

Budapesti Műszaki és Gazdaságtudományi Egyetem

Villamosmérnöki és Informatikai Kar

Czotter Benedek

Klaszterezési stratégiák alkalmazása RAG rendszerek hatékonyságának növelésére

Tudományos Diákköri Konferencia Dolgozat

Konzulens

Dr. Szűcs Gábor

BUDAPEST, 2025

Tartalomjegyzék

[Abstract 4](#_Toc211073311)

[1 Bevezetés 5](#_Toc211073312)

[1.1 A kutatás célja 5](#_Toc211073313)

[1.2 RAG-rendszerek szerepe a modern NLP-ben 5](#_Toc211073314)

[1.3 Hatékonysági kérdések nagy dokumentumkorpuszok esetén 5](#_Toc211073315)

[2 Elméleti háttér 6](#_Toc211073316)

[2.1 A RAG (Retrieval-Augmented Generation) elmélete 6](#_Toc211073317)

[2.1.1 RAG architektúra felépítése 6](#_Toc211073318)

[2.1.2 Dokumentum-visszakeresés alapjai 6](#_Toc211073319)

[2.2 Embeddingek és vektorterek 6](#_Toc211073320)

[2.2.1 Sentence-transformerek működése és hatásuk a reprezentációra 6](#_Toc211073321)

[2.2.2 Távolságmértékek 6](#_Toc211073322)

[2.2.3 Vektorindexelési módszerek áttekintése 6](#_Toc211073323)

[2.3 Klaszterezés az információkinyerésben 6](#_Toc211073324)

[2.3.1 Klaszterezés célja 6](#_Toc211073325)

[2.3.2 Offline és online klaszterezés 6](#_Toc211073326)

[2.3.3 River könyvtár és saját online Kmeans fejlesztése 6](#_Toc211073327)

[3 Adathalmaz és előfeldolgozás 7](#_Toc211073328)

[3.1 A SQuAD adathalmaz 7](#_Toc211073329)

[3.2 Szövegfeldolgozás és chunking 7](#_Toc211073330)

[3.3 Embedding generálás 7](#_Toc211073331)

[4 Klaszterezési módszerek implementációja és integrálása a retrieval folyamatba 8](#_Toc211073332)

[4.1 Offline klaszterezés 8](#_Toc211073333)

[4.2 Online klaszterezés 8](#_Toc211073334)

[5 Eredmények és összehasonlítás 9](#_Toc211073335)

[6 Összegzés és kitekintés 10](#_Toc211073336)

[Utolsó simítások 11](#_Toc211073337)

[Irodalomjegyzék 12](#_Toc211073338)

[Függelék 13](#_Toc211073339)

Összefoglaló

A Retrieval-Augmented Generation (RAG) modellek az utóbbi években kulcsszerepet kaptak a nagynyelvi modellek (LLM-ek) tudásbővítésében és naprakész információkkal való ellátásában. E rendszerek lényege, hogy a nyelvi modell válaszadási folyamata előtt releváns dokumentumokat vagy dokumentumrészleteket („chunkokat”) keresnek ki egy nagyméretű tudásbázisból embedding-alapú hasonlóságmérés segítségével. A jelenlegi gyakorlatban a legtöbb RAG-megoldás a teljes embedding-halmazt közvetlenül használja a hasonlósági kereséshez, ami azonban pontossági, redundancia- és futásidőbeli korlátokat eredményezhet. A dolgozat célja annak vizsgálata, hogy klaszterezési technikák integrálásával miként növelhető a RAG-rendszerek hatékonysága, pontossága és robusztussága; hiszen hogyha a dokumentumok embeddingjeit először klaszterekbe rendezzük, majd a felhasználói lekérdezés embeddingjét először klaszterszinten vetjük össze velük, a keresés jelentősen gyorsabbá és relevánsabbá válhat. A dolgozat bemutatja a különböző klaszterezési algoritmusok teljesítményét és alkalmazhatóságát eltérő embedding-reprezentációkon, valamint azt, hogy hogyan lehet lehetővé tenni új dokumentumok folyamatos integrálását a tudásbázisba a teljes újraklaszterezés költsége nélkül. A kutatás eredménye egy klaszterezésen alapuló, adaptív retrieval-pipeline kialakítása, amely a hagyományos RAG-architektúrákhoz képest magasabb hatékonyságot, jobb pontosságot és fokozott robusztusságot biztosít. A kutatás eredményei nemcsak elméleti szempontból járulnak hozzá a RAG-rendszerek fejlődéséhez, hanem gyakorlati alkalmazásokban is közvetlenül hasznosíthatók lesznek.

