Óbudai Egyetem

Doktori (PhD) értekezés
tézisfüzete

Forráskódok nyelvfüggetlen analízise GPU-n, gépi tanuláson alapuló technikával

Pintér Ádám

Témavezetők:
Prof. Dr. Szénási Sándor
Prof. Dr. Ludányi-Laufer Edit

Alkalmazott Informatikai és Alkalmazott Matematikai
Doktori Iskola

Budapest, 2026. május 26.

Tartalomjegyzék

1. A kutatás előzményei

A forráskódok elemzése napjainkban egyre fontosabbá válik, mivel a technológiai fejlődés révén a programkódok szinte minden digitális rendszerben jelen vannak. A kódszintű hasonlóságvizsgálat két kiemelt alkalmazási területe a kódklónok azonosítása, amely a nagy rendszerek fejlesztésében segít elkerülni a funkcionálisan azonos kódrészek felesleges ismétlődését, valamint a plágiumdetektálás, amely a forráskódok eredetiségének ellenőrzését szolgálja.

A forráskódok klónozásának felismerése és kezelése a szoftverfejlesztés egyik lényeges területe, amely elsősorban a kódminőség fenntartása és a karbantarthatóság javítása szempontjából bír jelentőséggel. Ennek egyik lehetősége a forráskódok szövegalapú összehasonlítása, amely lehetővé teszi a programrészletek közötti hasonlóságok azonosítását. A klónok fogalmát és osztályozását elsőként Bellon és társai [1] határozták meg 2007-ben, akik részletes definíciót adtak a kódrészletek hasonlóságán alapuló kategorizáláshoz. Ez a megközelítés azóta a szakirodalom több területén továbbfejlesztésre és kiegészítésre került, megalapozva a forráskód-klónozási terminológia szabványosított keretrendszerét. [2-5]

A klónozással kapcsolatos fogalmak egyik alapegysége a kód töredék (*Code Fragment – CF*), amely alatt egy vagy több sornyi forráskódot értünk – akár kommentekkel együtt, vagy anélkül. Ezek a töredékek lehetnek kisebb utasításblokkok vagy akár teljes függvények is, amelyeket a fájlnév, kezdősor és zárósor segítségével lehet azonosítani az eredeti kódban. Ha két ilyen kódrészlet valamilyen definíció alapján hasonlónak tekinthető, akkor azok klónoknak (*Code Clone – CC*),

illetve klónpárnak (*Clone Pair – CP*) minősülnek. Több, egymáshoz hasonló kódtöredék együttesen alkot klóncsoportot (*Clone Group – CG*), amely lehetőséget ad a fejlesztők számára a kódismétlés rendszerszintű felismerésére.

A kódklónokat a hasonlóság mértéke és jellege alapján négy típusba (*Clone Type – CT*) sorolhatjuk. Az első három típus szövegalapú: az első teljes egyezést feltételez a szóközök és megjegyzések kivételével, a második már szintaktikai eltéréseket – például változónevek vagy adattípusok módosulását – is megenged, míg a harmadik típus további szerkezeti változásokat is elfogad, például új utasítások beillesztését vagy eltávolítását. A negyedik típus különösen jelentős, mivel a funkcionális hasonlóságra épül: ide azok a kódrészletek tartoznak, amelyek eltérő formában vannak megírva, mégis azonos működést valósítanak meg. Ez utóbbi típus felismerése számottevő kihívást jelent, mivel túlmutat a felszíni szövegegyezéseken, és mélyebb kódszemantikai értelmezést igényel.

Prajila 2013-ban [6] megjelent munkája átfogó keretet nyújt a forráskódklónozás felismerésének típusairól (**1.1.** táblázat). A szerző négy fő megközelítést különböztet meg: szövegalapú, lexikális (tokenalapú), szintaktikai (AST-alapú) és szemantikai (PDG-alapú) módszereket. Ezekhez kapcsolódik még kettő további típus, a metrikaalapú és a hibrid megközelítés, amelyek speciális alkalmazási környezetekre, például webes dokumentumokra vagy ritkábban használt nyelvekre (pl. Lisp) nyújtanak pontosabb detektálási lehetőséget.

A szövegalapú megközelítés közvetlenül a forráskódokat, mint szöveges dokumentumokat hasonlítja össze, transzformáció nélkül. Ez

egyszerű, alacsony komplexitású megoldás, de érzékeny minden apró változtatásra. A lexikális módszer már tokenizálja a forráskódot, és a tokenek sorozatát veti össze, így robusztusabb, és alkalmas 1. és 2. típusú klónok azonosítására is. A szintaktikai megközelítés alapja az absztrakt szintaxisfa (AST), amelyek segítségével strukturális egyezések is vizsgálhatók. Ez a módszer pontosabb, mint a szöveg- vagy token alapú technikák, mivel képes a szerkezeti azonosságokat is felismerni, így lehetővé téve a 3. típusú klónok felismerését. A legösszetettebb a szemantikai megközelítés, amely statikus programelemzésen alapul, és program függőségi gráfokat (PDG) használ a kód viselkedésének modellezésére. Ez lehetővé teszi a funkcionális hasonlóság, azaz a 4. típusú klónok felismerését is, de ez magas számítási igényű.

1.1. *táblázat. Forráskód klónok és technikák osztályozása.*

	Szöveg alapú	**Token alapú**	**AST alapú**	**PDG alapú**
Kategória	Szöveges megközelítés	Lexikális megközelítés	Szintaktikai megközelítés	Szemantikai megközelítés
Klón típusa	1. típus	1., 2. típus	1., 2., 3. típus	1., 2., 3., 4. típus
Komplexitás	$O(n)$	$O(n)$	$O(n)$	$O(n^3)$
n jelentése	kódsorok száma	tokenek száma	AST csomópontok száma	PDG csomópontok száma

2. Célkitűzések

A kutatásom elsődleges célja egy olyan új módszertani megközelítés kidolgozása és bemutatása, amely hozzájárul a forráskódok közötti hasonlóság pontosabb meghatározásához. A szoftverfejlesztésben és kódelemzésben egyre fontosabb szerepet kapnak az automatikus összehasonlító algoritmusok, amelyek segítségével lehetőség nyílik plágiumvizsgálatra, vagy különböző implementációk közötti tartalmi egyezések azonosítására, a programkód tényleges futtatása nélkül.

