Универзитет ..Св. Кирил и Методиј“

Факултет за информатички науки и компјутерско инженерство

Предмет: *Управување со ИКТ проекти*

Нормализација и интеграција на податоци од продавници за електроника

(Анхоч и Нептун)

Теа Минова, 213004

Никола Јанковиќ, 213204

Ментор:

Милена Трајаноска

03.09.2025, Скопје

Содржина

Вовед

Ова истражување опфаќа собирање и обработка податоците за производите кои се продаваат во македонските продавници Нептун и Анхоч, со крајна цел да се изгради алгоритам кој ќе ги споредува производите од двете продавници и ќе детектира кои производи се исти, иако може да имаат различни имиња, формати на спецификации или дополнителни описи. Со оглед на тоа што секоја веб страница има сопствена структура и начин на прикажување на податоците, тука е потребно да се развие систематски пристап кој овозможува стандардизација, за понатамошна подетална споредба и соодветно спојување на совпаѓачките производи.

Краток опис на чекорите во истражувањето

Истражувањето беше изведено низ неколку последователни чекори:

Преземање на податоците

Податоците се извлечени директно од веб страниците на Нептун и Анхоч преку специјално развиени веб скрејпери. За оваа цел е користено Playwright, што овозможува работа со *headless* прелистувач за симулација на интеракции како отворање линкови, кликови и скролање. По вчитување на страниците, со помош на BeautifulSoup се селектираат релевантните HTML елементи и се составуваат CSV фајлови со добиените податоци.

Категоризација на производите

Со оглед на тоа што двете продавници имаат различни системи за категоризација, применет е LLM моделот LLaMA 3. За секој производ, користејќи ги негово име и спецификации, моделот генерира стандардизирана категорија. Temperature вредноста е поставена на 0 за да се добијат најконзистентни резултати. Дополнително, за секоја уникатна категорија е генерирана JSON структура со најчесто користените спецификации за тој тип производи.

Преработка на спецификации

Форматот на спецификациите значително варира меѓу веб страниците: структурирани во едниот случај, а повеќе описни и неструктурирани во другиот. За да се обезбеди унифициран формат, на моделот му се доставуваат текстуалните спецификации заедно со JSON структурата за соодветната категорија, со што моделот

ги пополнува вредностите за секој клуч. Овој процес овозможува добивање на конзистентни спецификации погодни за споредба.

Извлекување на клучни зборови

Бидејќи имињата на производите често содржат дополнителни информации (како технички спецификации или маркетинг елементи), на моделот му е зададена задача да го изолира главното име на производот.

Спојување на производи

Со помош на добиените стандардизирани податоци, е извршено ембедирање и пресметка на косинусна сличност за да се идентификуваат најсличните производи меѓу двете продавници.

За да се зголеми ефикасноста и да се намали времето на извршување, сите чекори се оптимизирани со користење на конкурентност. Применети се повеќе нишки и семафори за безбеден паралелен пристап до податоците. При скрејпањето се користи async_playwright, додека за повиците до LLM моделот (преку Groq API) е имплементирана логика за повторно обидување со exponential backoff.

Чекор 1 – Преземање на податоците

Првиот чекор од истражувањето беше да се соберат сите производи од двете веб страници - Нептун и Анхоч. Со оглед на тоа што секоја страница има различна структура на HTML елементите и навигацијата, за секоја од нив беа развиени посебни скрејпери.

Скрејпер за Нептун

За Нептун се користи асинхронски скрејпер со asyncio и playwright.async_api. Главните компоненти се:

- **get_category_urls(url)**
  Оваа функција ја презема главната страница, со requests и BeautifulSoup, и ги извлекува сите линкови до категориите.
- **get_inner_categories(context, url)**
  Некои категории имаат „внатрешни“ подкатегории. Оваа функција со Playwright отвара нова страница, симулира вчитување, и ги извлекува линковите до тие подкатегории.
- **get_specs(context, url)**
  За секој производ, од посебната product страница се селектира делот со спецификациите (panel-body), од каде се извлекуваат сите <li> елементи и се зачувуваат како текст.
- **scrape_products(context, url)**
  Ова е клучна функција, која:
  - Влегува во категоријата, го поставува бројот на производи по страница на 100 (ако е достапно).
  - Со BeautifulSoup ги извлекува сите производи: име, цена, „Happy“ цена (ако има попуст), слика и линк.
  - За секој производ дополнително се повикува get_specs() за да се соберат спецификациите.
  - Се справува со pagination (копчињата „Next“) за да се соберат сите производи од сите страници од категоријата.
- **scrape_page(context, category_url, base_url)**
  Со помош на семафори (за контрола на конкурентност), оваа функција ја повикува scrape_products() за секоја категорија и подкатегорија.