Abstract

Retrieval-Augmented Generation (RAG) models have become key components in recent years for enhancing the knowledge base of large language models (LLMs) and providing them with up-to-date information. The core idea behind these systems is that, before the language model generates a response, they retrieve relevant documents or document segments (“chunks”) from a large knowledge base using embedding-based similarity search. In current practice, most RAG solutions directly use the entire embedding set for similarity search, which can lead to limitations in accuracy, redundancy, and runtime performance. The aim of this thesis is to investigate how the integration of clustering techniques can improve the efficiency, accuracy, and robustness of RAG systems. By first organizing document embeddings into clusters and then comparing the query embedding at the cluster level, the retrieval process can become significantly faster and more relevant. The thesis explores the performance and applicability of various clustering algorithms on different embedding representations and examines how new documents can be continuously integrated into the knowledge base without the cost of full re-clustering. The outcome of the research is the design of a clustering-based adaptive retrieval pipeline that provides higher efficiency, better accuracy, and increased robustness compared to traditional RAG architectures. The results of this research contribute not only to the theoretical advancement of RAG systems but are also directly applicable in practical implementations.

# 1 Bevezetés

## A kutatás célja

A mesterséges intelligencia fejlődésének egyik legmeghatározóbb területe az utóbbi években a természetes nyelv feldolgozás (Natural Language Processing, NLP) volt. A nagyméretű nyelvi modellek (Large Language Models, LLM-ek) képesek emberi szintű szövegértésre és -generálásra, azonban működésük korlátozott, ha a feladatukhoz szükséges információ nem szerepel a modell paramétereiben. A Retrieval-Augmented Generation (RAG) megközelítés ezt a problémát kezeli azáltal, hogy a szövegalkotás előtt a modell külső dokumentumokból keres releváns kontextust, és azt integrálja a válasz generálásába.

Jelen kutatás célja a retrieval-lépés hatékonyságának növelése. Nagyméretű dokumentumkorpuszok (pl. Wikipédia, tudományos cikkgyűjtemények vagy QA-adathalmazok) esetén a releváns információ visszakeresése rendkívül számításigényes feladat, hiszen minden lekérdezéshez több tízezer, sőt akár milliónyi szövegrész (chunk) vektoros reprezentációját kell összehasonlítani. Ez a folyamat a RAG-rendszerek egyik legszűkebb keresztmetszete, amely korlátozza azok valós idejű vagy online alkalmazását.

A dolgozat célja ezért a RAG-rendszerek hatékonyabbá tétele online klaszterezési módszerek integrálásával, különösen a centroid-alapú keresés alkalmazásán keresztül. A kutatás során bemutatásra kerül, hogyan csökkenthető a visszakeresési idő anélkül, hogy a pontosság jelentősen romlana, és milyen kompromisszum érhető el a gyorsaság és információvesztés között. A munka kísérleti platformja a SQuAD (Stanford Question Answering Dataset) adathalmaz, amely jól mérhető alapot biztosít a retrieval-modulok összehasonlításához.

## RAG-rendszerek szerepe a modern NLP-ben

A modern NLP-ben a RAG-rendszerek kulcsszerepet töltenek be az adatvezérelt tudás-hozzáférés, a forráshivatkozással alátámasztott szövegalkotás, valamint a megbízhatóbb generatív mesterséges intelligencia kialakításában. Széles körben alkalmazzák őket keresés-alapú chatbotokban, tudásmenedzsment rendszerekben, valamint olyan feladatokban, ahol a pontosság és az aktualitás kiemelt fontosságú. A RAG tehát nemcsak technológiai újítás, hanem paradigmaváltás is, hiszen az intelligens nyelvi rendszerek egyre inkább a „tudásra épülő”, nem pedig kizárólag a „nyelvre tanított” modellek irányába fejlődnek.

