text{AGGREGATE}_k(\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\}) \quad (4.1)$$

Az aggregációs lépés második művelete az **adott csúcs új reprezentációjának kiszámítása**, amelyet a 4.2 egyenlet ír le. A művelet során először a *CONCAT* eljárás segítségével összefűzzük a v csúcs előző iterációból származó, saját $\mathbf{h}_v^{k-1}$ jellemző reprezentációját az előző műveletben előállított $\mathbf{h}_{\mathcal{N}(v)}^k$ szomszédsági jellemző vektorral. Ezután az összefűzés eredményét a $\mathbf{W}^k$ súlymátrixsal szorozzuk be, amely a modell rétegei (jelen architektúrában ugrási szintjei) közötti információ áramlást határozza meg. Végül az így kapott eredményt egy σ aktivációs függvényen vezetjük át, megkapva ezzel a művelet eredményét, nevezetesen az adott csúcs jellemző reprezentációját a jelenlegi iterációra, amelyet $\mathbf{h}_v^k$ jelöl. A súlymátrixok és aktivációs függvények szerepét a 2.2. fejezetben ismertettem részletesebben.

$$\mathbf{h}_v^k \longleftarrow \sigma(\mathbf{W}^k \cdot \text{CONCAT}(\mathbf{h}_u^{k-1}, \mathbf{h}_{\mathcal{N}(v)}^k)) \quad (4.2)$$

A tanítási folyamat során a GraphSAGE tulajdonképpen aggregációs függvényeket tanít be, amelyek különböző ugrási szinteken aggregálják a jellemző reprezentációkat. A 4.6 ábrán ezek a függvények az *aggregator₁* és *aggregator₂* nevet viselik, az általuk aggregált ugrási szintet pedig rendre a kék és zöld színek jelzik. Ez a mechanizmus a dinamikus gráfok kezelésének alapjaként szolgál, hiszen a betanított aggregációs függvények alkalmazásával új, addig ismeretlen csúcsok jellemző reprezentációját is képesek vagyunk előállítani [17].

4.3.4. Tanítási paraméterek

Az elosztott tanítás bemutatásának utolsó fejezetében a célom hogy a tanítás paramétereit (mind bemeneti és kimeneti) röviden ismertessem. A tanítás két legnagyobb paramétere az adathalmaz és modell, amelyeket részletesen ismertettem rendre a 4.3.1. és 4.3.3. fejezetekben mutattam be. A következő néhány bekezdésben alacsonyabb szintű, azonban a tanítás hatékonysága szempontjából hasonlóan fontos paramétereket mutatok be. Általánosságban elmondható, hogy szinte lehetetlen egy optimális állapotot találni paramétereinkre, miután mindegyikük változtatásának előnyei és hátrányai is lehetnek. Érdemes tehát mérlegelnünk, és igényeink illetve erőforrásaink szerint megválasztani a paramétereket. Miután a paraméterek száma magas, néhány különböző kategóriára osztottam őket. Elsősorban bemeneti és kimeneti paraméterekről beszélhetünk, azonban a bemeneti paraméterek sokasága miatt ezeket tovább osztottam alcsoportokra, nevezetesen adathalmazzal kapcsolatos és modellel kapcsolatos bemeneti paraméterekre.

Elsőként az adathalmazzal kapcsolatos paramétereket mutatnám be. Az első ilyen paraméter az **epochszám**, amely azt határozza meg, hány epoch-on keresztül tanítjuk a modellt, másszóval hányszor küldjük végig a teljes adathalmazt a neurális hálózatunkon. Természetesen minél tovább tanítunk, egyre pontosabb lesz a modell, azonban számolnunk kell két tényezővel. Egyrészt a modell pontossága nem fog a végtelenségig növekedni, ugyanis egy idő után egy adott értékhez fog konvergálni. Ez az a pont, ahol - a jelenlegi paraméterek mellett - már nem tudjuk tovább csökkenteni a veszteségfüggvényt (azaz növelni a pontosságot), így a tanítást nincs értelme tovább folytatni. Másrészt persze számolnunk kell azzal, hogy minden epoch adott idő alatt fut le, így a magas epochszám nagyobb modellek és adathalmazok esetén órákig vagy napokig futó tanítást eredményezhet. Adathalmazzal kapcsolatos bemeneti paraméter továbbá a **fanout**, ami ahogyan a 4.3.2. fejezetben említettem, azt határozza meg, hogy a mintavételezés során az egyes ugrási szinteken hány csúcsból álló mintát vegyünk. Ha a fanoutba magasabb értékeket állítunk be, több csúcs információját nyerjük ki, amely nagyobb pontossághoz vezethet, ugyanakkor vészesen megnövelheti a folyamat memóriaköltségét. A csoport utolsó paramétere a **köteg méret**, amely azt határozza meg, hogy hány minta kerül az egyes kötegekbe. A tanítók mindig egy kötegen iterálnak végig önálló számítást végezve a gradiensek szinkronizálása előtt. Ez azt jelenti, hogy egy nagyobb köteg mérettel elérhetjük, hogy a tanítási idő nagyobb részét tegye ki az önálló számítás, és kisebbet a hálózati kommunikáció. Ez jelentős javulást eredményezhet a teljesítményben olyan környezetekben, ahol a hálózati sebesség szűk keresztmetszet, azonban a kötegméretet érdemes megfontoltan beállítani, hiszen a tanított modell pontosságára is hatással lehet [21].

A következő csoport a modellel kapcsolatos bemeneti paraméterek. Kettő közülük a neurális hálózat architektúráját, a harmadik pedig annak tanulási sebességét határozza meg. Az architektúrát meghatározó paraméterek a **rejtett rétegek száma** és a **rejtett rétegek neuronjainak száma**. A rétegek és neuron számának növelésével megnövelhetjük a tanítható paraméterek számát, alkalmasabbá téve ezáltal a neurális hálózatot komplex feladatok megoldására, azonban az emelt paraméterszám velejárója a számítási erőforrás szükségletet növekedése. A harmadik és egyben utolsó paraméter a csoportban a **tanulási ráta**. A 4.3.2 felvázoltam a tanítási folyamat alapvető négy lépését: kiszámoljuk a kimeneteket a forward pass alatt, meghatározzuk a hiba mértékét a veszteségfüggvénnyel, ezt visszaáramoltatjuk a hálózaton a backpropagation alatt és új értékeket számolunk a hálózat súlyainak, amelyeket az utolsó lépésben az optimalizáló frissít. A tanulási ráta azt határozza meg, hogy amikor a súlyokat bizonyos irányba elmozdítjuk (a hiba csökkentésének érdekében), milyen mértékkel tesszük azt lépésenként. A tanulási ráta a modell tanításának egyik legfontosabb paramétere, így javasolt alaposan átgondolni döntésünket. Egy túl alacsony ráta ugyanis túl hosszúvá teheti a tanítási folyamatot, vagy akár be is akaszthatja, míg egy túl magas ráta kedvezőtlen súlyok megtanulásához vezethet [46].

Végezetül néhány kimeneti paramétert említenék meg. Ezek azok a paraméterek, amelyek alapján ki tudom értékelni az elosztott tanítási folyamatot és a betanított modellt. Magára az elosztott tanításra vonatkozó legfontosabb kimeneti paraméterek a **feldolgozási sebesség**, melyet minták/másodperc-ben mérünk, valamint a tanítás alatt eltelt idő, illetve annak összetétele. Érdekes lehet számunkra többek között a **teljes tanítási idő**, **egy epoch átlagos futási ideje**, vagy az egy mintán végzett **számítási folyamat időbeli felbontása** a 4.3.2. fejezetben ismertetett négy lépésre (Sampling, Fetching, Data Transfer, Computation). Miután elosztott tanítást vizsgálunk, fontos ezen tényezőket összehasonlítani különböző számú számítási állomás használatával. Mindemellett a paraméterek szerepének és kapcsolataik megismerése érdekében a kimeneti paramétereket érdemes különböző bemeneti paraméter értékek mellett vizsgálni.

4.4. Teljesítménymérés automatizálása

A tanítási paraméterek kidolgozása és tesztelése után a teljesítménymérésekre fókuszáltam. Természetesen a tanítási kód segítségével előállított adatokat manuális futtatások módján is kigyűjthettem és dokumentálhattam volna, azonban a szisztematikusság és reprodukálhatóság érdekében a célra dedikált, automatizált megoldásokat dolgoztam ki. Mindezt bash és python scriptek segítségével dolgoztam ki, amelyek együttes működésben paraméterezés után lefuttatják a tanítás(oka)t, a futtatások alatt naplózzák állomásaink erőforrásainak kihasználtságát, majd egy jól szervezett kimutatást készítenek szöveges és ábrázolt formában.

Az automatizált mérés legfőbb komponense a **benchmark.sh** bash script. Egy teljesítménymérés elvégzéséhez nincs más dolgunk, mint a paraméterek tetszés szerinti beállítása után egyetlen parancs kiadásával lefuttatni a scriptet. A futtatás közben végig látjuk a tanítási kód kimeneteit, és felügyelhetjük a folyamatot. Hála annak, hogy JupyterLab környezetben dolgozunk, nem kell végig ülnünk egy több órás vagy napos tanítást, egyszerűen elindítjuk a scriptet a webes felületen, utána pedig akár be is zárhatjuk a böngészőt. A teljesítménymérés lefutása után egy az adott mérésre generált mappában találjuk a kapcsolódó naplózást illetve kimutatásokat. A következő bekezdésekben a script három fő lépését mutatom be.

Az első lépés az **inicializáció**, amely alatt a tanítási paraméterek változóként vannak deklarálva, valamint létre van hozva az adott futtatás adatait és kimeneteit tartalmazni hivatott mappa. Ebbe a részbe gyűjtöttem a tanítást befolyásoló változókat, hogy azok könnyen testreszabhatóak legyen a script első soraiban.

A második lépés a script legfontosabb része, nevezetesen a **tanítás(ok) lefuttatása**. Miután a teljesítménymérések során fontos volt a folyamat skálázási hatékonyságának vizs-

gálata, az elosztott tanítás futtatását úgy valósítottam meg, hogy az növekvő állomásszámmal történjen. Ez azt jelenti, hogy ha négy számítási állomás áll rendelkezésünkre, a script először egy, majd kettő, három, és végül az összes állomás bevonásával futtatja le a tanítást. A gyakorlatban erről egy egytől a rendelkezésre álló állomások számáig iteráló ciklus valósítja meg. A naplózások és kimutatások minden egyes futtatásra elő vannak állítva, így a mérések mappái tovább vannak bontva állomásszám alapján. Mielőtt a konkrét tanítást indító parancs kiadásra kerülne, a script elindít egy **monitor.sh** nevű segédscriptet a háttérben, és lejegyzí az indított folyamat azonosítóját. Ahogyan arra neve is utal, a segédscript feladata, hogy a tanítás ideje alatt a háttérben gondoskodjon állomásaink erőforrás kihasználtságának naplózásáról. Mindezt a monitorozó segédscript a 4.3. fejezetben említett Prometheus idősor adatbázis periodikus lekérdezésével, és az így kapott metrikák naplózásával biztosítja. A folyamat azonosítójának lejegyzésére azért volt szükség, hogy a háttérben futó folyamatot a fő benchmark script le tudja állítani a tanítás végeztével. A monitorozó háttérfolyamat indítása után a script lefuttatja a tanításokat növekvő állomásszámmal, illetve a meghatározott tanítási paraméterekkel.