- **main()**
    - Ги зема сите категории.
    - Отвара headless Chromium прелистувач.
    - Паралелно (со asyncio.gather) скрејпува од сите категории.
    - Резултатите ги спојува и ги зачувува во neptun_products.csv.

Скрејпер за Анхоч

За Анхоч се користи синхронски пристап со playwright.sync_api, за поедноставно скрејпање на основните информации за производите, а потоа се користи дополнителен скрејпер за зачувување на спецификациите.

- **get_category_urls(url)**
  Од главната страница https://www.anhoch.com/categories, со BeautifulSoup се извлекуваат сите категории, при што се филтрираат непотребните (на пример, „vouchers“).
- **extract_products(page_html)**
  Од HTML на страниците за секоја категорија, функцијата ги наоѓа сите „product-card“ елементи и извлекува: име, цена, слика и линк.
- **get_total_pages(soup)**
  Бидејќи секоја категорија може да има повеќе страници со производи, од pagination елементот се извлекува бројот на вкупни страници.
- **scrape_products_from_category(category_url, page)**
  За дадена категорија:
    - Се добива бројот на страници од производи.
    - Се вчитува секоја страница (?page=N).
    - Од секоја страница се извлекуваат производите со extract_products().
- **get_products(category_urls)**
  Отвара Playwright browser и ги скрејпува сите категории по ред.
- **main()**
    - Ги зема категориите.
    - Ги скрејпува сите производи.
    - Резултатите ги зачувува во anhoch_products.csv.

Дополнителен скрејпер за спецификации (Анхоч)

Првиот Анхоч скрејпер ги зема основните информации (наслов, цена, слика, линк). За спецификациите е развиен посебен процес:

- **scrape_specifications(batch_df)**
    - Ги зема производите со празни спецификации.
    - Со Playwright го отвора секој линк и ја извлекува содржината од div#description.
    - Резултатот се зачувува во колоната „Specifications“.
- **main()**
    - Го вчитува CSV-то со основните податоци (anhoch_products_basic_info.csv).
    - Проверува кои производи немаат спецификации.
    - Ги обработува во батчови по 200 производи (за да не се блокира страната).
    - По секој батч, резултатите се запишуваат во anhoch_products_with_specs.csv.

Овој чекор овозможува сите производи од Анхоч да имаат потполни спецификации, што е критично за споредба со производите на Нептун.

Чекор 2 – Категоризација на производите

Откако се собрани основните податоци (наслов, цена, спецификации, слика, линк), следниот чекор е да се категоризира секој производ. Ова е потребно затоа што Нептун и Анхоч користат различни системи за категоризација, па со цел да може да се споредуваат производите е неопходно да има стандардизирани категории.

За оваа задача е искористен LLM моделот LLaMA 3 преку Groq API, кој овозможува обработка на природен јазик и генерирање на точни категории.

1. Подготовка и иницијализација

- Се вчитува CSV датотеката neptun_products_categorized.csv, која ги содржи производите.
- Ако колоната Category е празна или има грешка, производот се испраќа на повторна категоризација.
- Се иницијализира Groq клиентот со API клучот.

2. Промпт за моделот

Секој производ се претставува со неговото име и спецификации. На моделот му се дава задача:

You are a precise product categorization assistant for tech products.

Your task is to identify the most appropriate category for the given product based on its title and specifications.

Provide only the single, most specific category name in your response (e.g., "Processor", "Motherboard", "Graphics Card", "RAM").

Овој промпт гарантира дека моделот враќа една, специфична категорија, без дополнителен текст. Temperature е поставено на 0.0, за да се минимизира случајноста и да се добиваат доследни резултати.

