DOI: **10.37943/25VBID7988**

Madina Mansurova
Professor, Head of the Department of Artificial Intelligence and Big Data
madina.mansurova@kaznu.edu.kz, orcid.org/0000-0002-9680-2758
Al-Farabi Kazakh National University, Kazakhstan
Assel Ospan
Master degree, Senior Lecturer, Department of Artificial Intelligence and Big Data
assel.ospan@kaznu.edu.kz, orcid.org/0000-0002-1860-6997
Al-Farabi Kazakh National University, Kazakhstan
Fakhriddin Nuraliev
Doctor of Technical Science, Professor
nuraliev2001@mail.ru, orcid.org/0000-0002-0574-9278
Tashkent University of Information Technologies named after Muhammad al-Khwarizmi, Uzbekistan
Rustam Khamdamov
Doctor of Technical Sciences, Professor, Head of the Lab. "Smart Systems. Internet of Things"
khamdamovrustam007@gmail.com, orcid.org/0000-0003-3796-4631
Tashkent University of Information Technologies named after Muhammad al-Khwarizmi, Uzbekistan
Talshyn Sarsembayeva
Master degree, Senior Lecturer, Department of Artificial Intelligence and Big Data
talshyn.sagdatbek@kaznu.edu.kz, orcid.org/0000-0001-7668-2640
Al-Farabi Kazakh National University, Kazakhstan

BUILDING A DEEP SEARCH CRAWLER FOR THE KAZAKH LANGUAGE: A REPRODUCIBLE WEB-SCALE PIPELINE

Abstract: We present a reproducible, web-scale pipeline for building a Kazakh-language corpus from the national e-government portal. The system treats the website as a directed graph and performs breadth-first traversal to preserve section hierarchy. Static acquisition relies on robust HTTP requests and HTML parsing; for pages with dynamic widgets, we selectively enable a headless layer to render the final DOM prior to extraction. We define a minimal JSON schema aligned with downstream NLP needs (URL, category, titles, cleaned descriptions) and implement normalization (Unicode NFC/NFKC, transliteration repair for Kazakh, boilerplate removal) and fragment-level deduplication. To strengthen the scientific contribution, we formalize the crawling–extraction process as an optimization under resource constraints and propose field-level quality metrics (precision, recall, F1), coverage of categories, and completeness gains attributable to headless rendering. Our experimental protocol compares static parsing against a hybrid static+headless setup on multiple portal categories, reports field-wise effectiveness with confidence intervals, and analyzes dominant error sources (DOM drift, client-side rendering, code-switching). Ablation studies quantify the impact of normalization and duplication. We also outline ethical access (robots.txt compliance, throttling, conditional requests) and provide artifacts to ensure reproducibility (versioned scripts, schema validators, logging). We release open-source scripts, detailed runbooks, and a small, labeled benchmark to facilitate fair comparisons and independent replication across institutions. The resulting corpus targets low-resource Kazakh NLP and e-government analytics, supporting tasks such as classification, terminology normalization, named-entity recognition, and LLM adaptation. Overall, the proposed pipeline demonstrates that selective headless rendering combined with rigorous normalization is a practical and effective strategy for high-quality data acquisition in dynamically rendered public portals.

Keywords: web scraping; deep search crawler; Kazakh; e-government; breadth-first search (BFS); Django; BeautifulSoup; Playwright; Selenium; JSON schema.

Introduction

Received: 05.09.2025 *Accepted: 25.02.2026* *Published: 30.03.2026*

The rapid growth of web content, together with the Web as Corpus paradigm, has expanded the possibilities for collecting and studying linguistic data for NLP tasks and training large language models: the internet is viewed not as a static archive but as a continuously evolving textual repository that makes it possible to capture language shift, registers, and domain-specific features, including the official bureaucratic style of government resources [1]. Against this backdrop, web scraping and crawling serve as key infrastructural technologies: survey studies systematize methodologies and criteria for choosing approaches in light of source constraints and research goals [2].

Web scraping techniques are conventionally divided into static and dynamic modes. In the static mode, data are extracted directly from HTML, whereas the dynamic mode relies on a controlled browser engine that emulates user behavior [3]. Tool comparisons show that the combination of BeautifulSoup and Scrapy is effective for robust structure extraction and for building reproducible collection pipelines [4-5], while a browser automation layer (Selenium/Playwright) is required for pages with JavaScript rendering and hidden interface elements [6].