A harmadik és egyben utolsó lépés a szöveges és vizuális kimutatások elkészítése. Ennek érdekében a script először is összegyűjti a tanítási kód kimeneteit azok napló fájljaiból, és a legfontosabb kimeneti paramétereket egy szöveges fájlba összegzi. Hasonlóan jár el az erőforrás kihasználtsági adatokkal, amelyeket a monitorozó háttérfolyamat leállítása után összegyűjt és egy szöveges fájlba ír össze. Végül néhány python nyelven írt script is meghívásra kerül, amelyek segítségével a naplózott adatok alapján a script ábrákat készít a futtatás kimeneti paramétereinek vizualizációjára. Ezáltal automatizáltan ábra készül a tanítás időbeni felbontásáról, a modell tanítás közbeni és végső pontosságáról, illetve erőforrásaink kihasználtságáról.

A teljesítménymérések hatékonysága és az eredmények letisztultsága érdekében fontos volt, hogy a munkaterület rendszerezett legyen. A 4.7 ábrán láthatjuk az elkészült mérési környezet mappa struktúráját. Fentről lefelé haladva az első mappa a **benchmarks**, amely az automatizáltan lefuttatott mérések végeredményeit tartalmazza. Ezen belül minden mérés saját mappát kap, amelynek neve a mérési paraméterekből illetve a tanítás indításának időpontjából áll össze. Az időpont szerinti elnevezésre azért volt szükség, hogy két egyforma paraméterű futtatás is megkülönböztető legyen, és egyedi névvel rendelkezzen. Ahogyan korábban említettem, minden tanítás mappája tovább bomlik az állomások szerinti futtatásokra, amelyek neve a résztvevő állomások számát jelzi. Az ilyen mappákon belül már valóban a mérési eredményeket találhatjuk, amelyek szöveges és vizuális formájú kimutatások. A következő fő mappa a **dataset**, amely a nyers és partícionált adathalmazt tartalmazza. A partícionált gráfok külön mappába kerülnek aszerint, hogy hány részre lettek bontva. Ezek után scriptek következnek. Elsőként a **create_graph.py** python script, amely az adathalmazok letöltését és partícionálását végzi. Ezt követi a



4.7. ábra. Mérési környezet mappa struktúrája

train_dist.py, amely az elosztott tanítás kódját tartalmazza. A tanítási kód minden állomáson történő egyidejű lefuttatását biztosítja a **launch.py** python script, amelyet a DGL keretrendszer nyújt erre dedikált eszközként. Végül a teljesítménymérés automatizálásához tartozó scripteket láthatjuk, azaz az egész folyamatot koordináló és a háttérben monitorozó bash scripteket, illetve a vizuális kimutatásokat készítő python scripteket.

5. Kiinduló Állapot Kiértékelése

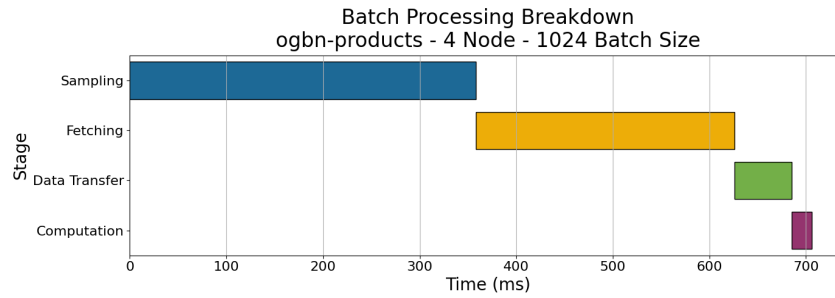
Miután az elosztott tanítási környezet és az automatizált teljesítménymérés is megvalósításra került, elkezdhettem a gráf neurális hálózatok elosztott tanításának területén folytatott kutatást. A célom az általam alkalmazott DGL keretrendszer és referencia architektúra kiinduló állapotának felmérése valamint az elosztott tanítási folyamat illetve annak paramétereinek vizsgálata és megértése, majd olyan pont(ok) beazonosítása, amelyeknek korrigálása vagy újragondolása megnövelheti az elosztott tanítás hatékonyságát. Ebben a fejezetben szeretnék néhány kezdeti mérési adatot bemutatni, és felvázolni az ezek alapján levont következtetéseket, végül pedig egyfajta célkitűzésként megemlíteni azokat az irányokat, amelyeken érdemes lehet elindulni a hatékonyság növelésének érdekében.

A fejezetben bemutatott mérések a HUN-REN Cloud erőforrásainak, és a munkában bemutatott Deep Graph Library referencia architektúra felhasználásával készültek. Az elosztott tanítási klaszter 4 virtuális gépből állt, amelyek mindegyike 8 CPU magot, 32GB memóriát, és 1 NVIDIA Tesla V100 16GB memóriával rendelkező GPU kártyát tartalmazott. A számítási állomásokat egy 10Gbps sávszélességű hálózat kötötte össze. A tanítási és mérési adatok illetve kódok tárolására és állomások közötti megosztására egy 1TB nagyságú SSD alapú köteg szolgált.

5.1. Mérési eredmények

Elsőként néhány olyan kimeneti adatokat vizualizáló ábrát szeretnék bemutatni, amelyeket a teljesítménymérési keretrendszer automatizáltan előállít minden futtatáshoz. Az ábrák címét és paramétereit változók segítségével adja át a benchmark.sh script a vizualizációs python scripteknek. A 5.8 ábrán láthatjuk az első ilyen ábrát, amely a tanítási folyamat időbeli lebontását ábrázolja. A négy sor a 4.3.2 fejezetben bemutatott négy fő lépést ábrázolja. Az ehhez szükséges adatok kinyeréséhez a tanítási kódban időzítőket helyeztem el, amelyek minden egyes köteg feldolgozásánál lementik az adott szakasz kezdeti és vég időpontját. A tanítás végén minden lépés kezdeti és vég időpontját a kimentett értékek mediánjának kiválasztásával határozom meg. Az ábra segíthet megérteni, hogy egy kötegni tanítás alatt pontosan min és mennyit dolgozik a rendszer, amely korrelációban állhat az alkalmazott infrastruktúrával és módszerekkel.

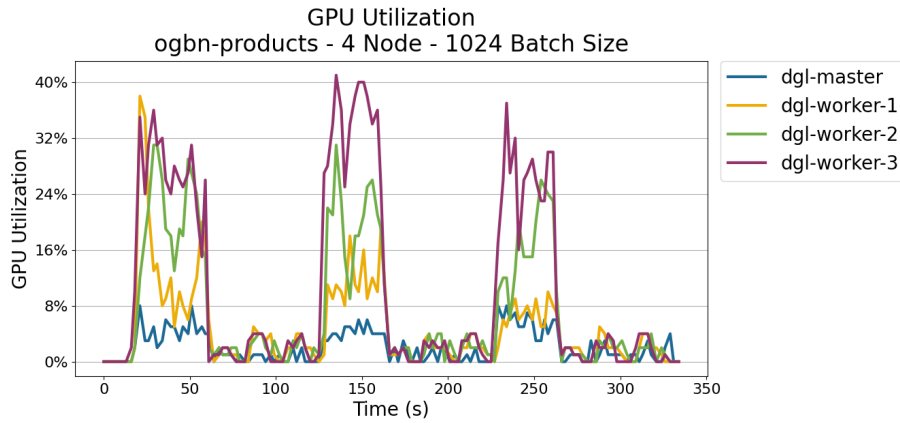
A 5.9 ábrán az állomásaink erőforrás kihasználtságát láthatjuk, nevezetesen a GPU kártyáik kihasználtságát és a kimenő hálózati forgalmukat. Az adatokat a háttérben futó monitoring.sh bash script gyűjtötte három tanítási epoch alatt. Az ábrákon a résztvevő állomásokat különböző színek jelölik. Mindkét ábráról nagyon szépen leolvasható hogy három epoch történt, ugyanis a kihasználtság periodikusan ismétlődik. A 5.9a ábrán látható, hogy a számítási lépések alatt magas a grafikus kártyák használata, a kiértékelési



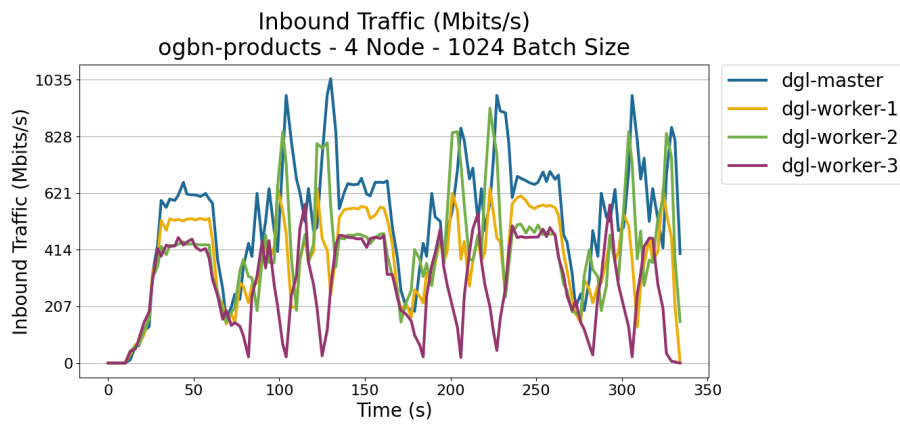
5.8. ábra. Tanítási folyamat időbeli lebontása

szakaszokon pedig alacsony értékre esik vissza. Látható továbbá, hogy az egyes gépek kártyáinak kihasználtsága között jelentős különbségek vannak, és még a számítási szakasz alatt is a GPU kártyák kapacitásának fele (vagy nagyobb része) kihasználatlanul marad. A kimenő hálózati forgalom (5.9b ábra) azonban másképp viselkedik: Leolvasható ugyan a 3 epoch számítási szakasza, a validáció során még a számítási szakasznál is nagyobb hálózati kihasználtságot érünk el, néhány pillanatnyi kiugrás formájában. Ennek oka, hogy a modell pontosságának kiszámítása nagyságrendekkel nagyobb kötegmérettel történik, néhány lépésben.