3. Retry механизам со exponential backoff

Поради тоа што API повиците можат да вратат грешки (на пример 429 Too Many Requests, 500 Server Error, timeout), имплементиран е систем за:

- Проверка дали грешката е retryable (HTTP 429–504, timeout, connection error).
- Повторување на барањето до 10 пати.
- Exponential backoff: по секој обид, времето на чекање се зголемува (2s, 4s, 8s, … до 60s)
- Се додава „jitter“ (случајна мала варијација), за да се избегне да се испраќаат барања синхронизирано.

Ова гарантира стабилност и минимизира загуба на податоци.

4. Конкурентна обработка

- Производите се делат во батчови од 20 (CONCURRENT_BATCH_SIZE = 20).
- Секој батч паралелно праќа барања кон API-то со asyncio.gather.
- По обработката на секој батч, резултатите веднаш се зачувуваат во CSV.
- Се прави кратка пауза од 2 секунди (DELAY_BETWEEN_BATCHES_S) за да се намали ризикот од rate limiting.

5. Запишување на резултатите

- За секој производ се запишува новата категорија во колоната Category.
- Прогресот се зачувува по секој батч → ако програмата се прекине, подоцна може да продолжи од таму.

Пример

Ако производот содржи:

Title: *„Intel Core i7-12700K Processor 12-Core 3.6GHz“*
Specs: *„12 cores, 20 threads, LGA 1700, Base clock 3.6 GHz, Turbo up to 5.0 GHz“*

Моделот враќа:

Processor

Овој чекор овозможува:

1. **Стандардизација** – сите производи добиваат унифицирани категории, независно од тоа како се прикажани на веб страниците.
2. **Подготвителен чекор за JSON структури** – уникатните категории понатаму се користат за генерирање на стандардизирани JSON шаблони за спецификациите.
3. **Автоматизација** – процесот е целосно автоматизиран со LLM, без потреба од рачна категоризација.
4. **Отпорност** – благодарение на retry логиката и батч процесирањето, скриптата е стабилна дури и при голем број производи.

Чекор 3 – Преработка на спецификации

Овој чекор е составен од три потчекори: Нормализација на категориите, генерирање на JSON-шеми по категорија и екстракција и пополнување според генерираните шеми.

1. Нормализација на категории (preprocess)

Влезен фајл: neptun_products_categorized.csv

Излезен фајл: products_preprocessed.csv

Пред да се генерираат шеми и да се извлекуваат спецификациите, потребни се конзистентни категории, за секоја категорија да има една JSON-шема. За таа цел, се стандардизираат имињата на категориите (пр. Television - TV, CPU - Processor, Flash Drive - USB Flash Drive), за да нема дупликати поради синоними.

2. Генерирање на JSON-шеми по категорија

Влезен фајл: products_preprocessed.csv

Излезен директориум: schemas/schema_<Category>.json

За секоја уникатна категорија се генерира едноставна, рамна JSON-шема (клучеви во snake_case, вредности празни стрингови).

- Се читаат уникатните категории од products_preprocessed.csv.
- За секоја категорија се праќа промпт до Groq / LLaMA 3 (модел llama3-8b-8192) да врати само „raw JSON“.
- Се зачувува како schemas/schema_<Category>.json.

Правила во промптот:

- Само JSON-објект, без објаснувања.
- Клучеви во snake_case.
- Вредности - празни стрингови (placeholders).

Ова создава стандард што потоа ќе се пополнува за секој конкретен производ, при што се добива ист формат на спецификации за различните продавници.

3. Екстракција и пополнување на шемите (LLM extraction)

Влезни фајлови: products_preprocessed.csv, ./schemas/*.json

Излезен фајл: products_extracted.csv

За секој производ (Title + неструктуриран Specs) се зема соодветната JSON-шема за неговата категорија, па со LLM се пополнуваат клучевите со вредности извлечени од описот.