Integrating scraping with natural language processing enables the transformation of unstructured web text into research-ready representations. In this work, NLP methods are not directly applied but are considered as downstream applications of the collected corpus. Scraping+NLP practices cover the extraction of semi-structured blocks, normalization, and subsequent linguistic analytics, as documented by modern surveys on their joint use [7]. Core NLP components such as Named Entity Recognition (NER) and Part-of-Speech (POS) tagging can be applied to the resulting dataset to support automatic identification of entities and grammatical roles in large document collections [8]; comparative studies report that NER accuracy varies by entity class and corpus type [9].

After collection and primary linguistic processing, an analytical layer can be applied to the resulting corpus. For example, sentiment analysis of social media messages can help reconstruct the dynamics of public opinion [10], while automatic summarization methods can compactly condense large volumes of text without losing key information [11]. In practical applications, the scraped and normalized corpus may be used for business analytics, topic modeling, and information extraction [12-13]. These applications are considered as potential downstream use cases of the proposed pipeline rather than components of the current study.

In the Kazakhstani context, systematic extraction from the e-government services portal egov.kz is particularly important. The “Online services” section spans 16 categories of agency scenarios; pages are structurally similar (repeated containers, tables, lists) yet subject to dynamic rendering and periodic frontend changes. However, for the purpose of this study, 4 categories were excluded from the experimental corpus due to incomplete structure, limited availability of service descriptions, or inconsistent page layouts that hinder reliable automated extraction. Therefore, the analysis focuses on 12 categories that provide stable, representative, and sufficiently structured data for evaluation. The production pipeline presented here crawls categories sequentially, extracts and normalizes domain-relevant fields, and then stores records in JSON. The minimal schema comprises url, category, Title (all <h1> headings), and descriptions (cleaned content fragments). Extraction is hybrid: DOM selectors, regular expressions, and anchor dictionaries are complemented by block-level heuristics (h1–h4 headings, repeated cards, stable th/td pairs); normalization follows Unicode standards (NFC/NFKC), removes transliteration variants (“qazaqstan” → “Қазақстан”, “bilim” → “білім”), harmonizes phone formats, and filters boilerplate (popup, modal, banner). Additionally, fragment-level deduplication is applied, e.g., using the Jaccard coefficient [14-15]. The linguistic specifics of Kazakh – code-switching (ru/kk), orthographic variation, and agglutinative morphology – require specialized normalization procedures and subsequent morphological disambiguation; recent work demonstrates the effectiveness of LLM-based approaches for this task [16]. To improve completeness on dynamic pages, headless rendering is used to load the “final” DOM before parsing [17].

Finally, integrating scraping and NLP entails technical and regulatory risks. Practical challenges include handling dynamic pages, controlling data quality and completeness, and accommodating server-side constraints and access policies [18]. Ethical aspects – privacy, copyright, and fair practices for data collection and reuse – are treated as integral to research protocols [19-20]; promising directions include tighter integration with AI-based information retrieval and the development of reproducible quality assessment procedures [21-22].

As a method of automated data extraction from websites, web scraping has become widespread amid the explosive growth of online content. A spectrum of approaches has formed – from simple rule-based parsing to more advanced methods using specialized libraries such as BeautifulSoup and Scrapy. Several studies show that these tools can effectively collect texts from social media; for example, Zeng et al. demonstrate how to extract tweets for downstream sentiment analysis and topic modeling [23]. Similar strategies are applied in e-commerce: Wong and Lee use web scraping to gather large-scale product reviews, enabling marketing research and the assessment of consumer sentiment [24-25].

Natural Language Processing (NLP) – the discipline studying how to teach computers to read, interpret, and generate human language – has become a key area of modern computer science. Basic text preparation includes tokenization, stemming, and stop-word removal. A major qualitative leap came from distributed word representations: the Word2Vec model markedly improved results in text classification and sentiment analysis [26]. Further progress is associated with transformer architectures, particularly BERT, which provide context-sensitive representations and significantly boost performance across a wide range of NLP tasks [27].