A felsoroltak mellett a benchmark script ábrát készít az elért pontosságról, a számítási állomások processzorainak kihasználtságáról, illetve a memória foglaltságáról (beleértve a GPU kártya videó memóriáját). A továbbiakban olyan kimutatásokat szeretnék ismertetni, amelyekhez az ábrát kézzel készítettem, és összetettebb képet adnak az elosztott tanítási folyamat hatékonyságáról. Ilyen például a 5.10 ábra, amely a skálázás hatásait vizsgálja különböző kimeneti paraméterekre. A 5.10a ábrán a teszt adathalmazon elért pontosság alakulását láthatjuk 50 epoch-os tanítás alatt. Ahogy az látható, a skálázásnak (minimális kezdeti és végpontbeli eltérés mellett) nincs jelentős hatása a tanítás során elért pontosságra. Minden felállásban, azaz egytől négy állomásos tanítások esetén a modell a 20. és az azt követő néhány epoch intervallumában eléri a 77%-os pontosságot. Leolvasható továbbá, hogy a maximális elért pontosság 77.35%, amelyet a 4 gépes konfiguráció ért el a legutolsó végrehajtott epoch után. Ez alapján azt hihetnénk, hogy nem értünk el jelentős javulást a tanítási folyamat felskálázásával, azonban a 5.10b ábra már egészen másról árulkodik. Itt ugyanis azt láthatjuk, hogy az előző ábra végső állapota, azaz 50 epoch-nyi tanítás összesen mennyi időbe telt az egyes konfigurációkon. Tisztán látható, hogy nem lineárisan ugyan, de a skálázással együtt jelentősen csökken a teljes tanítási idő. A végállapotot, azaz a 76% körüli pontosságot négy állomás használatával több mint kétszer (egész pontosan 2.84x) olyan gyorsan értük el, amely óriási előrelépést jelenthet egy sok órás tanítás esetén. Természetesen a felhasználó igényeinek és erőforrásainak kérdése, hogy ez kifizetődő-e.

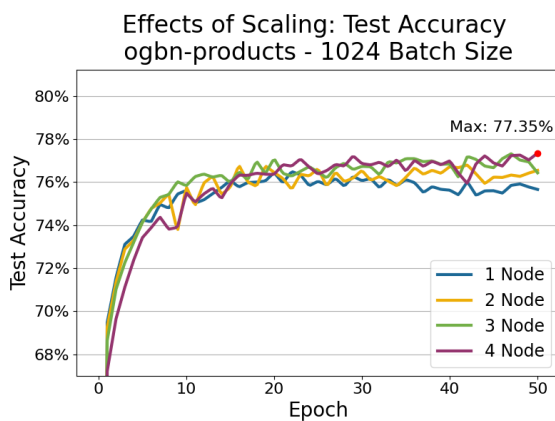


(a) GPU kártyák kihasználtsága

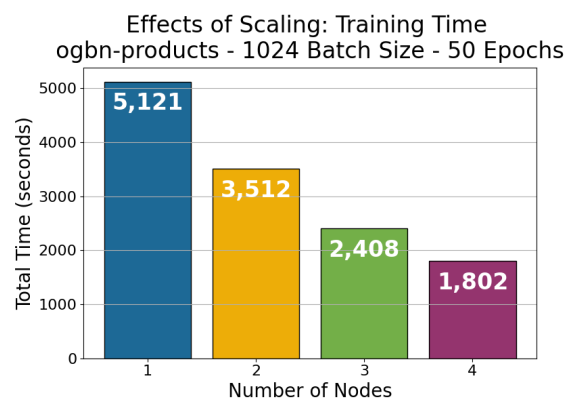


(b) Bejövő hálózati forgalom

5.9. ábra. Erőforrás kihasználtság 3 epoch alatt



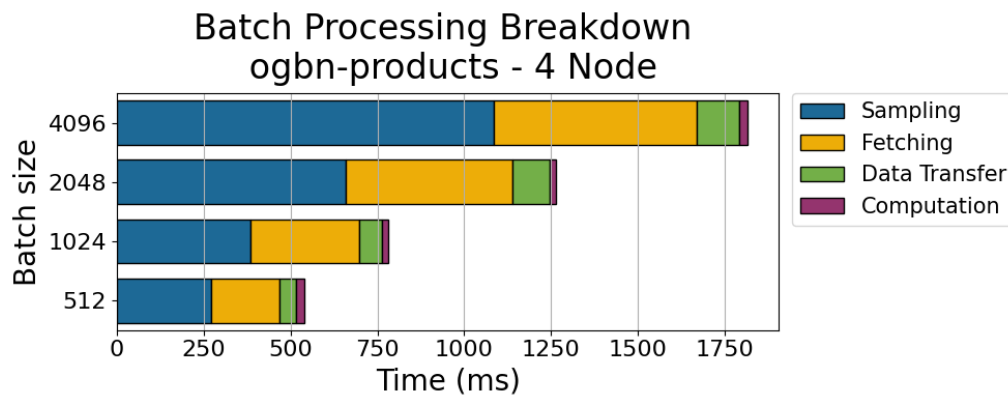
(a) Modell pontosság a teszt halmazon



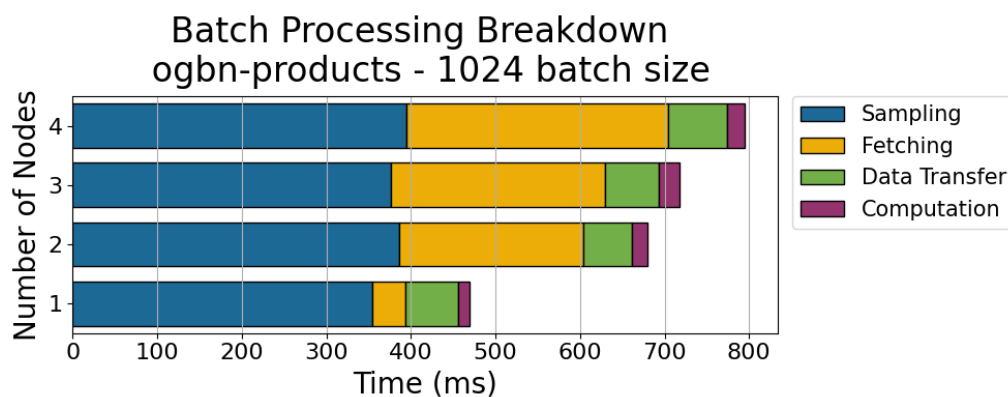
(b) Teljes tanítási idő

5.10. ábra. Skálázás hatásai 50 epoch-os tanításra

Végül az 5.11 ábráról mondanék néhány szót. Ez nagyon hasonlít a korábban bemutatott időbeli lebontásos ábrára (5.8. ábra), azonban ezúttal a tanítás négy lépése egy vonalba került, és ehelyett négy különböző konfigurációban kerül összehasonlításra az eloszlás. Az 5.11a ábrán a kötegméret, az 5.11b ábrán pedig a skálázás hatásait láthatjuk az tanítási folyamat időbeli lebontására. Látható, hogy a kötegméret növelésével, a Computation, az-az számítási lépést leszámítva együttesen növekszenek a folyamat fázisainak végrehajtási idejei. Ez logikus, hiszen mind a három számítás előtti lépés az adatokkal kapcsolatos, így a kötegek adatmennyiségének változása nagy hatást fejt ki. A skálázásnál hasonló, azonban kevésbé drasztikus tendenciákat fedezhetünk fel. A gépek számának növelésével elsősorban a Fetching szakasz időtartama növekedett, amit a gépeken keresztüli jellemző és címke átvitel indokol. Ez persze rendkívül alacsony időtartam egy állomás esetén, ahol a tanítónak csak a lokális memóriát kell elérnie, azonban ahogyan egyre több gép kerül bevonásra, fokozatosan növekszik a szükséges adatok átviteli ideje, csökkentve ezzel a tanítás skálázási hatékonyságát.



(a) Kötegméret



(b) Skálázás

5.11. ábra. Paraméterek hatása a tanítási folyamat időbeli lebontására

5.2. Kiinduló állapot előnyei és korlátai

Összegezve a bemutatott kezdeti eredményeket, az mondható el, hogy a gráf neurális hálózatok tanításának elosztott környezetben, az általam meghatározott paraméterek mellett egyértelműen előnyös. Alátámasztja ezt a 5.10 ábra, amelynek adatai igazolják, hogy a tanítás négy gépre történő skálázásával 2.84x gyorsítást érhattünk el, a modell pontosságán megfigyelhető negatív hatás nélkül.

A kezdeti eredmények azonban arra is rámutattak, hogy bőven van hová fejlődni a folyamat hatékonysága kapcsán. Ugyan a 2.84x gyorsítás egy nagyon szép eredmény, amely egy hosszú tanítás esetén sok órányi különbséget jelenthet, messzi áll az ideálistól, amely a 4x gyorsulást jelenti. Természetesen az ideális érték csak elméleti, hiszen a elosztott tanítás miatt fellépő többletkommunikációnak köszönhetően a gyakorlatban csupán megközelíthető.

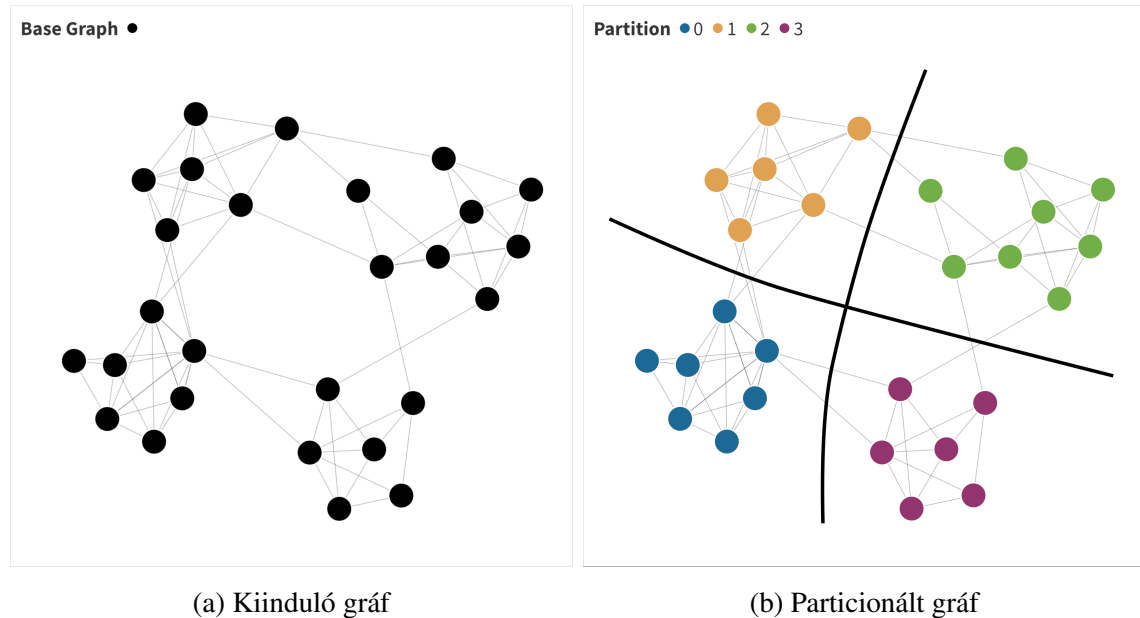
De lássuk a szuboptimális gyorsítás okait. Először is, az 5.9a ábrát megnézve azt látjuk, hogy az esetek döntő többségében a GPU kártyák kihasználtsága az 50%-ot sem éri el. Ez tulajdonképpen azt jelenti, hogy a rendkívül költséges grafikus gyorsítóink számítási kapacitásának több mint fele kárba veszik. Érdemes lehet tehát megvizsgálni, hogy milyen módjai lehetnek a GPU kihasználtság növelésének. További okokat fedezhetünk fel az 5.11 ábrán, amelyek közül a legszembetűnőbb, hogy a tanítás lényegi része, azaz a számítás minden konfiguráció esetén a folyamat időtartamának csupán egy töredékét teszi ki. Ez arra utal, hogy az egész tanítási idő jelentős része csupán az adatok előkészítésével telik, nevezetesen mintavétellel, jellemző és címke vektorok átvitelével, és az adatok felmásolásával a GPU kártyára.