Míg korábban a modellek statikus tudásra támaszkodtak, a RAG megközelítés lehetővé teszi, hogy a rendszerek folyamatosan bővítsék és frissítsék tudásukat anélkül, hogy újratanítást igényelnének. Ez különösen fontos a gyorsan változó információs környezetekben, például a jogi, tudományos vagy üzleti szférában, ahol a relevancia és az aktualitás kulcstényezők. A modern NLP kutatásában a RAG így nemcsak egy hatékony technikai megoldás, hanem a mesterséges intelligencia átláthatóbbá és megbízhatóbbá tételének egyik legfontosabb irányvonala.

## Hatékonysági kérdések nagy dokumentumkorpuszok esetén

A RAG-rendszerek egyik legkritikusabb gyakorlati problémája a retrieval-lépés skálázhatósága. Nagy dokumentumkorpuszok esetén, ahol több millió szövegrész (chunk) kerül embedding formába, a keresés időigénye drámaian megnő. Egyetlen lekérdezés esetében akár több százezer koszinusz-távolság számítás is történhet, ami valós idejű alkalmazásokban (pl. kérdés–válasz rendszerekben) nem megengedhető.

A keresés gyorsítása több irányból közelíthető meg. Az egyik megoldás a vektorindexelés, például a FAISS, HNSW vagy ScaNN algoritmusok alkalmazásával, amelyek különböző approximációs módszerekkel (pl. kvantizáció, graf-alapú közelítés) csökkentik a szükséges összehasonlítások számát. Ezek az indexek azonban gyakran statikusak, a klasszikus FAISS indexek nem támogatják egyáltalán a dinamikus frissítést, emiatt, ha új dokumentumok kerülnek be, gyakran újra kell építeni az indexet, hogy megmaradjon az optimális teljesítmény. A HNSW egy dinamikus gráfstruktúra, amely hatékonyan támogatja új pontok hozzáadását, azonban a törlés és a tömeges frissítés továbbra is problémás. A ScaNN hasonlóan a FAISS-hoz szintén nem támogatja jól a folyamatos indexfrissítést.

Ezzel szemben a klaszterezésen alapuló megközelítések lehetőséget adnak a keresési tér intelligens felosztására. Az adathalmaz vektoraiból képzett klaszterek centroidjai egyfajta reprezentatív középpontot képeznek, amelyek alapján előszűrés végezhető, így a keresés először csak a centroidok között történik, majd a legközelebbi klaszter(ek)en belül folytatódik a részletesebb keresés. Ez a kétlépcsős folyamat jelentősen csökkenti az összehasonlítások számát, miközben a releváns találatok többsége továbbra is megtalálható marad.

A hatékonyság kulcsa a pontosság és sebesség közötti kompromisszum megtalálása. Ha túl kevés klasztert használunk, a keresés továbbra is lassú marad, ha viszont túl sokat, nő a hibás kizárások aránya. A jelen kutatás egyik fő célja ezért a paramétertér (klaszterszám, top-k klaszterek száma, top-n chunk visszaadása) grid search alapú optimalizálása, különböző embedding-méretek mellett.

További kérdést vet fel az online klaszterezés lehetősége. Míg az offline KMeans vagy MiniBatchKMeans csak teljes újratanítással tud reagálni az új adatokra, az online klaszterezési megoldások adaptív modelljei folyamatosan képesek frissíteni a centroidokat a beérkező minták alapján. Ez a megközelítés elméletileg lehetővé teszi a RAG-rendszerek folyamatosan tanuló retriever komponensének megvalósítását, olyat, amely idővel egyre jobban illeszkedik a beérkező kérdések és dokumentumok eloszlásához.