Kutatási céljaim:

- Karakteres összehasonlítási metrika kidolgozása, illetve annak hatékony párhuzamosítása a forráskódok, mint szöveges állományok hasonlóságának vizsgálatára.
- Hatékony párhuzamosítási és ütemezési eljárás kidolgozása egy halmaz elemeinek reflexív, szimmetrikus, nem tranzitív reláció alapú kompatibilitási osztályokba sorolásához.
- Rövid szöveges válaszok kiértékelése neurális hálózattal, figyelembe véve a válaszok szemantikáját.
- Forráskódok szemantika szerinti osztályozása.

3. Vizsgálati módszerek

3.1. Szövegalapú hasonlósági metrikák

A természetes nyelvek szövegösszehasonlításához számos jól ismert metrika áll rendelkezésre, amelyek közül a legelterjedtebbek három alapvető kategóriába sorolhatók: szerkesztési távolság alapú, token alapú és szekvencia alapú módszerek. [7], [8] Ezek mind különböző szemlélet szerint közelítik meg a hasonlóság kérdését. A szerkesztési távolság alapú metrikák arra épülnek, hogy megmérik, milyen műveletekre (pl. karakterek beszúrására, törlésére vagy cseréjére) van szükség ahhoz, hogy az egyik szövegből a másikat előállítsuk. Ez a megközelítés közvetlen és intuitív, és jól használható akkor, ha az apró szerkesztési eltérések érzékenyen befolyásolják a hasonlóság értelmezését.

A Levenshtein távolság [9] a legismertebb ilyen metrika, amelyet gyakran normalizálnak, hogy összehasonlítható értéket kapjunk különböző hosszúságú szövegek esetén. A Jaro-Winkler [10] metrika ehhez hasonló módon működik, de figyelembe veszi a karakterek elhelyezkedését is, és bünteti az olyan módosításokat, amelyek nem egyszerű transzpozícióként értelmezhetők. Mindkét algoritmus dinamikus programozáson alapul, és futásidejük a szekvenciák hosszának szorzatával arányos. Ezek a módszerek különösen alkalmasak olyan alkalmazásokban, mint a hibajavítás vagy az automatikus szöveg-összehasonlítás.

A token alapú megközelítések egy másik népszerű irányt képviselnek, ahol a szöveget kisebb egységekre, tokenekre bontják. Ezek lehetnek szavak, karakterláncok vagy n-gramok, amelyeket halmazként vagy

vektorként kezelnek az összehasonlítás során. A legismertebb token alapú metrikák közé tartozik a koszinusz hasonlóság [11], amely két szöveg vektoriális reprezentációját hasonlítja össze a köztük bezárt szög alapján. Ezenkívül a Jaccard-index [12] és a Sorensen–Dice [13] együttható szintén elterjedtek, amelyek halmazműveletek segítségével határozzák meg a közös és eltérő elemek arányát. Ezek a metrikák hatékonyak, gyorsan számíthatók, és sokszor használják őket természetes nyelvi feldolgozási feladatokban, például dokumentumok összevetésénél vagy plágiumdetektálásnál.

A harmadik kategóriát a szekvencia alapú metrikák alkotják, amelyek nem pusztán halmazként vagy karakterláncként kezelik a szöveget, hanem figyelembe veszik az elemek sorrendiségét is. A Longest Common Subsequence (leghosszabb közös részsorozat) [14] metrika célja, hogy megtalálja a két szöveg leghosszabb olyan közös részsorozatát, amely megtartja az eredeti sorrendet, de nem feltétlenül folytonos. Ez a módszer egyensúlyt teremt a strukturális és tartalmi azonosság között, ám komplexitása hasonló a Levenshtein-távolsághoz. Ez a metrika kevésbé ideális abban az esetben, ha a szöveg jelentős átrendeződése vagy strukturális módosulása történik, de előnye, hogy képes figyelmen kívül hagyni kevésbé releváns eltéréseket.

A bemutatott metrikák jól meghatározott matematikai alapokon nyugszanak, és bár különböző szempontból mérik a szövegek közötti kapcsolatot, mindegyiknek megvan a maga előnye és korlátja. A választott metrika típusát mindig az adott alkalmazási cél határozza meg. Például rövid, karakteralapú hibajavításra a Levenshtein lehet ideális, míg hosszabb dokumentumok tematikus egyezésének mérésére a koszinusz-hasonlóság nyújthat jobb eredményt. A kutatás

ezek közül az általánosan elterjedt, széles körben elfogadott metrikákra építi az összehasonlítást, hogy azokkal szemben mutassa be a saját módszer előnyeit és korlátait.

3.2. GPU adatpárhuzamos modell

A grafikus feldolgozóegység (GPU) az adatpárhuzamos feldolgozás egyik leghatékonyabb eszköze, amelyet eredetileg számítógépes grafikai feladatokra fejlesztettek ki, de mára a nagy számítási igényű tudományos és mesterséges intelligencia-alkalmazások alapvető eleme lett. A GPU fő erőssége abban rejlik, hogy több ezer párhuzamos feldolgozó egységgel rendelkezik, amelyek lehetővé teszik nagyszámú, hasonló művelet egyidejű végrehajtását. Ez a tömeges párhuzamosítás különösen előnyös olyan feladatoknál, mint például a duplikáció keresése vagy neurális hálózatok tanítása, ahol az adatok elemeinek párhuzamos vizsgálata és összehasonlítása jelentős teljesítménynövekedést eredményezhet.