The combination of web-scraping technologies (for large-scale text acquisition) and NLP methods (for analyzing unstructured data) provides a robust framework for extracting meaningful insights from online content. A representative example is given by Ghosh and Veeraraghavan, where scraping aggregates texts from multiple social-media platforms and the resulting corpus is subsequently processed with sentiment analysis, classification, and topic modeling [28].

The scientific and practical contributions of this work are twofold. First, we describe a reproducible data-collection pipeline for the egov.kz portal that covers 16 “Online Services” categories, explicitly accounting for legal and technical access constraints; we experimentally show that adding headless rendering noticeably increases the completeness of recognized fields compared to static parsing. Second, we propose an automatic tag-annotation scheme for service cards, oriented toward the official register and suitable for subsequent LLM-based applications (e.g., classification, entity extraction, terminology normalization). We also present a JSON schema and a minimal set of validation metrics (precision/recall by field, category coverage, share of dynamic pages), together with a protocol for manual spot-checking. Finally, we release a reproducible set of scripts and examples (JSON demo fragments) that can serve as a starting point for building Kazakh-language domain corpora in the official-administrative style.

Our research questions are as follows.

Q1: How comprehensively does the pipeline cover the 12 categories and 915 egov.kz service descriptions in terms of the target fields extracted?

Q2: What precision and recall does automatic annotation achieve on the gold sample, and how are these metrics affected by headless rendering?

Q3: What are the main error sources (dynamic content, DOM drift, code-switching, orthographic variability), and which heuristics/normalizations yield the largest quality gains?

Q4: To what extent is the proposed pipeline transferable to structurally similar government portals and adjacent domains?

Thus, the work contributes a high-quality, structured, and legally compliant subcorpus for Kazakh-language NLP and LLM adaptation. Unlike broad web corpora, where the official register is often fragmentary, our dataset deliberately focuses on normative-administrative lexicon and typical citizen–government interaction scenarios, addressing a critical data gap [10-12]. In the longer term, the linked corpus is viewed as part of a broader data ecosystem (including topic

modeling and semantic vector representations) and as a reference resource for downstream e-gov analytics and dialogue systems [13-14].

The article is structured as follows: Section 2 discusses the data source and access policy; Section 3 – materials and methods (crawler architecture, auto-tagging, JSON schema, metrics); Section 4 – experimental setup; Section 5 – results and error analysis; Section 6 – discussion and transferability; Section 7 – ethics and legal aspects; followed by limitations, conclusions, code/data availability, and appendices.

Data source and access policy

Our primary data source is the national e-government portal egov.kz (the "Online Services" section). We collect only public, unauthenticated pages via public URLs. A user supplies a starting section/subsection URL; the system then traverses nested pages strictly within the portal's public navigation and extracts textual elements from individual service cards (see Figure 1).