A dolgozat későbbi fejezeteiben bemutatásra kerül, hogy az ilyen online klaszterezési módszerek hogyan illeszthetők a RAG-rendszerek retrieval folyamatába, és milyen teljesítményt nyújtanak a hagyományos, minden lekérdezéskor teljes vektortérben végzett kereséssel szemben. A cél egy olyan robusztus, adaptív és gyors retrieval-architektúra megalkotása, amely a valós idejű RAG-rendszerek egyik kulcskomponensévé válhat.

# 2 Elméleti háttér

## 2.1 A RAG (Retrieval-Augmented Generation) elmélete

### 2.1.1 RAG architektúra felépítése

### 2.1.2 Dokumentum-visszakeresés alapjai

## 2.2 Embeddingek és vektorterek

### 2.2.1 Sentence-transformerek működése és hatásuk a reprezentációra

### 2.2.2 Távolságmértékek

Az embedding vektorok közötti hasonlóság vagy távolság mérésére több módszer is létezik. A koszinusz hasonlóság (cosine similarity) a két vektor közötti szöget méri, ami különösen jól működik a nagyméretű, normalizált embeddingeknél. Az euklideszi távolság (euclidean distance) a vektorok geometriai távolságát adja meg, míg a Manhattan távolság (Manhattan distance) az egyes koordináták abszolút különbségeinek összegét használja. A dot product lineáris kapcsolatot mér, amelyet gyakran neurális hálózatokban alkalmaznak.

A képen Betűtípus, szöveg, fehér, diagram látható

Automatikusan generált leírás

1. ábra Koszinusz hasonlóság számítási módja ‎[1]

A továbbiakban a munkám során az embedding vektorokat a cosine similarity segítségével hasonlítottam össze. A teljesítmény növelése érdekében minden vektort L2-normalizáltam, vagyis úgy skáláztam, hogy a hosszúságuk 1 legyen. A cosine similarity a vektorok közötti szöget méri, nem a nagyságukat; ha a vektorok különböző hosszúságúak, a nagyobb vektorok torzíthatják az eredményt. L2-normalizálás után a cosine similarity két embedding között egyszerűen a skaláris szorzatra (dot product) redukálódik, így a számítás is egyszerűsödik.

## 2.3 Klaszterezés az információkinyerésben

### 2.3.1 Klaszterezés célja

### 2.3.2 Offline és online klaszterezés

### 2.3.3 River könyvtár és saját online Kmeans fejlesztése

# 3 Adathalmaz és előfeldolgozás

## 3.1 A SQuAD adathalmaz

## 3.2 Szövegfeldolgozás és chunking

## 3.3 Embedding generálás

# 4 Klaszterezési módszerek implementációja és integrálása a retrieval folyamatba

## 4.1 Offline klaszterezés

## 4.2 Online klaszterezés

# 5 Eredmények és összehasonlítás

# 6 Összegzés és kitekintés

# Utolsó simítások

Miután elkészültünk a dokumentációval, ne felejtsük el a következő lépéseket:

* Kereszthivatkozások frissítése: miután kijelöltük a teljes szöveget (Ctrl+A), nyomjuk meg az F9 billentyűt, és a Word frissíti az összes kereszthivatkozást. Ilyenkor ellenőrizzük, hogy nem jelent-e meg valahol a "Hiba! A könyvjelző nem létezik." szöveg.
* Dokumentum tulajdonságok megadása: a dokumentumhoz tartozó meta adatok kitöltése (szerző, cím, kulcsszavak stb.). Erre való a Dokumentum tulajdonságai panel, mely a Fájl / Információ / Tulajdonságok / Dokumentumpanel megjelenítése úton érhető el.
* Kinézet ellenőrzése PDF-ben: a legjobb teszt a végén, ha PDF-et készítünk a dokumentumból, és azt leellenőrizzük.