A GPU-ra optimalizált feldolgozás során a feladatokat kisebb, önálló részfeladatokra kell bontani. A CUDA alapú implementációk például úgy szervezik az adatokat, hogy minden blokk az adatnak egy meghatározott részen dolgozik, és a szálak ezen belül osztják fel egymás között a részfeladatokat. Mivel a GPU hardveresen korlátozott (például egy blokk maximum 1024 szálat tartalmazhat), a feladat szétosztása körültekintő tervezést igényel. Az eljárások során felmerülhetnek olyan problémák is, mint a szinkronizáció hiánya vagy az aszinkron futás, ami miatt az egyes szálak ideiglenes eredményeit külön tárolókban kell elmenteni. A végső eredményt egy további kernel hívás segítségével kell összegyűjteni.

Az egyik legnagyobb előnye a GPU-val végzett adatpárhuzamos feldolgozásnak a drasztikus gyorsulás, ugyanazt a műveletet (például karakterek előre másolása vagy részsorozatok eltávolítása) több ezer szál végezheti egyszerre. Ez lehetővé teszi, hogy olyan feladatok, amelyek CPU-n akár órákig is eltartanának, a GPU-n másodpercek alatt lefussanak. Ezen túlmenően, ha a párhuzamosítás jól szervezett és a hardver korlátaihoz igazított, a GPU képes közel valós idejű feldolgozásra is. Ez a potenciál különösen értékes a mesterséges intelligencia, bioinformatika, adatbányászat és szövegfeldolgozás területén, ahol nagy mennyiségű adatot kell rövid idő alatt kiértékelni és feldolgozni.

3.3. Egymásba ágyazott ciklusok párhuzamosítása

Az egymásba ágyazott ciklusok feldolgozása kulcsszerepet játszik számos nagy számítási igényű területen, például numerikus módszerek, big data elemzések vagy tömbalapú adatkezelés során. Ilyen esetekben gyakran többdimenziós adatstruktúrák, például mátrixok feldolgozására van szükség, amit egymásba ágyazott ciklusokkal valósítanak meg. A ciklusok párhuzamosításának célja, hogy több szál egyszerre, egy időben végezze a számításokat, ezáltal jelentősen csökkentve a futásidőt és javítva a számítási teljesítményt.

A párhuzamosítás során alapvető kérdés, hogy a ciklusok hierarchiáján belül melyik szinten hajtsuk végre a felosztást. [15-18] Az egyik leggyakoribb módszer az „Outermost” stratégia, ahol a párhuzamosítás a legkülső ciklus szerint történik, így a belső ciklusok továbbra is szekvenciálisan futnak. Ez egyszerűen implementálható és hatékony, ha elegendően sok iterációval rendelkezik a külső ciklus.

Ugyanakkor a szálak száma korlátozott az iterációk számával, így egy kisebb ciklus esetén nem használható ki teljesen a rendelkezésre álló számítási kapacitás. Ilyen esetekben érdemes a „Nested” vagy az „Inner” stratégiák közül választani, amelyek több ciklusszinten is párhuzamosítanak, de ezek nagyobb szinkronizációs költséggel járhatnak, illetve nehezebb kontrollálni a szálak viselkedését.

A „Collapsing” stratégia ennél is tovább megy, mivel képes egyetlen ciklussá olvasztani a többszinten egymásba ágyazott ciklusokat, ezáltal jelentősen megnövelve a párhuzamosítás mértékét. Ennek az eljárásnak az előnye, hogy javítja a CPU kihasználtságot és csökkenti a ciklusokkal kapcsolatos vezérlési költségeket. A párhuzamosítás tehát nem csupán a futásidő csökkenéséhez vezet, hanem lehetővé teszi a nagyobb méretű adathalmazok kezelését is, amelyek soros feldolgozással nem lennének elfogadható időn belül feldolgozhatók. A párhuzamos cikluskezelés különösen jelentőségteljes a modern adatintenzív környezetekben, például neurális hálózatok betanításánál, szimulációknál vagy tömeges adatösszehasonlítási feladatoknál.

3.4. Szóvektortér

A természetes nyelv feldolgozásában a neurális hálózatok hatékony működésének egyik alapfeltétele, hogy a bemeneti adatok vektortérbe legyenek leképezve. A szöveges adatok azonban gyakran tartalmaznak elgépeléseket, írásjeleket, rövidítéseket vagy más formában előforduló szavakat, amelyek nehezen illeszthetők a vektortérbe. Ez különösen problematikus, ha a modell nem talál egyezést az adott szó és a vektortér vektorai között, így nem tudja az adott szóhoz a megfelelő numerikus reprezentációt társítani. Az ilyen esetek kiküszöbölésére

fontos lépés az előfeldolgozás, amely során eltávolításra kerülnek a felesleges írásjelek, egységesítésre kerülnek a szóalakok, valamint javításra a helyesírási hibák, mindezt annak érdekében, hogy a szöveg minden eleme leképezhető legyen a vektortérbe. [19], [20]

A szóvektorok előállításához kulcsfontosságú egy releváns és megfelelően előkészített korpusz [21], [22], amely lefedi a feldolgozandó szókészletet, és tartalmazza a szavak közötti szemantikai kapcsolatokat. A szóvektorokat például a word2vec [19], [23], [24] algoritmus CBOW (Continuous Bag of Words) változatával lehet előállítani, amely a szavakat környezetük alapján tanulja meg reprezentálni. [23], [25] A szóvektorokból felépülő mátrix a bemeneti adathalmaz formáját adja meg a neurális hálózat számára, amely a feldolgozás első lépése. A mátrix minden sora egy-egy szó vektora, az oszlopok pedig a vektorok dimenzióit jelentik. A mátrix méretének szabályozása érdekében általában fix sorhosszt határoznak meg – például 100 szóból álló válasz [26], [27] –, így rövidebb válasz esetén kitöltő vektorokat lehet használni, hosszabb válasz esetén pedig le kell vágni a felesleget.