Mindenképpen az elsőszámú lehetséges fejlődési iránynak tartom, hogy a felsorolt lépések átfogó vizsgálata után megkíséreljem a gyorsításukat, ezáltal lecsökkentve egy köteg feldolgozásának, és az egész tanítás végrehajtási idejét. A három lépés közül a Fetching, azaz a jellemző és címke átvitel az, ami kiemelten hátráltatja az elosztott tanítást, hiszen ahogyan azt az 5.11b ábrán is láthattuk, több tanítási állomás bevonásával jelentősen nő a végrehajtási ideje. Miután ez a lépés jelenti a legnagyobb idővesztést, ennek a javítását tűztam ki célul az elosztott tanítás hatékonyságának növelésének érdekében. A következő fejezetben részletesen ismertetem a probléma okait, és a terveimet annak javítására.

5.3. Csúcs jellemző és címke átvitel elosztott környezetben

A Fetching lépés okozta problémák megértéséhez először az adatpárhuzamos gráf-alapú elosztott tanítás alapjait, és a Deep Graph Library kapcsolódó módszereit kell megismernünk. A 4.3.1 fejezetben már említésre került, hogy elosztott tanítás esetén a bemeneti gráfot több, a gépek számával azonos részre kell partícionálnunk. A folyamatot az 5.12

ábrán egy egyszerű minta gráf segítségével igyekszem bemutatni a folyamatot. A partícionált gráf ábráján könnyen látható, hogy a DGL által alkalmazott METIS algoritmus igyekszik a szorosan kapcsolódó csúcsokat egy partícióban hagyni, másszóval minél kevesebb élt "elvágni".



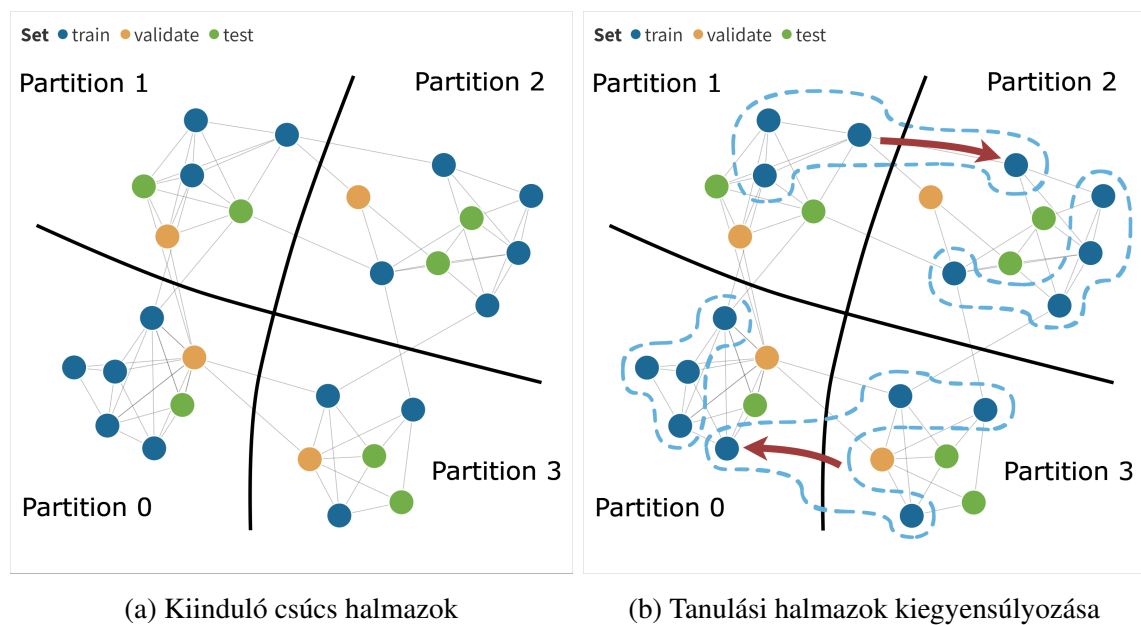
5.12. ábra. Gráf partícionálás elosztott tanításhoz

Az adatok kezelését összetettebbé teszi, hogy a gráf csúcsai más csoportosításon is átesnek, ráadásul még mielőtt a tanítási keretrendszer kezelni kezdi őket. Ahogyan azt a 4.3.1 fejezetben bemutattam, a mély gépi tanulásban használt gráf adathalmazok elemei fel vannak osztva tanulási, validációs és tesztelési halmazokra. Ez a csoportosítás a bemeneti gráf része, és miután nem véletlenszerűen történik, meghatározza a gráfhoz kapcsolódó feladatot, így nem módosítható.

Az 5.13 ábrán a korábban bemutatott minta gráfon szemléltetem, hogy miért is fog nehézségeket okozni, hogy a gráf csúcsait két különböző, egymástól független logika alapján osztjuk fel. Az 5.13a ábrán láthatjuk, hogy a gráf csúcsai a három lehetséges osztályból (tanulás, validáció, tesztelés) melyikbe lettek beosztva. Ha csak a kék színnel jelölt tanulási halmazra összpontosítunk, könnyen láthatjuk, hogy annak elemei az egyes partíciók között egyenetlenül kerültek felosztásra. Ez azért probléma, mert az alkalmazott adatpárhuzamos tanítási módszer megköveteli, hogy minden tanuló azonos mennyiségű adatköteggel dolgozzon.

A DGL úgy orvosolja ezt, hogy azon partíciók esetén, ahol kevesebb tanulási csúcs található, más partíció(k)-tól "vesz kölcsön" csúcsokat, ezzel kiegyensúlyozva a tanulási

halmazokat a tanítás megkezdése előtt. A megoldás kimenetele és egyben gyengesége az 5.13b ábrán látható: Az 1-es számú partícióban csak 3 tanulási csúcs van, így az oda tartozó halmazba a 2-es partícióból veszünk át egy csúcsot, majd hasonlóan járunk el a 0-ás és 3-as partíció között. Végezetül minden partíció tanulási halmazában 4 csúcs található, így előálltak az adatpárhuzamos tanítás feltételei. Fontos megjegyezni azonban, hogy az átvett csúcsok tárolási helye nem változott meg, a csúcsok csoportosítása csupán logikai szinten történik. Ez azt jelenti, hogy az 1-es és 3-as partíció esetében "túl kell nyúlunk" a partíció határán, azaz hálózaton keresztül, egy másik gépről kell elérnünk egy a halmazban szereplő csúcs adatait, ahogy azt az ábrán a vörös nyilak szemléltetik.



5.13. ábra. A DGL csúcs elosztása partíciók között

A helyzetet tovább súlyosbítja, hogy a tanítás során szomszédság alapú mintavételezés történik, hiszen a partícionálási eljárásnak hála egy távoli csúcsnak többnyire távoli szomszédai lesznek, amelyek adatainak begyűjtése jelentősen növeli a hálózaton keresztüli kommunikáció szükségleteit. Ez az oka annak, hogy ahogyan az 5.11b ábrán azt láthattuk, hogy a Fetching lépés időtartama jelentősen növekszik több gép bevonásával. Az adatbetöltés problémája egy igen aktív kutatási terület a gráf neurális hálózatok bevezetése óta, és a 3.1 fejezetben bemutatott létező megoldások közül is több foglalkozott a területtel. Miután munkám az elosztott tanításra és a hatékony skálázhatóságra fókuszál, ezt a problémát tűztem ki célul, és igyekszem leküzdeni az általam kidolgozott megoldással, amelyet a következő fejezetben mutatok be.

6. Partíció Tudatos Csúcs Gyorsítótárazás (PANCA)

Az előző fejezetben ismertetett probléma ismeretében gyorsan belátható volt, hogy a megoldás kulcsa a hálózaton keresztüli csúcs-adat elérés megszüntetése, de legalább csökkenése lesz. Erre két alapvető megoldás lehet: vagy limitáljuk a tanulást kizárólag lokális csúcsokra, vagy valamilyen módon lokálisan elérhetővé tesszük a távoliakat.

Az első felvázolt megoldás egyszerűnek tűnhet, azonban több tényező is ellene szól. Egyrésztől miután elosztott környezetben tanítunk, a bemeneti gráf partícionáláson esett át, és szomszédsági mintavételezést alkalmazunk, nagyon nehéz lenne garantálni a távoli csúcsok kizárását. Ha a 5.13 ábrán szereplő minta gráfot vesszük alapul, könnyen látható, hogy minden partíciónak van olyan tanulási halmazba tartozó (kézzel jelölt) csúcsa, amelyből más partícióban lévő csúcsba vezet él. Ez esetben nem elég csupán lokális csúcsokra szűrni a tanítási halmazt, de még a lokális csúcsok szomszédait is végig kellene szűrni, amely egy élelnagyságú gráfnál igen jelentős erőfeszítést jelentene, és nagyban csökkentené a tanulásra rendelkezésre álló csúcsokat. Ehhez kapcsolódik a második probléma, nevezetesen hogy a tanítási halmaz módosítása, de legfőképp csökkentése jelentősen befolyásolhatja a betanított modell pontosságát.

A második irány, azaz a távoli csúcsok elérhető tétele a lokális partíción már ígéretesebb volt, így ezen a vonalon indultam el, azonban számos meggondolandó pont volt. Természetesen a legegyszerűbb az lenne, ha az összes távoli csúcs adatait a lokális gépre másoljuk. Ugyan egyszerűen kivitelezhető lenne, szembe megy az adatpárhuzamos tanítás alapelveivel, és egyáltalán nem jól skálázható, hiszen a gráf partíciót éppen azért hajtjuk végre, mert nagyméretű adathalmazok esetén teljes gráf nem fér el a számítási állomások, vagy a bennük lévő GPU kártyák memóriájában. Ennek okán egy olyan megoldást kellett találnom, amely jelentősen csökkenti a hálózati eléréseket, azonban egy logikus döntés alapján válogat a távoli partíciók csúcsai között, így korlátozva a többlet erőforrás szükségletet. Számos kisebb kísérletet végeztem el, hogy átfogóbban megérthessem a Fetching lépés menetét. A kísérletek rávezettek arra, hogy mindamellett, hogy a lekért csúcsok listájában nagy gyakorisággal szerepelnek távoli csúcsok, a teljes tanítás során a csúcs ismétlődés is jelentősen magas. Ez utóbbi a szomszédsági mintavételezésnek köszönhető, amely egyben a megoldásom egyik kulcsa is. A folyamat karakterisztikái alapján egy statikus gyorsítótárat terveztem, amelyet a tanítás kezdete előtt töltök fel távoli csúcsokkal illetve azok szomszédaival, lokális hozzáférést biztosítva ezáltal ezek adataihoz. A megoldást Partíció Tudatos Csúcs Gyorsítótárazásnak neveztem el, avagy angol nyelven Partition Aware Node Caching (PANCA).