Irodalomjegyzék

1. Varun, (2020.09.27.), Cosine similarity: How does it measure the similarity, Maths behind and usage in Python, towardsdatascience.com ,<https://towardsdatascience.com/cosine-similarity-how-does-it-measure-the-similarity-maths-behind-and-usage-in-python-50ad30aad7db/>

Függelék

Általános információk

A diplomaterv szerkezete:

1. Diplomaterv feladatkiírás
2. Címoldal
3. Tartalomjegyzék
4. A diplomatervező nyilatkozata az önálló munkáról és az elektronikus adatok kezeléséről
5. Tartalmi összefoglaló magyarul és angolul
6. Bevezetés: a feladat értelmezése, a tervezés célja, a feladat indokoltsága, a diplomaterv felépítésének rövid összefoglalása
7. A feladatkiírás pontosítása és részletes elemzése
8. Előzmények (irodalomkutatás, hasonló alkotások), az ezekből levonható következtetések
9. A tervezés részletes leírása, a döntési lehetőségek értékelése és a választott megoldások indoklása
10. A megtervezett műszaki alkotás értékelése, kritikai elemzése, továbbfejlesztési lehetőségek
11. Esetleges köszönetnyilvánítások
12. Részletesés pontos irodalomjegyzék
13. Függelék(ek)

Felhasználható a következő oldaltól kezdődő Diplomaterv sablon dokumentum tartalma. Ügyeljen a tanszék, a hallgató, a konzulens nevét és a beadás évét jelölő szövegdobozokra, mert azokra külön ki kell adni a frissítést. A mezők tartalma a sablonban a dokumentum adatlapja alapján automatikusan kerül kitöltésre (Fájl/Információ/Tulajdonságok/Speciális tulajdonságok).

A diplomaterv szabványos méretű A4-es lapokra kerüljön. Az oldalak tükörmargóval készüljenek (mindenhol 2.5cm, baloldalon 1cm-es kötéssel). Az alapértelmezett betűkészlet a 12 pontos Times New Roman, másfeles sorközzel.

Minden oldalon - az első négy szerkezeti elem kivételével - szerepelnie kell az oldalszámnak.

A fejezeteket decimális beosztással kell ellátni. Az ábrákat a megfelelő helyre be kell illeszteni, fejezetenként decimális számmal és kifejező címmel kell ellátni. A fejezeteket decimális aláosztással számozzuk, maximálisan 3 aláosztás mélységben (pl. 2.3.4.1.). Az ábrákat, táblázatokat és képleteket célszerű fejezetenként külön számozni (pl. 2.4. ábra, 4.2 táblázat vagy képletnél (3.2)). A fejezetcímeket igazítsuk balra, a normál szövegnél viszont használjunk sorkiegyenlítést. Az ábrákat, táblázatokat és a hozzájuk tartozó címet igazítsuk középre. A cím a jelölt rész alatt helyezkedjen el.

A képeket lehetőleg rajzoló programmal készítsék el, az egyenleteket egyenlet-szerkesztő segítségével írják le.

Az irodalomjegyzék szövegközi hivatkozása történhet a Harvard-rendszerben (a szerző és az évszám megadásával) vagy sorszámozva. A teljes lista névsor szerinti sorrendben a szöveg végén szerepeljen (sorszámozott irodalmi hivatkozások esetén hivatkozási sorrendben). A szakirodalmi források címeit azonban mindig az eredeti nyelven kell megadni, esetleg zárójelben a fordítással. A listában szereplő valamennyi publikációra hivatkozni kell a szövegben. Minden publikáció a szerzők után a következő adatok szerepelnek: folyóirat cikkeknél a pontos cím, a folyóirat címe, évfolyam, szám, oldalszám tól-ig. A folyóirat címeket csak akkor rövidítsük, ha azok nagyon közismertek vagy nagyon hosszúak. Internet hivatkozások megadásakor fontos, hogy az elérési út előtt megadjuk az oldal tulajdonosát és tartalmát (mivel a link egy idő után akár elérhetetlenné is válhat), valamint az elérés időpontját.

Fontos:

* a szakdolgozat készítő/diplomatervező nyilatkozata (a jelen sablonban szereplő szövegtartalommal) kötelező előírás Karunkon, ennek hiányában a szakdolgozat/diplomaterv nem bírálható és nem védhető!
* mind a dolgozat, mind a melléklet maximálisan 15 MB méretű lehet!

Jó munkát, sikeres szakdolgozat készítést ill. diplomatervezést kívánunk!