A vektortérből hiányzó szavak kezelése háromféleképpen történhet: figyelmen kívül lehet hagyni őket (ami torzíthatja az értelmezést), továbbítani lehet az eredeti formájukban az felügyelőhöz (pl. manuális értékelés céljából), vagy újra lehet tervezni a korpuszt és az előfeldolgozási lépéseket (a tanítási fázisban). A cél minden esetben az, hogy a szavak konzisztens, értelmezhető formában jelenjenek meg a vektortérben, és ezzel biztosítsák a modell működőképességét. Miután a szavakhoz tartozó vektorok rendelkezésre állnak, a válaszból összeállított mátrix közvetlenül betölthető például egy konvolúciós

neurális hálózat bemenetére, amely így képes a szöveg jelentésének értelmezésére és kiértékelésére.

3.5. Konvolúciós neurális hálózatok

A konvolúciós neurális hálózatok (CNN-ek) rétegzett felépítésükről ismertek, ahol a jellemzők kinyerése konvolúciós rétegeken keresztül történik. Ezt gyakran pooling rétegek követik, amelyek a jellemzőtér méretének csökkentésére szolgálnak. A kimeneti réteg előtt rendszerint teljesen összekapcsolt rétegek is helyet kapnak, amelyek a döntéshozatalt segítik. Az ilyen hálózatok kialakításában többféle tervezési döntés játszik szerepet, beleértve a kernel- vagy ablakméret megválasztását, a használt szűrők számát, valamint a különböző hiperparaméterek konfigurációját. [22]

Ezeket a hálózatokat a kezdetekben a képfeldolgozás során használták, azonban a szövegek vektorizációjának elterjedését követően, már lehetőség nyílt ezek osztályozására is. A vektortér esetén használt konvolúciós neurális hálózatok rendszerint csak egyetlen konvolúciós réteggel rendelkeznek [28], [29] (eltérően a gépi látástól, ahol ezek jellemző száma meghaladja a hármat). Ezzel elveszítve az eredeti intuíciók kinyerését, amelyek magasabb absztrakciós szinten egyébként se segítik a feldolgozást, ellentétben a képfeldolgozással. [30-33]

4. Új tudományos eredmények

4.1. 1. Téziscsoport – Nem gépi tanuláson alapuló összehasonlító módszerek

4.1.1. 1.1 tézis

Újfajta mérőszámot dolgoztam ki forráskódok szövegalapú összehasonlítására, amely a leghosszabb közös részstring (Longest Common Substring - LCS) keresés iteratív alkalmazásán alapul. Ezen mérőszám és egy megfelelően választott határérték segítségével egy új módszert dolgoztam ki, amelyik két forráskódról képes megállapítani, hogy azok ugyanazt a feladatot oldják-e meg. A módszert 4 különböző feladatot megoldó, 400 darab ember által készített forráskód egymással való összehasonlításával validáltam, amely alapján igazolni tudtam, hogy az jelentősen magasabb F1 score értéket ért el, mint a meglévő hasonló célú módszerek.

A kifejlesztett leghosszabb közös részstring alapú (**4.1.** algoritmus) módszer nem igényli a forráskódok előfeldolgozását, alternatív reprezentációját, kezeli a strukturális, illetve sorrendiségből adódó különbségeket, továbbá programozási nyelvtől függetlenül alkalmazható. Az elért eredmények bizonyítottan relevánsabb hasonlósági érték képviselnek a gyakori, természetes nyelveknél használt metrikákkal szemben, amit a valós példákon keresztül is igazoltam.

A tézishez kapcsolódó saját publikáció: [S1]

4.1. algoritmus. *A kidolgozott módszer alapját adó leghosszabb közös*

részstring algoritmusa.

```
Bemenet:   S1 – első szöveg, S2 – második szöveg,
           |S1| – első szöveg hossza, |S2| – második szöveg hossza
Kimenet:   a leghosszabb közös részstring hossza
 1: function LCS(S1, S2, |S1|, |S2|)
 2:     lCS ← 0
 3:     for i ← 0 to |S1| do
 4:         for j ← 0 to |S2| do
 5:             k ← 0
 6:             while (i + k) < |S1| ∧ (j + k) < |S2| ∧ S1^(i+k) = S2^(j+k) do
 7:                 k ← k + 1
 8:             end while
 9:             lCS ← max(lCS, k)
10:         end for
11:     end for
12:     return lCS
13: end function
```

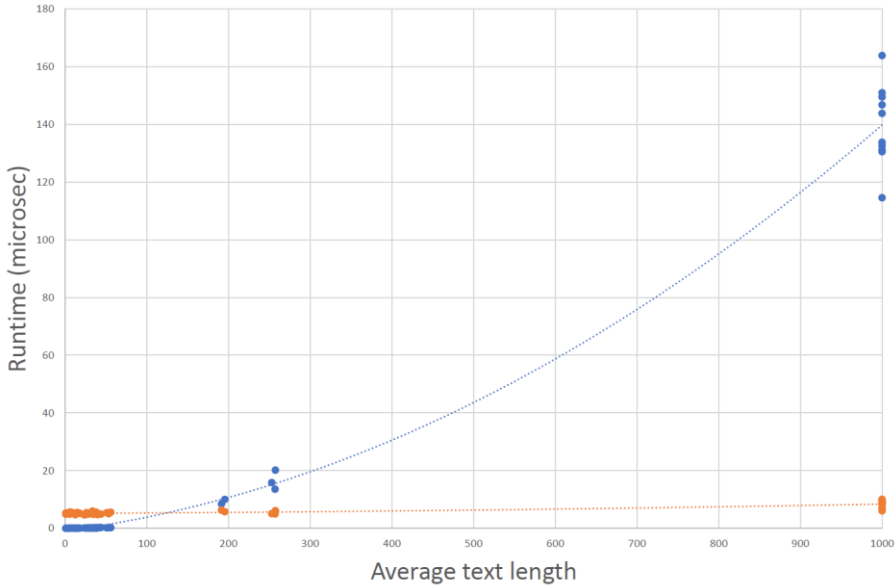
4.1.2. 1.2 tézis

Kidolgoztam egy újszerű adatpárhuzamos módszert szövegek hasonlóságának mérésére. A módszer a leghosszabb közös részstring alapú összehasonlító módszer iteratív megvalósításán alapul, az abban található erőforrásigényes összehasonlításokat, illetve a megtalált részstringek eltávolítását azonban jóval hatékonyabban, adatpárhuzamos módon végzi el. A dinamikus programozás alapú szekvenciális és a GPU-n implementált adatpárhuzamos módszerek összehasonlítása alapján megállapítottam, hogy a nagyon rövid szövegek kivételével az általam kidolgozott módszer jelentősen kisebb futásidőt igényel, miközben azonos pontosságot ér el.