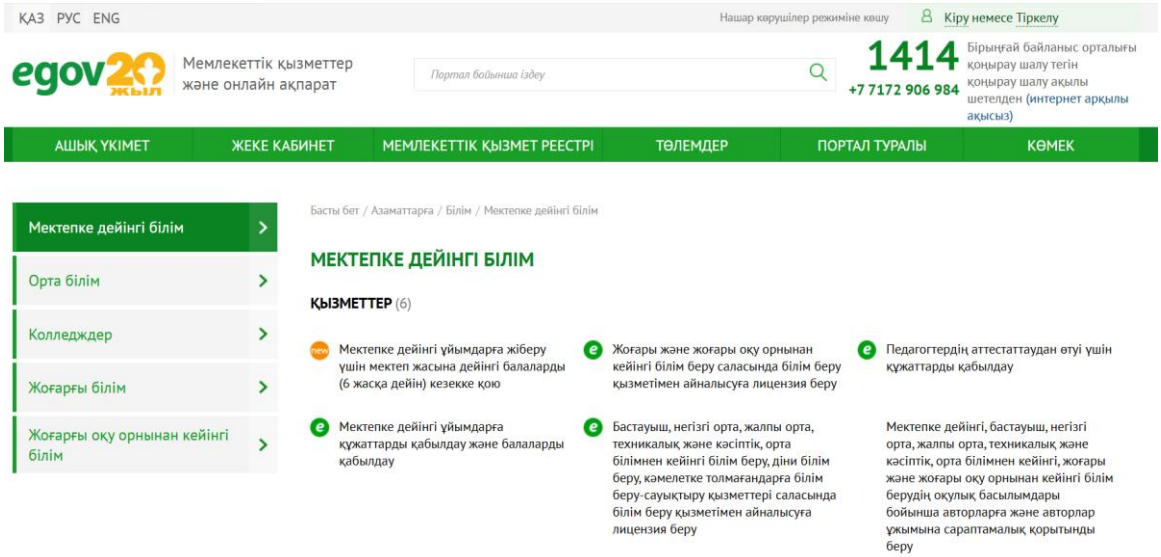

Figure 1. The structure of the egov.kz portal

The implementation uses a Django web view (scrape_view) that accepts a POST form with a single url field. After validating the address (checking the http prefix), the crawler issues idempotent HTTP GET requests, normalizes responses to UTF-8 to handle Kazakh text correctly, and parses HTML with BeautifulSoup (html.parser). Parsed records are serialized into the session's temporary store and exposed for on-screen review and JSON export via download_json. The UI (scrape.html, scrape_result.html) provides an input form, a discovered-links counter, and a JSON export button.

Traversal follows the site's actual DOM structure: the entry point is <ul class="egov-submenu"> (subcategory links), subpages expose lists of services under , and each service card is fetched on its own page. From the card page, the system extracts all <h1> headings, content fragments from , and the category inferred from breadcrumbs in (the third element when present). The minimal record schema contains url (canonical page address), category (from breadcrumbs), Title (all <h1> texts), and descriptions (cleaned, concatenated main-content fragments) (see Fig.2). These records are returned as a JSON array and simultaneously rendered in the results template.

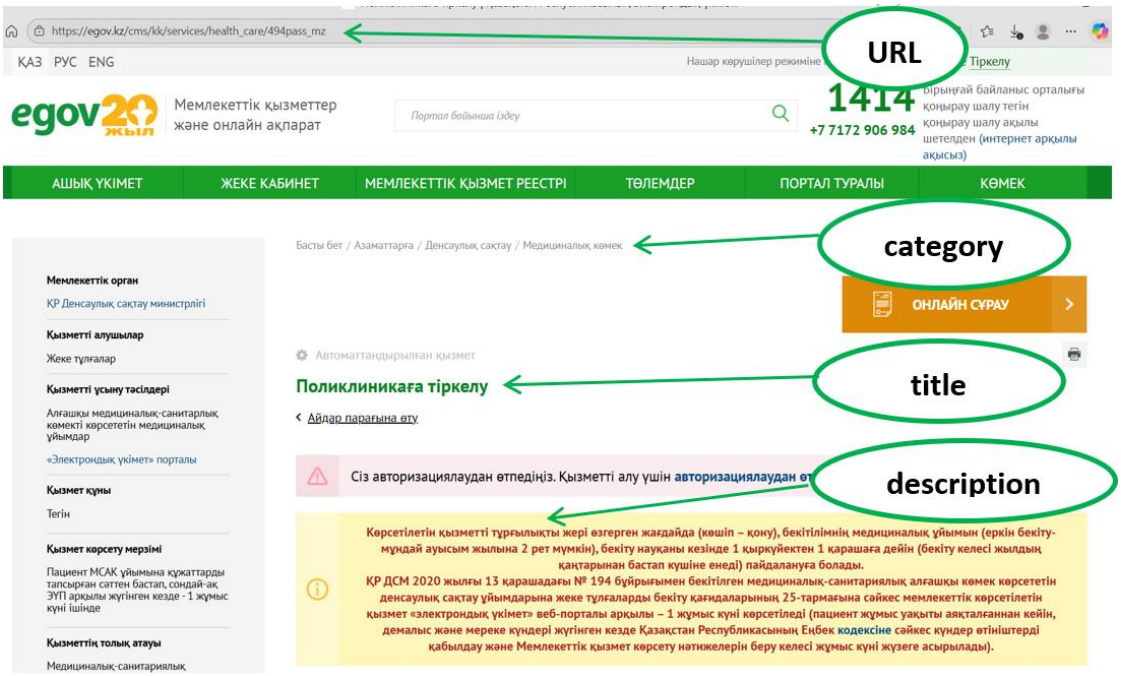

Figure 2. Basic markup fields on egov.kz service pages used during parsing

As a research corpus, we target 12 top-level Online-Services categories. In our current crawl, the largest slice is Education (115 services), followed by Social security (104) and Real estate (114); the smallest is Consular services (12), with other categories (e.g., Family, Healthcare, Transport & Communication, Legal Assistance, Customs & Taxation, Employment) in the mid-range. The distribution is shown in Table 1. This coverage yields a domain-specific subcorpus of official/administrative language suitable for downstream Kazakh NLP and e-government analytics.