- Ги вчитува сите шеми од ./schemas.
- Ако постои products_extracted.csv, продолжува (resume) од таму; инаку стартува ново со две нови колони:
    - extraction_status (статус за секој ред)
    - extracted_specs (JSON како string)
- За редовите што сè уште не се „complete“, креира batch од CONCURRENT_BATCH_SIZE=15 и паралелно праќа повици до LLM.
- Промпт логика:
    - Пополнување на JSON-шемата според Title + Specifications.
    - Превод на вредностите од македонски на англиски.
    - За непостоечки вредности се поставува null.
    - Се користи response_format={"type":"json_object"} за да се форсира чист JSON-одговор.
- Отпорност**:**
    - Retry со exponential backoff + jitter (2s → 4s → 8s … до 60s).
    - „Retryable“ грешки: 429/5xx/timeout/connection error.
    - По секој batch: cooldown 2s (намалува rate-limit ризик).
- Обележување на случаи**:**
    - skipped_no_specs ако редот нема спецификации.
    - skipped_no_schema ако нема валидна/пронајдена шема за таа категорија.
    - error или error_retries_failed за дефекти при екстракција.
    - complete кога JSON е успешно добиен и снимен.
- Запис**:**
    - extracted_specs ја чува пополнетата JSON-структура (како string) — лесно за парсирање понатаму.
    - Секој batch се зачувува веднаш (отпорен на прекин).

Пример

Category: *Graphics Card*

Schema (дел):

{
"brand": "",
"model": "",
"chipset": "",
"vram_size": "",
"memory_type": "",
"boost_clock": "",
"power_connector": ""
}

Specs (неструктурирани): *„ASUS Dual RTX 4060 8GB GDDR6, Boost 2460 MHz, 8-pin…“*

LLM излез:

{
"brand": "ASUS",
"model": "Dual RTX 4060",
"chipset": "NVIDIA GeForce RTX 4060",
"vram_size": "8 GB",
"memory_type": "GDDR6",
"boost_clock": "2460 MHz",
"power_connector": "8-pin"
}

Со овој пристап се овозможува:

- **Еднаква структура** за сите продукти - директно споредување/спојување.
- **Отпорен на варијации** во описите (LLM ги „фаќа“ важните вредности и ги преведува).
- **Скалабилен и стабилен** (batch + concurrency + backoff + авто-resume).

Чекор 4 – Извлекување на клучни зборови

Во насловите на производите обично има многу дополнителни информации — големини, бои, спецификации, па дури и маркетинг термини. Во овој чекор, главната цел е од насловите на производите да се издвои само суштинската вредност, односно името на моделот на производот (пр. „Core i9-14900K“, „GeForce RTX 4070 GAMING OC“, „VG-SCFT55WT“), за полесна и попрецизна споредба на производите.

Влезен фајл: products_extracted.csv

Излезен фајл: products_final.csv (истите податоци + нова колона Model Name).

1. **LLM промпт**
    - Се праќа промпт до LLAMA 3 (преку Groq API) кој прецизно бара:
        - да се врати само името на моделот,
        - без воведи („The model name is:“),
        - без наводници или објаснувања.
    - Дадени се и примери за насочување на моделот.
2. **Асинхрона обработка**
    - Се користи asyncio со CONCURRENT_BATCH_SIZE = 20, за повеќе барања во паралела.
    - По секој batch има пауза од 2s за да се намали ризикот од rate-limits.
3. **Retry logic**
    - До MAX_RETRIES = 10 обиди со exponential backoff + jitter (2s → 4s → 8s … до 60s).
    - „Retryable“ грешки: 429 (Too Many Requests), 5xx, timeout, connection errors.
    - Ако не успее - Extraction Error.
4. **Resuming**
    - Ако постои products_final.csv, скриптата продолжува од таму.

- Секој batch се снима веднаш во CSV, така што нема губење на напредок.

Колони во финалниот датасет

- Title – оригинален наслов на производот.
- Specs – оригинални спецификации.
- extracted_specs – JSON со стандарден формат од претходниот чекор.
- Model Name – извлечено јадро на моделот (клучен збор).

Пример

Влез:

Title: "CPU Intel Core i9-14900K 3.2GHz FC-LGA16A"

LLM излез:

Core i9-14900K

Чекор 5 – Спојување на производи

Во овој чекор се користат сите претходни резултати (чистени наслови, извлечени спецификации, клучни зборови) за интелигентно да се најдат исти или еквивалентни производи од двете продавници.

1. Предобработка на податоци

- **Чистење на текст**: функцијата clean_text() ги отстранува интерпункциските знаци, повеќекратните празни места и ја претвора содржината во lower-case.
- **Сплескување на JSON спецификациите**: функциите flatten_json_for_text() и json_to_representative_string() го претвораат структурираниот JSON од спецификациите во една репрезентативна текстуална низа.
- Во preprocess_dataframe() за секој производ се добиваат:
    - Title_clean – исчистен наслов,
    - Specs_clean – текстуално претставени спецификации.