Az előzőleg bemutatott leghosszabb közös részstring alapú hasonlósági metrika alkalmazása meglehetősen időigényes. A leghosszabb közös részstringek keresését ugyanis mindaddig kell ismételni, amíg találunk ilyen elemet a két forráskódban. Ezzel a

módszerrel már pár száz valós forráskód összehasonlítása is nagyon magas futásidőt eredményez, ami a gyakorlati használhatóságot erősen korlátozza, ezért kidolgoztam a metrika GPU-alapú adatpárhuzamos módszerét. Az adatpárhuzamos megvalósítás tekintetében méréseket végeztem, amik alapján megállapítottam, hogy lényegesen jobb futásidejű eredmény érhető el, illetve, azt is, hogy körülbelül ~130 karakternél hosszabb szövegek esetén válik gyorsabbá ez, a CPU-s implementációhoz képest (**4.1.** ábra).

A tézishez kapcsolódó saját publikáció: [S2]



4.1. ábra. *A CPU és a GPU implementáció futásidejének összehasonlítása a bemeneti szövegek átlagos hossza alapján. A kék pontok a CPU, a piros pontok a GPU méréseit jelzik.*

4.1.3. 1.3 tézis

Kidolgoztam egy újszerű particionálási eljárást, ami alkalmas egy halmaz elemeinek reflexív, szimmetrikus, nem tranzitív reláció alapú kompatibilitási osztályokba sorolásának hatékony párhuzamosítására. Az általam kidolgozott módszer a szükséges összehasonlító műveleteket a létező hasonló megoldásokhoz képest jobb terheléselosztással valósítja meg. A módszer igazolásához számos tesztet futtattam, amelyek igazolták, hogy egy minimális elemszám felett az általam kidolgozott particionálás jelentősen kisebb futásidőhöz vezet, mint a meglévő hasonló felosztó módszerek.

Az egymásba ágyazott ciklusok párhuzamosításának egyik lehetősége, ha a külső ciklus szerint bontjuk szét a feladatot részfeladatokra. Ha olyan kompatibilitási osztályokat szeretnénk létrehozni az adatkészlet elemei között, amelyek a hasonló elemeket egy osztályba sorolják (vagyis olyan műveletet hajtunk végre, amely reflexív, szimmetrikus, de nem tranzitív), akkor lehetőség van hatékonyabb párhuzamosításra, mint az egyszerű futóváltozó szerinti egyenletes szétosztás.

A kidolgozott párhuzamosítási eljáráshoz (**4.2.** algoritmus) ismerni kell az adathalmaz méretét, a partíciók célszámát, amik alapján meghatározható az optimális iterációk száma egy partícióban. Ezek alapján az összes partícióra ki lehet számítani a külső ciklus alsó (beleértve) és a felső (kizáró) indexeit. Szintetikusan generált tesztekkel igazoltam az újonnan kidolgozott algoritmus helyességét, illetve a futásidő csökkenését. Továbbá megállapítottam, hogy több mint 10 000 elem esetén érdemes az általam készített párhuzamosítási eljárást alkalmazni.

A tézishez kapcsolódó saját publikáció: [S3]

4.2. algoritmus. *A kidolgozott párhuzamosított megvalósítás.*

Bemenet: S – adathalmaz, N – elemek száma az adathalmazban
Kimenet: S – a feldolgozott adathalmaz

```
1:  function JavítottPárhuzamosMegvalósítás(S, N)
2:      P = NumberOfProcessors()        ▷ Optimális szálak száma
3:      T = CreateThreads(P)            ▷ P szál létrehozása
4:      a = -1
5:      b = 2N + 1
6:      K_alsó = 0
7:      I_alsó = 0
8:      for p ← 1 to P do
9:          c = (P(N−K_alsó)(N−K_alsó+1)−N(N+1))/P − N² − N
10:         K_felső = (−b+√(b²−4ac))/2a
11:         I_felső = ⌊K_felső⌋         ▷ felső (exkluzív) index
12:         T[p]=ThreadProcess(I_alsó, I_felső, S, N)
13:         K_alsó = K_felső
14:         I_alsó = I_felső
15:     end for
16:     WaitAll(T)
17:     return S
18: end function

19: function ThreadProcess(alsó, felső, S, N)
20:     for i ← alsó to felső do
21:         for j ← i + 1 to N do
22:             if S[i].T1 R S[j].T1 then    ▷ R a reláció két elem között
23:                 S[i].T2 =S[i].T2 ∪ {S[j]}
24:                 S[j].T2 =S[j].T2 ∪ {S[i]}
25:             end if
26:         end for
27:     end for
28:     return S
29: end function
```

A pszeudókódban az adathalmaz objektumokat tartalmaz, melyek rendelkeznek T_1 és T_2 tulajdonságokkal. A T_1 tulajdonság alapján kerülnek összehasonlításra az objektumok és a T_2 tulajdonságokba kerülnek összekapcsolásra a hasonló elemei a halmaznak az objektum referencia által.