6.1. Implementáció

A gyorsítótár kidolgozásához először is szükségem volt a logikára, amely kiválasztja mely távoli csúcsok adatai kerüljenek lokális tárolásra. Az erre a célra kidolgozott eljárást mutatja be lentebb egyszerűsített formában az 1-es algoritmus. Az algoritmus szöveges bemutatása során csúcsok alatt minden esetben a csúcsok azonosítóját értem. Az algoritmus bemenetei a kezdeti csúcs lista (esetünkben az adott géphez beosztott tanulási csúcsok), illetve a kívánt ugrási szám avagy mélység, amely meghatározza hogy a távoli csúcsok hány ugrásnyi szomszédságát gyűjtsük be. A folyamat első lényegi lépése a 2. sorban történik, ahol megkeressük a tanulási csúcsok és lokális csúcsok halmazainak különbségét, másszóval a tanulási halmaz távoli csúcsait. Ezeket lementjük a *remote_nodes_with_neighbors* változóba, majd a 6-11. sorokon szereplő ciklusban végrehajtott szomszédsági mintavételezés alapjául szolgálnak. A távoli csúcsok összes szomszédját begyűjtjük (7.), kiszűrjük közülük az ismétlődőket (8.), majd lementjük őket a *remote_nodes_with_neighbors* változóba az eredeti távoli csúcsok mellé (9.). Amennyiben egynél nagyobb mélységet adtunk meg, a következő ciklusban a most begyűjtött szomszédok szolgálnak majd alapul, és azok szomszédságát mentjük le. A ciklus végére megkapjuk a tanulási halmaz eredeti távoli csúcsait, illetve azok teljes szomszédságát kívánt mélységig. Végezetül az ismétlődések mellett (13.) szűrést végzünk lokális csúcsokra is (14.), hiszen távoli csúcs szomszédjai közé tartozhatnak lokális csúcsok is, amelyek már eredetileg is lokálisan vannak tárolva. Az algoritmus kimenete a gyorsítótárazni kívánt csúcsok azonosítóinak listája.

1. Algorithm Gather remote node IDs

Input: nodes, num_hops

Output: cache_list

```
1: local_node_ids  $\leftarrow$  GET_LOCAL_IDS()
2: remote_node_ids  $\leftarrow$  SETDIFFERENCE(nodes, local_node_ids)
3: seed  $\leftarrow$  remote_node_ids
4: remote_nodes_with_neighbors  $\leftarrow$  seed
5:
6: for hop  $\leftarrow$  1 to num_hops do
7:   sample  $\leftarrow$  DISTRIBUTED_SAMPLE_NEIGHBORS(seed)
8:   sampled_nodes  $\leftarrow$  UNIQUE(sample.NODES())
9:   CONCATENATE(remote_nodes_with_neighbors, sampled_nodes)
10:  seed  $\leftarrow$  sampled_nodes
11: end for
12:
13: remote_nodes_with_neighbors  $\leftarrow$  UNIQUE(remote_nodes_with_neighbors)
14: cache_list  $\leftarrow$  SETDIFFERENCE(remote_nodes_with_neighbors, local_node_ids)
```

A bemutatott algoritmus a 4.3.2 fejezetben ismertetett DistGraph (azaz ElosztottGráf) komponensben került implementációra, aminek köszönhetően a felhasználói szinten egyetlen parancs segítségével meghívható a tanítási kódban. A gyorsítótárazni kívánt csúcsok megtalálása után továbbításra kerül a kérés az lokális adattárnak, amely a hálózati kommunikációért felelős RPC komponens eljárásainak segítségével összegyűjti a kijelölt csúcsok adatait, majd lokálisan eltárolja azokat. Míg az eddig felsorolt komponensek Python alapúak, maga az adatbegyűjtési lépés C++ nyelven íródott a gyorsabb végrehajtás érdekében. Leegyszerűsítve a folyamat a gyorsítótárral való kiegészítés után az alábbi lépéseket hajtja végre:

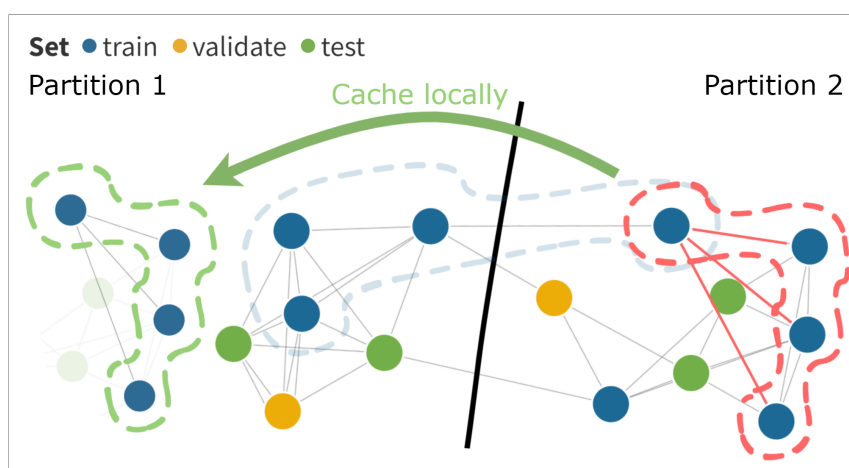
1. Végig nézi a bekért csúcsok listáját, és lejegyzí melyek vannak eltárolva lokálisan, a gyorsítótárban, illetve távoli gépen
2. Elküldi a hálózati kéréseket a távoli csúcsok adataiért
3. Az eredmény vektorba másolja a lokális csúcsok adatait
4. Az eredmény vektorba másolja a gyorsítótárazott csúcsok adatait
5. Fogadja a hálózati válaszokat, majd az eredmény vektorba másolja a távoli csúcsok adatait

A gyorsítótár tulajdonképpen egy második, eltérő elemeket tartalmazó lokális adattárként funkcionál, és a hálózati kérelmek számát hívatott csökkenteni. Felmerülhet a kérdés, hogy az adatok beszerzése miért a távoli csúcsokkal kezdődik (2. lépés). Ez tulajdonképpen egy okos optimalizálási "trükk" a DGL fejlesztőitől, amelynek lényege hogy a hálózati válaszokra való várakozást lefedjük a lokális adatok másolásával, ezzel is csökkentve a teljes végrehajtási időt.

Természetesen miután milliszekundumokban mérhető a Fetching lépés végrehajtása, elsődleges sorrend a kód hatékonysága (mi sem támasztja ezt alá jobban, minthogy annak ellenére hogy a DGL egy Python könyvtár, az adatbegyűjtés C++ nyelven íródott). Ennek következtében rendkívül fontos volt, hogy az általam eszközölt kiegészítések és változtatások optimálisan legyenek kidolgozva. Példának okáért vegyük az adatbegyűjtés 1. lépését, amely során a bekért csúcsok listáján egyesével végig megyünk, majd ellenőrizzük hogy az adott csúcs lokálisan, a gyorsítótárban, vagy távoli gépen van tárolva. A gyorsítótár korai implementációjával amennyiben egy csúcs nem volt a lokálisan tároltak listájában, a gyorsítótár elemeit egy újabb ciklussal végig nézve kerestem egyezést, találat esetén pedig a gyorsítótárazottak közé raktam a csúcs azonosítóját. Ez egyáltalán nem volt hatékony megoldás, ugyanis az ellenőrzést végző ciklus minden egyes csúcsnál, minden egyes adatbekérésnél végrehajtott, és rengeteg többletet okozott a végrehajtási időben. A továbbfejlesztett verzióban már a gyorsítótár feltöltésekor létrehozok egy úgynevezett maszk vektort, amely a gráf csúcsainak számával egyenlő hosszú, és a gyorsítótárazott csúcsok helyén 1, egyéb helyeken pedig 0 értéket tartalmaz. A maszk vektor segítségével

vel ciklus nélkül azonnal ellenőrizhetem, hogy az adott csúcs szerepel-e a gyorsítótárban, ezáltal jelentősen jobb teljesítményt elérve.

A kidolgozott megoldás működését a 6.14 ábra mutatja be vizuálisan a korábban használt minta gráfon. Az eljárás lényege, hogy azokon a gépeken, ahol megkeressük a távoli tanulási halmazba eső csúcsokat, majd azokat a (tetszőleges mélységű) szomszédságukkal együtt a lokális gyorsítótárba másoljuk át. Ezek után ezen csúcsok adatainak eléréséhez nincs szükség hálózati kommunikációhoz, a tanulási folyamat a lokális memóriában elérheti őket. Az ábra alapján könnyen látható továbbá az, hogy a tanulási halmaz lokálisan tárolt csúcsainak is lehetnek távoli szomszédai. Ennek és a szomszédság alapú mintavételezésnek hála a módszer általános esetben nem eliminálja teljesen a hálózati kommunikáció szükségletét, csupán nagyban csökkenti azt.



6.14. ábra. A PANCA működése két partíció között

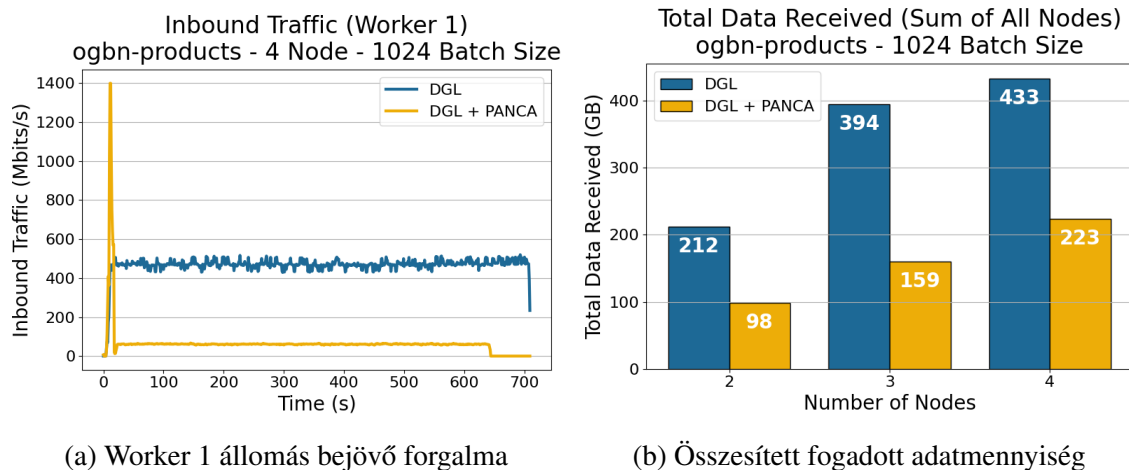
6.2. Tesztelés és értékelés

A Partíció Tudatos Csúcs Gyorsítótár (PANCA) tesztelésekor a kiinduló állapot felméréséhez hasonlóan a 4.4 fejezetben bemutatott automatizált teljesítménymérési keretrendszerre támaszkodtam. A gépekre telepített DGL keretrendszer forráskódján ejtett változtatásoktól eltekintve a tanítási környezet elemei (mint például az azt alkotó gépek, és a köztük lévő hálózati kapcsolat), valamint a tanítási paraméterek is teljesen megegyezők a korábbi mérésekkel. Ezáltal olyan adatokat tudtam kinyerni, amelyek összevethetőek a kiinduló állapoton végzett mérések eredményeivel.