Table 1. Name of categories of online services of the egov.kz portal.

№	Name of categories	Number of services
1.	Education	115
2.	Military registration and service	52
3.	Social security	104
4.	Family (birth, marriage, divorce)	88
5.	Healthcare	60
6.	Transport and communication	72
7.	Legal assistance	66
8.	Customs and taxation	77
9.	Real estate	114
10.	Job search and employment	68
11.	Consular services	12
12.	Civil services	85

Access and ethics policies enforce public-content-only and read-only operation, explicit User-Agent identification, graceful skipping on network errors (no retries), and forced UTF-8 decoding. Current limitations include no automatic robots.txt checks, no explicit rate-limiting, and no headless JavaScript rendering; for production, we plan to add robots.txt compliance, throttling with jitter and bounded parallelism, limited retries and conditional GETs, a headless layer (Selenium/Playwright), Unicode normalization (NFC/NFKC), boilerplate filtering, a formal JSON Schema, error logging, and coverage counters.

3 Methods and Materials

The developed pipeline is focused on reproducible collection and language processing of service descriptions from the national portal egov.kz (section "Online services") with subsequent automatic tagging and export to JSON. Below are the crawler architecture, principles of auto-tag tagging, target JSON schema, quality assessment methodology, as well as the algorithmic crawl plan, data flow representation (DFD) and the mathematical models used.

3.1. Mathematical Model

Let the public portal be a directed graph $G = (V, E)$with pages $v \in V$and hyperlinks $(u, v) \in E$restricted to the target domain. A breadth-first traversal maintains a frontier Q_t, depth $d(v)$, and a visited set $\mathcal{V}$. For each page $v$the extractor targets a field set $F =$ {url,category,title,description}. Let $y_{v,f} \in \{0,1\}$indicate whether field $f$was correctly extracted on v(per gold labels $y^{\backslash *}_{v,f}$). Precision/recall and F1 per field are shown in formula 1, 2, 3.

$$\mathrm{Prec}_f = \frac{\sum_v 1[y_{v,f}=1 \wedge \hat{y}_{v,f}=1]}{\sum_v 1[\hat{y}_{v,f}=1]} \tag{1}$$

$$\mathrm{Rec}_f = \frac{\sum_v 1[y_{v,f}=1 \wedge \hat{y}_{v,f}=1]}{\sum_v 1[y_{v,f}=1]} \tag{2}$$

$$\mathrm{F1}_f = 2\,\mathrm{Prec}_f \mathrm{Rec}_f / (\mathrm{Prec}_f + \mathrm{Rec}_f) \tag{3}$$

Headless rendering is a binary decision $h_v \in \{0,1\}$ with costs c_s(static) and $c_h > c_s$(headless). Under a time budget B, choose $h = \{h_v\}$ to maximize a weighted field-level F1 (formula 4):

$$\max_{h_v \in \{0,1\}} \sum_{f \in F} w_f\, \mathrm{F1}_f(h) \text{ s.t. } \sum_{v \in V} (c_s + (c_h - c_s)\, h_v) \le B \tag{4}$$

We model field completeness as $p_{v,f}(h_v) = \sigma(\alpha_f + \beta_f h_v + \gamma_f^{\top} z_v)$, where $z_v$summarizes DOM complexity (e.g., script density, presence of tabs). This yields an expected-utility relaxation (formula 5):

$$\max_h \sum_f w_f\, \mathbb{E}\big[\mathrm{F1}_f(h)\big] \tag{5}$$

Coverage is $C = | V_{\text{parsed}} | / | V_{\text{reachable}} |$at the configured depth limit. Fragment-level duplicates are filtered by Jaccard similarity $J(x, y) = | x \cap y | / | x \cup y |$with threshold θ; report pre/post-filter corpus sizes and the share of removed near-duplicates.