4.2. 2. téziscsoport – Gépi tanuláson alapuló összehasonlító módszerek

4.2.1. 2.1 tézis

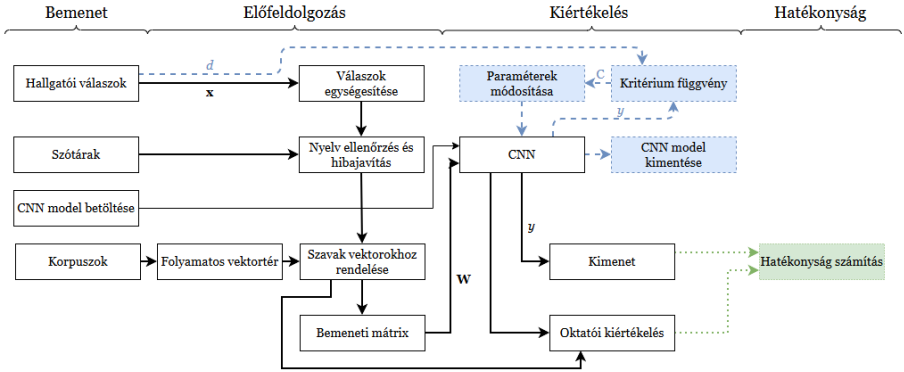
Kidolgoztam egy módszert, amely alkalmas szabadszavas rövid szöveges válaszok kiértékelésére. A módszer az eredetileg képfeldolgozási területen használt konvolúciós neurális hálózatokon alapul, kiegészítve azt speciális elő- és utófeldolgozó lépésekkel, amelyek jelentősen növelik a hálózat hatékonyságát a megadott területen. A valódi adatokon történő vizsgálatok kimutatták, hogy megfelelő méretű tanítási adatbázis megléte esetén az általam javasolt modell nagy pontosságot ért el.

A feladatokat típusuk szerinti osztályozás esetén két nagy csoportra lehet bontani: passzív feladatok és aktív feladatok. A rövid szöveges válaszok, melyek szabadon megfogalmazhatóak az aktív feladatok közé tartoznak és validálásuk kifejezetten nehéz, tekintettel annak a szabad megfogalmazására. Az elérhető módszerek döntő többsége előre definiált szabályrendszer segítségével értékeli ki a válaszokat, mellőzve azok szemantikáját, tényleges megértését.

Az általam kidolgozott modell (**4.1.** ábra) alapja a konvolúciós neurális hálózat, aminek köszönhetően mellőzhető a megengedett (vagy tiltott) kulcsszavak, kifejezések definiálása, illetve az elvárt struktúrák megadása a felügyelő által minden feladat tekintetében. Ennek a megközelítésnek a másik előnye a modell nyelvfüggetlensége, így lehetővé téve annak működésének helyességének validálását eltérő nyelven írt teszthalmazokon. Az elért eredmények alapján kijelenthető, hogy a modell teljesítette a kitűzött célt, és bár a nagy nyelvi modellek

folyamatos fejlődése jobb, azonban összességében erőforrásigényesebb kiértékelési rendszerek megjelenéséhez vezethetnek.

A tézishez kapcsolódó saját publikáció: [S4]



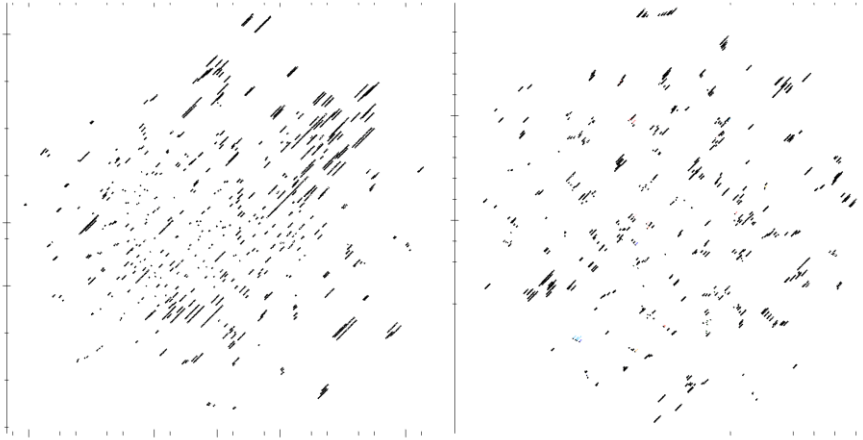
4.1. ábra. *Rövid szöveges válaszok feldolgozásának modellje.*

4.2.2. 2.2 tézis

Kidolgoztam egy újszerű módszert, amely alkalmas forráskódok előre megadott osztályokba sorolására. Megoldásom a természetes nyelvi feldolgozáshoz kidolgozott szóvektor modellt adaptálja forráskód tokenekre, továbbá bevezet egy erre a célra specializált adattisztítási eljárást. Az így felépített szóvektor tér és egy konvolúciós neurális hálózat segítségével az általam kidolgozott modell képes forráskódokhoz valószínűségeket rendelni aszerint, hogy azok a tanítási adatbázis melyik osztályába illeszkednek. A valós programokon futtatott validáció igazolja, hogy a módszer nagy pontossággal képes a forráskódokról megállapítani, hogy azok melyik feladatot oldják meg.

A fejlesztők számára kifejezetten hasznos, ha funkcionalitás alapján tudnak keresni a kódbázisban, így elkerülve a kódismétlést, de egyetemi számonkérések során is nélkülözhetetlen lehet a leadott megoldások feladatok szerinti automatikus osztályozása során. Annak érdekében, hogy a forráskódok szemantikai szempontok szerinti rendszerezése megvalósulhasson, kidolgoztam a természetes nyelveknél sikeresen alkalmazott szóvektor és konvolúciós neurális hálózat alapú modell forráskódokon használható változatát. A vektortér minőségét kvalitatív módszerekkel vizsgáltam meg, illetve azt javítottam az adatok tokenizált előfeldolgozásával (**4.2.** árba). A modellt valós adatokkal teszteltem, amivel igazoltam a módszer használhatóságát.

A tézishez kapcsolódó saját publikációk: [S5], [S6], [S7]



*(**A**) A nyers adatok alapján készített vektortér, ahol a szavak csoportosulása nem jellemző.*

*(**B**) A megtisztított adatok alapján készített vektortér, ahol a szavak jelentésük szerint csoportosulnak.*

***4.2. ábra.** A nyers, illetve megtisztított adatok első 500 eleme alapján készített vektorterek vizualizációja.*

5. Az eredmények hasznosítási lehetősége

A forráskódok analízise, azon belül is azok osztályozás egy fontos terület, nem csak a disszertációban tárgyalt aspektusai esetén. Számos megoldás létezik mind a forráskódok karakter alapú összehasonlítására, mind azok tartalma szerinti osztályozására.