Elsőként a hálózati kommunikációban elért előrelépésekre szeretnék fókuszálni, hiszen erre a területre volt a Fetching lépés 5.3 fejezetben felvezetett problémájának legfőképp negatív hatással. A 6.15a ábrán a klaszter egyetlen állomásának bejövő hálózati forgalmát láthatjuk a tanítás alatt, a DGL alapállapotában (késsel jelölve), illetve a PANCA

implementációja után (sárgával jelölve). Tisztán látható, hogy a bemeneti forgalom karakterisztikája teljesen átalakult. Míg a DGL esetén egy folytonos, 500 Mbit/s körüli értéket láthatunk, a PANCA alkalmazásakor a tanítás kezdetekor pillanatnyi felugrást tapasztalunk egészen 1400 Mbit/s sebességig, majd egy visszaesést 100 Mbit/s alá, amely változatlan marad egészen a tanítás végéig. A hirtelen, szélsőséges kiugrást a sebességben maga a gyorsítótár okozza: ebben a pillanatban töltődik fel ugyanis távoli csúcsok adataival, amihez természetesen jelentős hálózati kommunikációra van szükség. A feltöltés után azonban az eredeti bejövő forgalom csupán töredékét tapasztaljuk, hála annak, hogy a gyorsítótár nagyban csökkenti a távoli elérések számát. Észrevehetünk még egy érdekes részletet: A PANCA használata esetén a bejövő forgalom láthatóan hamarabb megszűnik. Tekintettel arra, hogy mindkét esetben 50 epoch hosszú tanítást hajtottunk végre, ez az első árulkodó jel arra, hogy a módszer a tanítási sebességre is pozitív hatással van. Fontos azonban megjegyezni, hogy a drasztikus változás a bejövő forgalomban csupán azokon a gépeken áll fenn, amelyek a tanulási halmaz elosztásakor "kölcsonkértek" csúcsokat más partíciókról, ugyanis jelenleg csak ezek alkalmazzák a gyorsítótárat.

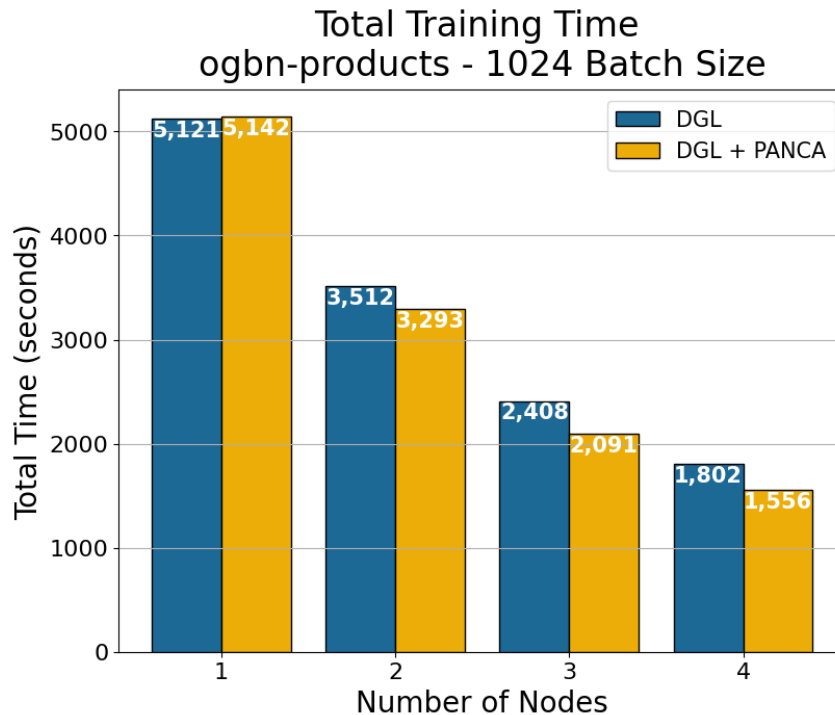
A 6.15b ábrán azt láthatjuk, hogy az 50 epoch hosszú tanítás alatt milyen mennyiségű (hány gigabyte) adatot fogadtak a klaszter gépei összesen. Természetesen egy gép esetén nincs szükség hálózati kommunikációra, így kettő, három, illetve négy gépen végrehajtott tanítások adatait látjuk. A két módszert összehasonlítva könnyen belátható, hogy a PANCA óriási adatforgalmat képes megtakarítani számunkra. A két gépes tanítás esetén egy, a három és négy gépes tanítások során pedig kettő gép alkalmazta a gyorsítótárat, ami összhangban van az összehasonlításban látható arányokkal.



6.15. ábra. A PANCA hatékonysága a hálózati kommunikációt tekintve

Miután a tanítás elosztott környezetben történő végrehajtásának legfőbb célja a tanítási idő csökkentése, ez a kapcsolódó módszerek hatékonyságának egyik legkritikusabb mérője. Az általam kidolgozott módszer eredményeit a 6.16 ábrázolja. Amint az látható,

a PANCA alkalmazása mindegyik elosztott konfiguráció esetén javulást ér el. Az eredményeket számszerűsítve négy gép esetén az eredeti 2.84x gyorsulást a PANCA 3.30x-ra javítja, amely alacsony, azonban szignifikáns különbség egy olyan folyamatnál, amely több órát vesz igénybe. Az eltérés mértékének két fő limitáló oka az, hogy a PANCA egyedül a tanítás Fetching lépésére fókuszál, illetve hogy a módszer nem eliminálja teljesen a hálózati forgalmat, csupán nagyban csökkenti azt.

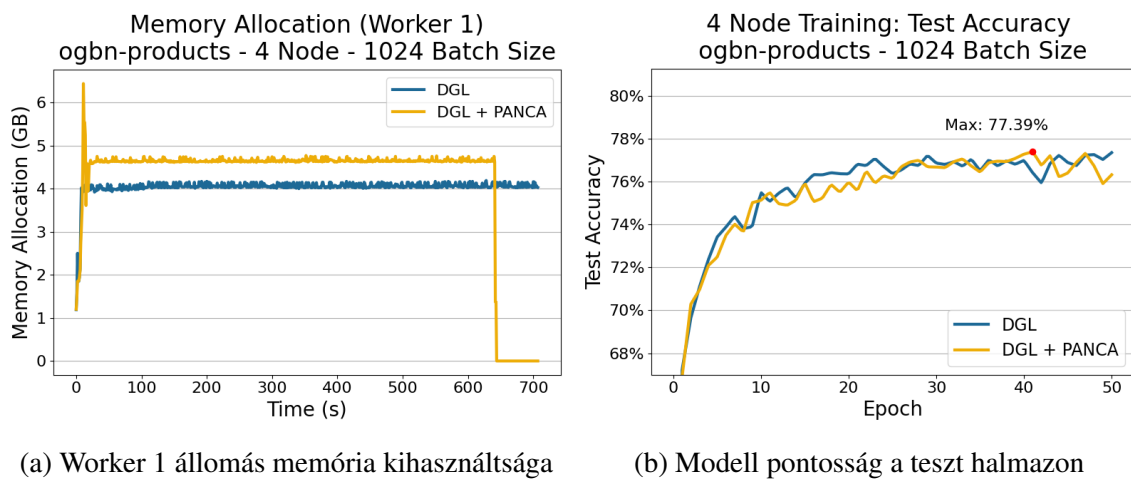


6.16. ábra. A PANCA hatása a teljes tanítási időre

A módszer kiértékelésének utolsó pontjaként szeretném bemutatni a PANCA használatának "árát". Ezt a tényezőt két szempontból is megvizsgálom, nevezetesen az erőforrás kihasználtság és a tanítási hatékonyság terén. Miután a PANCA tulajdonképpen egy kiegészítő lokális gyorsítótárat hoz létre a klaszter egyes gépein, ezek memória kihasználtsága lesz az elsődleges tényező az erőforrás szükségletek összehasonlításánál. A tanítás hatékonyságának vizsgálatánál pedig rendkívül fontos megjegyezni, hogy a folyamat bármiféle gyorsítása (szélsőséges eseteket leszámítva) kizárólag akkor előnyös, ha a betanított modell pontossága nem romlik.

A 6.17 ábra az előbb felsorolt tényezőkkel kapcsolatos eredményeket mutatja be. A gyorsítótárat alkalmazó gép memória kihasználtságát nézve, melyet a 6.17a ábra mutat be, azt láthatjuk, hogy a hálózati forgalomhoz hasonlóan a tanítás első másodperceiben a gyorsítótár feltöltése miatt egy pillanatnyi kiugrás van a memória használatban. Ezek után, a hálózati forgalommal ellentétben, az alapállapotnál enyhén magasabb kihasználtságot mutat.

ságot tapasztalunk, ami logikus, hiszen több adatot tárolunk a lokális gépen. Ezen az ábrán is észrevehető továbbá, hogy a PANCA használata megrövidíti a teljes tanítási időt. A 6.17b ábrára rátérve láthatjuk, hogy a PANCA használatának nincs negatív hatása a tanított modell pontosságára. Ez olyannyira igaz, hogy az eredetileg 5.10a ábrán bemutatott kiinduló állapotban futtatott 4 gépes tanítás 77.35%-os legjobb eredményéhez képest mindössze néhány századdal tért el a PANCA használatával futtatott tanítás legjobb eredménye, amely 77.39% lett. Az eltérés mértékéből adódóan az elhanyagolható. A pontosság változatlansága logikus, hiszen a gyorsítótár alkalmazása csupán az adatok begyűjtésének forrását változtatja meg, nem magukat az adatokat. Természetesen kisebb eltérések mindig előfordulnak mérések között, ugyanis a kötegek tartalma véletlenszerűen van kiválasztva.



6.17. ábra. A PANCA hatása egyéb paraméterekre

6.3. Kapcsolódó munkák

Ebben a fejezetben néhány olyan megoldást mutatok be, amely a PANCA-hoz hasonlóan gyorsítótárra alapozva igyekszik csökkenteni a GNN-ek elosztott tanításakor fellépő magas adatbetöltési költségeket. Az egyes módszereket megkülönböztethetjük többek között olyan tényezők alapján, mint például a gyorsítótárak tartalomkezelése (statikus vagy dinamikus), a gyorsítani kívánt művelet (tanítási vagy dedukciós szakasz), vagy a cél hardver (fő- vagy GPU-memória). További nagyon fontos szempont, hogy az adott megoldás mely paramétert - úgymint nyers áteresztőképesség, kommunikációs költség, vagy éppen erőforrás kihasználtság - hívatott javítani.