3.2. Crawler Architecture

The rapid progress of NLP research has created a sustained demand for scalable corpora of the Kazakh language. To address this, we designed a modular acquisition pipeline in which a Django server shell orchestrates the crawling process (seeding start URLs, monitoring job status), while the extraction core is implemented in Python using requests and BeautifulSoup; Scrapy is integrated when needed. This combination of crawler, web wrapper, and storage follows well-established engineering patterns and industrial web-scraping practices [23-24]. Results and metadata are normalized into a unified JSON representation, simplifying downstream analytics and integration with external APIs [25].

The architecture follows a layered design consisting of: (1) a control layer (Django interface), (2) a crawling and extraction layer, and (3) a storage layer. This separation of concerns improves reproducibility, maintainability, and scalability of the system.

The overall workflow of the crawler is illustrated in Fig. 3. The system starts from user-defined seed URLs, processes links through normalization and validation (including conversion of relative paths to absolute ones and duplicate filtering via a visited registry), and retrieves page content via HTTP requests. Successfully fetched pages (HTTP 200 OK) are parsed to extract structured information, followed by pre-processing steps such as removal of markup and scripts, whitespace normalization, and text cleaning. Extracted data are serialized into JSON format together with metadata (URL, headings, content fragments). Newly discovered links are iteratively added to the processing queue, enabling recursive traversal until the predefined depth or link frontier is exhausted.

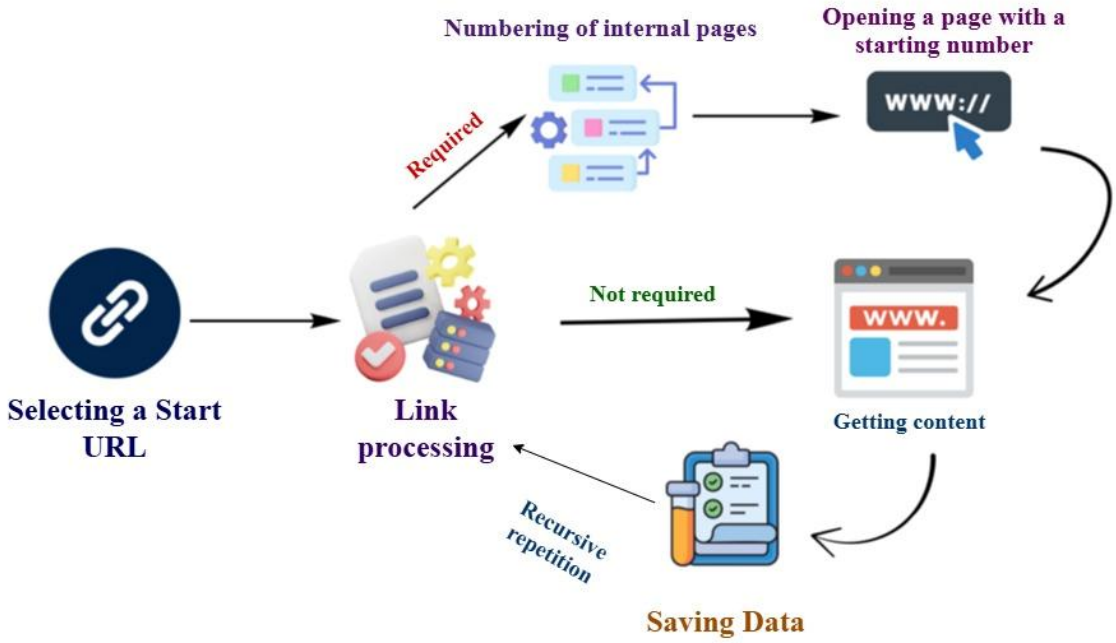

Figure 3. Breadth-first search (BFS) crawler architecture

To handle dynamically generated content, a headless rendering layer (e.g., Playwright or Selenium) can be optionally activated to obtain the final DOM before parsing. This hybrid design significantly improves extraction completeness and robustness to frontend changes and is consistent with recommended practices in modern web-scraping systems [23-24]. Overall, the integration of Django orchestration, lightweight parsing (requests/BeautifulSoup), optional headless rendering, and JSON-based storage provides a reproducible, scalable, and transparent pipeline for constructing Kazakh-language corpora in the e-government domain [25].