Az általam kidolgozott leghosszabb közös részstring alapú metrika, illetve annak GPU-alapú párhuzamosított variánsa alkalmas számonkérések során a plágiumok nagy pontossággal való megállapítására. Ennek alkalmazhatóságához szorosan kapcsolódik az egymásba ágyazott ciklusok indexfüggő párhuzamosításnak lehetősége, mikor a GPU, mint feldolgozó egység nem áll rendelkezésre.

Az általam kidolgozott rövid szöveges válaszokat kiértékelni képes konvolúciós neurális hálózat alapú modell az oktatás során alkalmazható a számonkéréseknél, tehermentesítve az oktatót.

Az általam kidolgozott szóvektor alapú tokenizálás és konvolúciós neurális hálózatra épülő megközelítés alkalmas a forráskódok osztályozására, ezzel elősegítve például a forráskódok feladatok szerinti csoportosítását.

6. Irodalmi hivatkozások listája

[S1] Á. Pintér and S. Szénási, **"Longest Common Subsequence-based Source Code Similarity"**, *2023 IEEE 17th International Symposium on Applied Computational Intelligence and Informatics (SACI)*, Timisoara, Romania, 2023, pp. 000123-000128, doi: 10.1109/SACI58269.2023.10158551

[S2] Á. Pintér and S. Szénási, **"GPU Acceleration of Longest Common Substrings Algorithm"**, *2023 IEEE 17th International Symposium on Applied Computational Intelligence and Informatics (SACI)*, Timisoara, Romania, 2023, pp. 000189-000194, doi: 10.1109/SACI58269.2023.10158638

[S3] Á. Pintér and S. Szénási, **"Index Dependent Nested Loops Parallelization with an Even Distributed Number of Steps"**, *INFORMATICA-JOURNAL OF COMPUTING AND INFORMATICS 45*: 4 pp. 493-506., 16 p. (2022), doi: 10.31449/inf.v45i4.3130

[S4] Á. Pintér, B. Schmuck, and S. Szénási. **"Short text evaluation with neural network."** *Pollack Periodica* 13.3 (2018): 107-118. doi: 10.1556/606.2018.13.3.11

[S7] Á. Pintér and S. Szénási, "**Automatic Analysis and Evaluation of Student Source Codes,**" *2020 IEEE 20th International Symposium on Computational Intelligence and Informatics (CINTI)*, Budapest, Hungary, 2020, pp. 000161-000166, doi: 10.1109/CINTI51262.2020.9305819

[S6] Á. Pintér and S. Szénási, "**Comparison of Source Code**

Storage Methods," 2019 IEEE 19th International Symposium on Computational Intelligence and Informatics and 7th IEEE International Conference on Recent Achievements in Mechatronics, Automation, Computer Sciences and Robotics (CINTI-MACRo), Szeged, Hungary, 2019, pp. 000231-000236, doi: 10.1109/CINTI-MACRo49179.2019.9105260

[S7] Á. Pintér and S. Szénási, **"Classification of source code solutions based on the solved programming tasks,"** *2018 IEEE 18th International Symposium on Computational Intelligence and Informatics (CINTI)*, Budapest, Hungary, 2018, pp. 000277-000282, doi: 10.1109/CINTI.2018.8928201.

6.1. A tézispontokhoz nem kapcsolódó tudományos közlemények

[S8] N. Neumann and Á. Pintér, "**Solving ARC with non-procedural program induction,**" *2023 IEEE 23rd International Symposium on Computational Intelligence and Informatics (CINTI)*, Budapest, Hungary, 2023, pp. 000271-000276, doi: 10.1109/CINTI59972.2023.10382009

[S9] A. Szabó and Á. Pintér, "**Galaxy detection and classification in sky images with neural network,**" *2023 IEEE 23rd International Symposium on Computational Intelligence and Informatics (CINTI)*, Budapest, Hungary, 2023, pp. 000277-000280, doi: 10.1109/CINTI59972.2023.10382059

6.2. További tudományos közlemények

[1] S. Bellon, R. Koschke, G. Antoniol, J. Krinke and E. Merlo, ”*Comparison and Evaluation of Clone Detection Tools*”, Transactions on Software Engineering, 33(9):577-591 (2007).

[2] C K. Roy, J. R. Cordy, R. Koscheke. ”*Comparison and Evaluation of Code Clone Detection Techniques and Tools: A Qualitative Approach*”, Science of Computer Programming, 44(7):470-495 (2009).

[3] B. Kumar, S. Singh, ”*Code Clone Detection and Analysis Using Software Metrics and NEural Network-A Literature Review*”, Internal Journal of Computer Science Trends and Technology, 3(4):127-132 (2015).

[4] M. White, M. Tufano, C. Vendome, D. Poshyvanyk, ”*Deep learning Code Fragments for Code Clone Detection*”, Automated Software Engineering, ISBN: 978-1-4503-3845-5/16/09, Singapure (2016).

[5] E. Kodhai, S. Kanmani, ”*Method-level code clone detection through LWH (Light Weight Hybrid) approach*”, Journal of Software Engineering Research and Development (2014).

[6] P. Prem, ”*A Review on Code Clone Analysis and Code Clone Detection*”, International Journal of Engineering and Innovative Technology, 2(12):43-46 (2013).

[7] Sun, Y., Ma, L., & Wang, S.”*A comparative evaluation of string similarity metrics for ontology alignment.*” Journal of Information &Computational Science, 12(3), 957-964. (2015). doi: 10.12733/jics20105420

[8] M. Agrawal and D. K. Sharma, ”*A state of art on source code plagiarism detection,*” 2016 2nd International Conference on Next Generation Computing Technologies (NGCT), Dehradun, India, 2016, pp. 236-241, doi: 10.1109/NGCT.2016.7877421.