Az SCGraph [47] eljárás egy GPU kártyákon működő dinamikus gyorsítótár, amelynek célja a CPU és GPU közötti adatátvitel, valamint a nagyméretű gráfoknál megfigyelhető magas adatbetöltési költségek csökkentése. Szemben a PANCA-val, amely az elosztott

tanításra fókuszált, ez a módszer az egy állomáson végrehajtott, több GPU kártyán párhuzamosított tanításra lett tervezve. Ennek megfelelően minden GPU-n eltérő tartalmú gyorsítótár kerül létrehozásra, amelyek között NVLink [48] kapcsolat teszi lehetővé a gyors adatmozgatást. A gyorsítótárak tartalmának kiválasztásához a csúcsok tulajdonságait vizsgálja meg, nevezetesen azok fokszámát (belőlük kiinduló élek számát), ugyanis a nagyobb fokszámú csúcsok nagyobb valószínűséggel kerülnek kiválasztásra szomszédsági mintavétel során.

Míg a PANCA és az SCGraph egyaránt a tanítási szakaszra lettek tervezve, addig a PCGraph [49] egy a dedukciós szakasz optimalizációjára hivatott rendszer. A szomszédság alapú mintavételezés által okozott kihívások, nevezetesen a redundáns csúcs adatbetöltés és reprezentáció számítás leküzdésére két kulcs ötletet mutat be: Az első egy adaptív dedukciós technika, amely képes az adott gráf tulajdonságai alapján (pl. átlagos fokszám) dedukciós algoritmust választani. A második, és az én munkám szempontjából relevánsabb komponens a jellemző partició gyorsítótár. Ennek alapja, hogy a tanítási szakasszal szemben a dedukció esetén nem véletlenszerű a célcsúcsok kiválasztása, ezáltal minden köteg feldolgozásakor beazonosítható és gyorsítótárazható a teljes szükséges csúcs tartomány. A PCGraph kötegről kötegre betölti az összes szükséges adatot a GPU memóriába, jelentős csökkenést elérve ezáltal az adatbetöltési költségben, azonban a módszer alkalmazhatóságát elosztott környezetben a szerzők nem vizsgálták meg.

Az előző két kapcsolódó megoldással ellentétben, [50] az elosztott tanításra fókuszál, és a hálózati tényezőket is figyelembe veszi. A szerzők egy kétszintű dinamikus gyorsítótárat mutatnak be a gépek közötti csúcs jellemző átvitel költségének csökkentésére. A kétszintűség onnan ered, hogy a bemutatott rendszer a fő- és GPU memóriát is kihasználja, így elsődleges és másodlagos gyorsítótár is használatban van. A GPU kártyán lévő gyorsítótár az elsődleges, és itt kerülnek tárolásra olyan csúcsok jellemzői, amelyek nemrég lettek kérve, vagy nagy valószínűséggel lesznek lekérve a közeljövőben. Amikor a GPU gyorsítótár betelik, a tartalma cserélődni kezd, és a kikerült csúcsok a CPU gyorsítótárba kerülnek át. A másodlagos gyorsítótárból szintén prioritás alapján kerülnek ki a csúcsok miután az betelik. A szerzők a módszer kiértékelését három különböző hálózati konfiguráción végezték el (1/10/20 Gbps), valamint az elosztott környezetben homogén és heterogén sáv szélességet is vizsgáltak az állomások között. A saját munkámmal ellentétben azonban nem tértek ki a hálózati forgalomra kifejtett hatásra, és elsősorban a teljesítménnyel kapcsolatos eredményekre, nevezetesen a gyorsítótár gyorsaságára és tárolási rátájára, valamint a tanítási sebességre és modell pontosságra koncentráltak.

Általánosságban a PANCA elsődleges megkülönböztető tulajdonságai a statikus tartalom, a fő memória exkluzív alkalmazása, valamint az elosztott környezetre és hálózati forgalomra való koncentráltsága, amely téren a módszer jelentős eredményeket ért el.

7. Továbbfejlesztési lehetőségek és további kutatási irányok

Ahogy a kiértékelésben olvasható, a Partíció Tudatos Csúcs Gyorsítótárazás (PANCA) ígéretes eredményeket ért el. Alkalmazása igen jelentős pozitív hatást fejtett ki a hálózati kommunikációra, és a teljes tanítási időn is javítani tudott. A módszert egy erős lépésnek gondolom a gráf neurális hálózatok elosztott tanításának optimalizálása felé, és több lehetséges továbbfejlesztési illetve további kutatási irányt is látok vele kapcsolatban.

Véleményem szerint a módszer egyik jelenlegi nagyobb hiányossága, hogy csupán azokon a gépeken kerül alkalmazásra, amelyek "kölcsonkérnek" csúcsokat más gépeken tárolt partíciókból. Ennek oka, hogy a gyorsítótárazni kívánt csúcsokat kiválasztó algoritmus 1 alapvetően az adott gép tanulási halmazába tartozó, távoli csúcsokból indul ki. Lesznek azonban gépek az elosztott tanítási klaszterben, amelyek esetén minden tanulási halmazukba tartozó csúcs lokálisan van tárolva. Ahhoz, hogy az ilyen gépek hálózati szükségleteit is csökkentsük, érdemes lehet egy kiegészítő, az eredetire alapozott algoritmust kidolgozni, amely tetszőleges logika alapján kiválogat távoli csúcsokat. Egy lehetséges irány lehet itt a csúcsok fokának vizsgálata, és az az alapján történő szelekció, révén hogy a sok éllel rendelkező csúcsok nagyobb eséllyel kerülnek kiválasztásra szomszédsági mintavételezés esetén.

Természetesen egy ilyen mértékben idő- és erőforrás igényes folyamat esetén az alkalmazott módszer logikai szinten történő fejlesztésén felül kiemelten fontos a kód szintű optimalizáció is. Akár a kód kismértékű változtatása is észrevehető különbséget jelenthet a teljesítményben, hiszen a kötegfeldolgozás másodperc alatti időtartam alatt hajtódik végre.

A további kutatási irányokra rátérve, úgy gondolom hogy az egyik legfontosabb irány a teljesítménymérések diverzifikációja. Az elosztott mély gépi tanulási folyamat hatékonysága rendkívül sok paraméter függvénye, így mindenképpen érdemesnek tartom a teljesítménymérések folytatását különböző modellek, adathalmazok, konfigurációk alkalmazása mellett. Ezáltal átfogóbb képet kaphatnánk a kidolgozott módszer hatékonyságáról, és az azt befolyásoló karakterisztikáiról.

Mindemellett egy igen érdekes kérdés a meglévő módszer paramétereinek finomhangolása. Amire itt elsősorban gondolok, az a gyorsítótárazásra jelölt csúcsok mennyisége, a hálózati kommunikáció mértéke és a tanítási teljesítmény, valamint a többlet memória szükséglet közötti kompromisszum.

8. Összefoglalás

A kutatásom célja gráf neurális hálózatok tanításának vizsgálata volt elosztott környezetben. A dolgozat az elméleti alapok ismertetését, elérhető eszközök összehasonlítását, valamint új megoldások implementációs részleteit és átfogó kiértékelését tartalmazza. A főbb tartalmi fejezetek a következők:

Az első tartalmi fejezet egy áttekintést hívatott nyújtani a gráfelmélet és a neurális hálózatok kapcsolódó elméleti háttéréről. Az elméleti alapok lefektetése után ismertetésre kerültek a gráf neurális hálózatok felé irányuló kutatás motivációi, a megalkotott modellek műveletei és struktúrája, továbbá típusai és alkalmazási területei. Szó esett továbbá a neurális hálózatok tanításáról, annak kihívásairól, és azok kezelését segítő megoldásokról.

A dolgozat második tartalmi fejezete a gráf neurális hálózatok eszközkészletéről szól, nevezetesen az azok tanítására és kiértékelésére alkalmas keretrendszerekről. Általánosságban elmondható, hogy a gráf neurális hálózatok hatékony elosztott tanítása egy jelenleg is nagyon aktív kutatási terület, hiszen kevés kiforrott megoldás elérhető.

A harmadik tartalmi fejezet első részében a mérési környezetként szolgáló digitális kutatási infrastruktúra megvalósításának részleteit írtam le. Rendkívül fontos lépés volt az egész munka szempontjából, hogy meghatároztam az eszközkészletet, amellyel a környezet alapjait lefektettem. A fejezet második részében az elosztott tanítás paramétereire koncentráltam, nevezetesen a bemeneti adathalmazra, tanítási kódra, a tanított modellre, valamint a teljesítménymérések automatizálásának megvalósítására.

A negyedik tartalmi fejezet a kiinduló állapot kiértékeléséről szólt, amely során szövegesen és ábrákon keresztül ismertettem az eddigi eredményeket. A fejezet célja az volt, hogy elemezze a mért eredmények és a tanítás paramétereire illetve környezete közötti kapcsolatokat, és következtetéseket vonjon le velük kapcsolatban. Az eredmények alapján beazonosításra került az a tanítási lépés, amelyet saját megoldásommal igyekeztem optimalizálni, nevezetesen csúcsok jellemző és címke vektorainak kezelése elosztott környezetben.

Az utolsó tartalmi fejezetben bevezettem és kiértékeltem az általam létrehozott megoldást, a partíció tudatos csúcs gyorsítótárazást. Számos különböző szempontból hasonlítottam össze azt a DGL keretrendszer előző fejezetben bemutatott mérési eredményeivel. A dolgozat céljaként kidolgozott PANCA módszer igen jelentős javulást ért el a hálózati kommunikációt tekintve, valamint egy 10%-ot meghaladó csökkenést az elosztott tanítás teljes időtartamán. A gyorsítótár alkalmazása enyhe memória kihasználtsági többletet jelent, a betanított modell pontosságára pedig nincs hatással.

9. Summary

The aim of my research was to investigate the training of graph neural networks in a distributed environment. The thesis includes an overview of theoretical foundations, a comparison of available tools, and the implementation details and comprehensive evaluation of new solutions. The thesis is structured as follows:

The first content chapter provides an overview of the theoretical background of graph theory and neural networks. After laying down the theoretical foundations, the motivation for researching graph neural networks, the operations and structure of the created models, as well as their types and application areas were discussed. Additionally, the training of neural networks, its challenges, and solutions to address them were covered.

The second content chapter focuses on the toolkit for graph neural networks, specifically frameworks suitable for their training and evaluation. In general, it can be said that the efficient distributed training of graph neural networks is still a very active research area, as there are few mature solutions available.

The first part of the third content chapter describes the details of implementing the digital research infrastructure serving as the measurement environment. It was a crucial step for the entire work to define the toolkit that laid the foundations for the environment. The second part of the chapter focused on the parameters of distributed training, namely the input dataset, training code, the trained model, and the implementation of automated performance measurements.

The fourth content chapter dealt with the evaluation of the initial state, presented through both text and figures. The chapter aimed to analyze the measured results, relationships between training parameters and the environment, and draw conclusions based on them. The identified training step, which I attempted to optimize with my solution, was the handling of node feature and label vectors in a distributed environment.