2. Генерирање векторски репрезентации (embeddings)

Се користи SentenceTransformer моделот all-MiniLM-L6-v2 кој дава векторски репрезентации на текст:

- Се прават embeddings за наслови од двете продавници.
- Се прават embeddings и за спецификациите.

3. Пресметка на сличност

- Се пресметува cosine similarity матрица за сите наслови, со што се покажува колку се слични насловите на производите од Анхоч и Нептун.
- За секој производ од Анхоч:
    - Се земаат top_k најслични кандидати од Нептун според насловите.
    - Се филтрираат со title_threshold ≥ 0.5.
    - Потоа се проверуваат и спецификациите, при што само ако имаат spec_threshold ≥ 0.8, производите се сметаат за доволно слични.

4. Дополнителен филтер: model tokens

Функцијата have_no_shared_model_tokens() проверува дали постојат заеднички токени со броеви и букви (пр. „i9-14900K“, „RTX4070“) во спецификациите. Ако нема заеднички модел токени, производот најчесто не е ист, па следствено кандидатот се исклучува.

5. Финално спојување

За секој прифатен пар производи:

- Се чуваат насловите, цените, линковите, спецификациите (JSON), сликите.
- Се чуваат и метрики за сличност (title similarity, specs similarity).
- Сето тоа се снима во matched_products.csv.

Излезниот датасет ги содржи следните колони:

- Title_Store1, Title_Store2 – наслови од двете продавници,
- Price_Store1, Price_Store2, HappyPrice_Store1, HappyPrice_Store2 – цени,
- Link_Store1, Link_Store2 – линкови кон производите,
- Specs_JSON_Store1, Specs_JSON_Store2 – оригинални спецификации,
- Image_Store1, Image_Store2 – слики,
- Title_Similarity_Score, Specs_Similarity_Score – оценка за сличност.

Пример

Влез:

Анхоч производ: „CPU Intel Core i9-14900K 3.2GHz FC-LGA16A“

Нептун производ: „Intel Core i9-14900K Processor 3.2GHz LGA1700“

Излез:

Title_Similarity_Score = 0.86

Specs_Similarity_Score = 0.91

=> Производите се мапираат како идентични.

На овој начин се овозможува:

- Автоматизирано да се откриваат дупликати или исти производи од различни продавници, иако се опишани на различен начин.
- Се добива основа за агрегација на цени, споредба на понуди, динамично прикажување на најевтини/најскапи продавници.
- Финалниот резултат е чиста и унифицирана база на податоци со вкрстени производи.

Заклучок

Како што е наведено и на почетокот, целта на ова истражување беше да се изгради систем кој автоматски ќе ги обработува, категоризира и споредува производите од продавниците за електроника во Македонија, со цел нивно обединување во унифицирана база на податоци погодна за анализа, споредба и понатамошна интеграција во апликации за споредба на цени и спецификации.

Главни придобивки

- Создадена е стандардирана база на производи со унифицирани категории и клучни спецификации.
- Овозможено е автоматско откривање на ист производ од различни извори, без потреба од рачно споредување.
- Системот е робустен благодарение на механизми за повторни обиди, толеранција на грешки и динамично приспособливи прагови за сличност.
- Резултатот е практична алатка која може да се користи за агрегатори на цени, паметни препораки или како основа за деловна аналитика во трговија со електроника.

Идеи за понатамошно подобрување

- Воведување на поголеми јазични модели (LLMs) за подобра семантичка анализа на спецификации.
- Изградба на централизирана база која континуирано ќе се надополнува со нови податоци од повеќе продавници.
- Додавање на механизам за следење на промени во цени и историска анализа.

Истражувањето покажа дека е можно со комбинација од техники за обработка на текст (NLP), машинско учење и јазични модели да се изгради систем кој сигурно и ефикасно ги спојува производите од различни извори. Со тоа се поставуваат темелите за создавање на платформи за споредба на цени во Македонија, кои ќе им овозможат на потрошувачите транспарентност, а на продавниците фер конкуренција и подобро позиционирање на пазарот.