[9] В. И. Левенштейн (1965). ”Двоичные коды с исправлением выпадений, вставок и замещений символов [Binary codes capable of correcting deletions, insertions, and reversals]”. Доклады Академии Наук СССР (in Russian). 163 (4): 845–848. Appeared in English as: Levenshtein, Vladimir I. (February 1966). “Binary codes capable of correcting deletions, insertions, and reversals”. Soviet Physics Doklady. 10 (8):707–710.

[10] Winkler, W. E.”*String Comparator Metrics and Enhanced Decision Rules in the Fellegi-Sunter Model of Record Linkage*”. Proceedings of the Section on Survey Research Methods. American Statistical Association: 354–359. (1990).

[11] N. Cheng, Z. Yu and K. Wang, ”String similarity computing based on position and cosine,” 2017 7th IEEE International Conference on Electronics Information and Emergency Communication (ICEIEC), Macau, China, 2017, pp. 256-261, doi: 10.1109/ICEIEC.2017.8076557.

[12] Jaccard, P.”Étude comparative de la distribution florale dans une portion des Alpes et des Jura.” *Bull Soc Vaudoise Sci Nat*, *37*, 547-579. 1901.

[13] Carass, A.; Roy, S.; Gherman, A.; Reinhold, J.C.; Jesson, A.; et al. "Evaluating White Matter Lesion Segmentations with Refined Sørensen-Dice Analysis". *Scientific Reports. 10 (1): 8242. Bibcode:2020NatSR..10.8242C*. 2020, doi:10.1038/s41598-020-

64803-w

[14] L. Bergroth, H. Hakonen and T. Raita, ”*A survey of longest common subsequence algorithms,”* Proceedings Seventh International Symposium on String Processing and Information Retrieval. SPIRE 2000, A Curuna, Spain, 2000, pp. 39-48, doi: 10.1109/SPIRE.2000.878178.

[15] Beata Bylina and Jaroslaw Bylina. ”Strategies of parallelizing nested loops on the multicore architectures on the example of the wz factorization for the dense matrices.” Proceedings of the Federated Conference on Computer Science and Information Systems, pages 629–639, 2015. doi: 10.15439/2015f354.

[16] Beata Bylina and Jaroslaw Bylina. ”*Parallelizing nested loops on the intel xeon phion the example of the dense wz factorization.*” Proceedings of the Federated Conference on Computer Science and Information Systems, 8:655–664, 2016. doi:10.15439/2016f436.

[17] Adrian Jackson and Orestis Agathokleous. “*Dynamic loop parallelisation.*” arXiv:1205.2367, 2012.

[18] M.E. Wolf and M.S. Lam. ”*A loop transformation theory and an algorithm to maximize parallelism.*” IEEE Transactions on Parallel and Distributed Systems, 2(4):452–471, 1991. doi: 10.1109/71.97902.

[19] Document-specific word2vec, Training corpus, url: http://kbpedia.com/use-cases/document-specific-word2vec-training-corpuses/, (utoljára megtekintve: 2025.06.01).

[20] Norvig P., ”*How to Write a Spelling Corrector,*”, 2016. url: https://norvig.com/spell-correct.html (utoljára megtekintve: 2025.06.01)

[21] Ye, C., Yang, Y., Fermuller, C., & Aloimonos, Y. (2017). ”*On the importance of consistency in training deep neural networks*”. arXiv preprint arXiv:1708.00631. 2017.

[22] Dudek-Dyduch, E., Tadeusiewicz, R., & Horzyk, A. ”*Neural network adaptation process effectiveness dependent of constant training data availability*.” Neurocomputing, 72(13-15), 3138-3149. 2009.

[23] Liu J., Meng F., Yong Z., Liu B. ”*Character-Level neural networks for short text classification*,” 2017 International Smart Cities Conference (ISC2), Wuxi, China, 14-17 Sept 2017.

[24] word2vec project website, url: https://code.google.com/archive/p/word2vec/ (utoljára megtekintve: 2025.06.01)

[25] K. N. M. Ngafidin, S. T. Safitri and D. M. Kusumawardani, ”*A Study on SME Traditional Food Product Feedback by Leveraging Word2vec,*” 2024 Beyond Technology Summit on Informatics International Conference (BTS-I2C), Jember, East Java, Indonesia, 2024, pp. 71-76, doi: 10.1109/BTS-I2C63534.2024.10942267.

[26] K. Yoon, “*Coonvolutional Neural Networks for Sentence Classification*”, Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, pp. 1746-1751, 25 August 2014.

[27] D. Britz, “Understanding Convolutional Neural Networks for NLP”, url: http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/ (utoljára megtekintve: 2025.06.01)

[28] Huy N., Minh-Le N. ”*A Deep Neural Architecture for Sentence-level Sentiment Classification in Twitter Social Networking*”, Conference of the Pacific Association for Computational Linguistics, Yangon, Maynmar, August 16 – 18, 2017

[29] Huy T. N., Minh-Le N. ”*Sentence Modeling with Deep Neural Architecture using Lexicon and Character Attention Mechanism for Sentiment Classification*”, Proceedings of the The 8th International Joint Conference on Natural Language Processing, Taipei, Taiwan, November 27 - December 1, 2017, pp. 536-544.

[30] Kim Y. ”*Convolutional neural networks for sentence classification,*” Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, Doha, Qatar, 25-29 October 2014, pp. 1746–1751

[31] Severyn A., Moschitti A. ”*Learning to rank short text pairs with convolutional deep neural networks,*” Proceedings of the 38th Internal ACM SIGIR Conference on Research and Development in Information Retrieval, Santiago, Chile, 9-13 Aug 2015, pp. 373–382

[32] N. D. Q. Bui, L. Jiang, Y. Yu, “Cross-Language Learning for Program Classification using Bilateral Tree-Based Convolutional Neural Networks”, arXiv:1710.06159, 2017.

[33] S. Gilda, “*Source code classification using Neural Networks*”, IEEE 14th International Joint Conference on Computer Science and Software Engineering, 12-14 July 2017.