In the last content chapter, I introduced and evaluated the solution I created, namely partition aware node caching (PANCA). I compared it from various perspectives with the initial measurement results of the DGL framework, presented in the previous chapter. The PANCA method, developed as the goal of the thesis, achieved significant improvements in network communication and over a 10% reduction in the total time duration of distributed training. The application of the cache implies a slight memory overhead, however it has no impact on the accuracy of the trained model.

Ábrajegyzék

2.1. Neurális hálózatok általános felépítése.....	5
2.2. Neurális hálózat belső paraméterei	6
4.3. Deep Graph Library referencia architektúra	27
4.4. <i>products</i> gráf felosztása [33].....	30
4.5. A DGL elosztott tanítási komponensei [44]	31
4.6. GraphSAGE mintavételezés és aggregáció [17]	36
4.7. Mérési környezet mappa struktúrája	41
5.8. Tanítási folyamat időbeli lebontása	43
5.9. Erőforrás kihasználtság 3 epoch alatt	44
5.10. Skálázás hatásai 50 epoch-os tanításra	44
5.11. Paraméterek hatása a tanítási folyamat időbeli lebontására	45
5.12. Gráf partícionálás elosztott tanításhoz	47
5.13. A DGL csúcs elosztása partíciók között	48
6.14. A PANCA működése két partíció között.....	52
6.15. A PANCA hatékonysága a hálózati kommunikációt tekintve	53
6.16. A PANCA hatása a teljes tanítási időre	54
6.17. A PANCA hatása egyéb paraméterekre	55

Irodalomjegyzék

- [1] J. Schmidhuber, „Deep Learning in Neural Networks: An Overview,” *Neural Networks*, vol. 61, pp. 85–117, Jan. 2015. arXiv:1404.7828 [cs].
- [2] S. Sharma, S. Sharma, and A. Athaiya, „Activation functions in neural networks,” *towards data science*, vol. 6, no. 12, pp. 310–316, 2017.
- [3] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, „A comprehensive survey on graph neural networks,” *IEEE transactions on neural networks and learning systems*, vol. 32, no. 1, pp. 4–24, 2020.
- [4] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, „Geometric deep learning: going beyond euclidean data,” *IEEE Signal Processing Magazine*, vol. 34, no. 4, pp. 18–42, 2017.
- [5] F. Liang, C. Qian, W. Yu, D. Griffith, and N. Golmie, „Survey of graph neural networks and applications,” *Wireless Communications and Mobile Computing*, vol. 2022, 2022.
- [6] S. Abadal, A. Jain, R. Guirado, J. López-Alonso, and E. Alarcón, „Computing graph neural networks: A survey from algorithms to accelerators,” *ACM Computing Surveys (CSUR)*, vol. 54, no. 9, pp. 1–38, 2021.
- [7] J. Chen, A. Pareja, G. Domeniconi, T. Ma, T. Suzumura, H. Kanezashi, T. Kaler, T. B. Schardl, C. E. Leiserson, *et al.*, „Evolving graph convolutional networks for dynamic graphs,” Aug. 19 2021. US Patent App. 16/790,682.
- [8] T. N. Kipf and M. Welling, „Semi-supervised classification with graph convolutional networks,” *arXiv preprint arXiv:1609.02907*, 2016.
- [9] F. Scarselli, A. C. Tsoi, M. Gori, and M. Hagenbuchner, „Graphical-based learning environments for pattern recognition,” in *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*, pp. 42–56, Springer, 2004.
- [10] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, „The graph neural network model,” *IEEE transactions on neural networks*, vol. 20, no. 1, pp. 61–80, 2008.
- [11] B. Sanchez-Lengeling, E. Reif, A. Pearce, and A. B. Wiltchko, „A gentle introduction to graph neural networks,” *Distill*, vol. 6, no. 9, p. e33, 2021.

- [12] J. Yang, J. Lu, S. Lee, D. Batra, and D. Parikh, „Graph r-cnn for scene graph generation,” in *Proceedings of the European conference on computer vision (ECCV)*, pp. 670–685, 2018.
- [13] J. Johnson, A. Gupta, and L. Fei-Fei, „Image generation from scene graphs,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1219–1228, 2018.
- [14] D. Marcheggiani and I. Titov, „Encoding sentences with graph convolutional networks for semantic role labeling,” *arXiv preprint arXiv:1703.04826*, 2017.
- [15] J. Zhang, X. Shi, J. Xie, H. Ma, I. King, and D.-Y. Yeung, „Gaan: Gated attention networks for learning on large and spatiotemporal graphs,” *arXiv preprint arXiv:1803.07294*, 2018.
- [16] G. Li, M. Müller, B. Ghanem, and V. Koltun, „Training graph neural networks with 1000 layers,” in *International conference on machine learning*, pp. 6437–6449, PMLR, 2021.
- [17] W. Hamilton, Z. Ying, and J. Leskovec, „Inductive representation learning on large graphs,” *Advances in neural information processing systems*, vol. 30, 2017.
- [18] W.-L. Chiang, X. Liu, S. Si, Y. Li, S. Bengio, and C.-J. Hsieh, „Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks,” in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 257–266, 2019.
- [19] A. Farkas, G. Kertész, and R. Lovas, „Parallel and distributed training of deep neural networks: A brief overview,” in *2020 IEEE 24th International Conference on Intelligent Engineering Systems (INES)*, pp. 165–170, IEEE, 2020.
- [20] T. Ben-Nun and T. Hoeffler, „Demystifying parallel and distributed deep learning: An in-depth concurrency analysis,” *ACM Computing Surveys (CSUR)*, vol. 52, no. 4, pp. 1–43, 2019.
- [21] A. Farkas, K. Póra, S. Szénási, G. Kertész, and R. Lovas, „Evaluation of a distributed deep learning framework as a reference architecture for a cloud environment,” in *2022 IEEE 10th Jubilee International Conference on Computational Cybernetics and Cyber-Medical Systems (ICCC)*, pp. 000083–000088, IEEE, 2022.
- [22] M. Héder, E. Rigó, D. Medgyesi, R. Lovas, S. Tenczer, F. Török, A. Farkas, M. Emődi, J. Kadlecik, G. Mező, Á. Pintér, and P. Kacsuk, „The past, present and future of the ELKH cloud,” *Információs Társadalom*, vol. 22, p. 128, aug 2022.

- [23] M. Fey and J. E. Lenssen, „Fast graph representation learning with pytorch geometric,” *arXiv preprint arXiv:1903.02428*, 2019.
- [24] M. Y. Wang, „Deep graph library: Towards efficient and scalable deep learning on graphs,” in *ICLR workshop on representation learning on graphs and manifolds*, 2019.
- [25] Z. Jia, S. Lin, M. Gao, M. Zaharia, and A. Aiken, „Improving the accuracy, scalability, and performance of graph neural networks with roc,” *Proceedings of Machine Learning and Systems*, vol. 2, pp. 187–198, 2020.
- [26] S. Gandhi and A. P. Iyer, „P3: Distributed deep graph learning at scale,” in *15th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 21)*, pp. 551–568, 2021.
- [27] PyG, „Graph neural network library for pytorch.” Available at https://github.com/pyg-team/pytorch_geometric (2023/11/09).
- [28] PyG, „Pyg documentation.” Available at <https://pytorch-geometric.readthedocs.io/en/latest/> (2022/11/29).
- [29] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, *et al.*, „{TensorFlow}: a system for {Large-Scale} machine learning,” in *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, pp. 265–283, 2016.
- [30] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang, „Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems,” *arXiv preprint arXiv:1512.01274*, 2015.
- [31] DGL, „Python package built to ease deep learning on graph, on top of existing dl frameworks..” Available at <https://github.com/dmlc/dgl> (2023/11/09).
- [32] DGL, „Welcome to deep graph library tutorials and documentation.” Available at <https://docs.dgl.ai/en/latest/> (2023/11/09).
- [33] W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, and J. Leskovec, „Open graph benchmark: Datasets for machine learning on graphs,” *Advances in neural information processing systems*, vol. 33, pp. 22118–22133, 2020.
- [34] T. Baruah, K. Shivdikar, S. Dong, Y. Sun, S. A. Mojumder, K. Jung, J. L. Abellán, Y. Ukidave, A. Joshi, J. Kim, *et al.*, „Gnnmark: A benchmark suite to characterize graph neural network training on gpus,” in *2021 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp. 13–23, IEEE, 2021.

- [35] OpenStack, „Open source cloud computing infrastructure.” Available at <https://www.openstack.org/> (2023/11/09).
- [36] Nvidia, „Nvidia v100 tensor core gpu.” Available at <https://www.nvidia.com/en-us/data-center/v100/> (2023/11/09).
- [37] Nvidia, „Nvidia a100 tensor core gpu.” Available at <https://www.nvidia.com/en-us/data-center/a100/> (2023/11/09).
- [38] E. Cloud, „Referencia architektúrák.” Available at <https://science-cloud.hu/referencia-architekturak> (2023/11/09).
- [39] Terraform, „Automate infrastructure on any cloud with terraform.” Available at <https://www.terraform.io/> (2023/11/09).
- [40] Ansible, „Ansible is simple it automation.” Available at <https://www.ansible.com/> (2023/11/09).
- [41] B. Rabenstein and J. Volz, „Prometheus: A next-generation monitoring system (talk),” (Dublin), USENIX Association, May 2015.
- [42] M. Chakraborty and A. P. Kundan, *Grafana*, pp. 187–240. Berkeley, CA: Apress, 2021.
- [43] G. Karypis and V. Kumar, „Metis: A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices,” 1997.
- [44] DGL, „Distributed training.” Available at <https://docs.dgl.ai/guide/distributed.html> (2023/11/09).
- [45] NVIDIA, „Nvidia collective communications library (nccl).” Available at <https://developer.nvidia.com/nccl> (2023/11/09).
- [46] L. N. Smith, „Cyclical learning rates for training neural networks,” in *2017 IEEE winter conference on applications of computer vision (WACV)*, pp. 464–472, IEEE, 2017.
- [47] Y. He, Z. Lai, Z. Ran, L. Zhang, and D. Li, „Accelerating sample-based gnn training by feature caching on gpus,” in *2022 IEEE 7th International Conference on Smart Cloud (SmartCloud)*, pp. 163–164, IEEE, 2022.
- [48] D. Foley and J. Danskin, „Ultra-performance pascal gpu and nvlink interconnect,” *IEEE Micro*, vol. 37, no. 2, pp. 7–17, 2017.

- [49] L. Zhang, Z. Lai, Y. Tang, D. Li, F. Liu, and X. Luo, „Pcgraph: Accelerating gnn inference on large graphs via partition caching,” in *2021 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)*, pp. 279–287, IEEE, 2021.
- [50] Z. Zhang, Z. Luo, and C. Wu, „Two-level graph caching for expediting distributed gnn training,” in *IEEE INFOCOM 2023-IEEE Conference on Computer Communications*, pp. 1–10, IEEE, 2023.