3.3. Crawling and Parsing: Navigation, Dynamics, and Normalization

The crawler operates by traversing the target website and extracting structured content from individual pages. Starting from one or more seed URLs, the system forms a processing queue and retrieves page content via HTTP requests. Each address undergoes normalization, including conversion of relative paths to absolute URLs and removal of tracking parameters, while a visited registry prevents duplicate processing. For sections with pagination, a dedicated module generates canonical links and preserves the hierarchical structure “category → subcategory → service card.”

After a successful server response (HTTP 200 OK), syntactic parsing is performed using BeautifulSoup to extract key elements such as headings, descriptive content blocks, and hyperlinks. Extracted links are deduplicated and reintroduced into the processing queue, enabling recursive traversal until a predefined depth limit or link frontier is reached. Error responses (e.g., 404/5xx) are handled through graceful skipping to avoid unnecessary load escalation.

Because a significant portion of content on government portals is rendered client-side, static extraction (requests + HTML parsing) is complemented by an optional headless-rendering layer. Depending on the site profile, Selenium, Playwright, or Puppeteer is used to render the final DOM, including dynamic widgets, tabs, and modal components. This hybrid approach substantially reduces data incompleteness typical of purely static methods and aligns with established practices in modern web scraping systems.

Following extraction, HTML/DOM normalization is applied prior to linguistic processing. All responses are converted to UTF-8 encoding; HTML entities (, <, >, &) are decoded; and non-informative elements (<script>, <style>, navigation blocks) are removed. The main textual content is identified using heuristic rules based on block length, structural consistency, and absence of boilerplate elements. When descriptions are distributed across multiple paragraphs, consecutive <p> elements are merged into coherent text segments while preserving logical structure.

To ensure data quality, duplicate and near-duplicate fragments are filtered, and the cleaned content is serialized into a structured JSON format. Each record contains key fields, including URL, category (derived from breadcrumbs), title, and description, along with execution metadata. This standardized representation supports downstream NLP tasks and integration with external systems [25].

The normalization and extraction procedures are specifically adapted to the linguistic characteristics of the Kazakh language. In particular, the pipeline accounts for code-switching between Kazakh and Russian, which is common in e-government content, and applies normalization of transliterated forms (e.g., Latin-to-Cyrillic conversion).

Furthermore, the system considers the agglutinative nature of the Kazakh language, where grammatical meaning is expressed through suffixes, which influences token boundaries and text segmentation. To address this, heuristic rules are applied to preserve semantically meaningful text blocks and avoid fragmentation of morphologically complex expressions.

In addition, domain-specific lexical patterns characteristic of the official-administrative register (e.g., service descriptions, procedural instructions) are used to improve the identification of relevant content. These adaptations distinguish the proposed pipeline from generic web scraping approaches and make it suitable for constructing high-quality Kazakh-language corpora.

3.4. Automatic Tagging: Taxonomy, Linguistic Processing, Normalization

Automatic tagging is organized as a multi-stage pipeline with a clear taxonomy of target fields and a blend of rules, lexicons, and statistical models. A core anchor lexicon encodes terms of the official–bureaucratic register and interface markers (e.g., section indicators such as “required documents,” “how to obtain,” and “timelines/fees”). On top of this layer, regular expressions and DOM patterns capture stable page structures (headings h1–h4, repeated card containers, tabular th/td pairs). In ambiguous cases, an LLM assistant is invoked under a weak-supervision regime: the model proposes tag candidates, while a rule-based resolver makes the final decision, ensuring reproducibility and traceability.

To route text correctly, block-level language identification (ru/kaz) is performed, followed by morphological disambiguation for Kazakh content. Normalization applies Unicode procedures (NFC/NFKC), removes transliteration variants (“qazaqstan” → “Қазақстан”, “bilim” → “білім”), standardizes phone and date formats, and canonicalizes named entities (agencies, institutions, geographic objects) using curated gazetteers. To prevent overfitting and distortions in statistics, deduplication proceeds in two passes: exact duplicates are removed first, then highly similar fragments are filtered by cosine similarity over embeddings; for each document, the source identifier and a content hash are retained.

The output is a validated JSON object conforming to the target schema: required fields, types, and admissible ranges are enforced by a schema validator; a tagging log records applied rules, lexicon versions, and model decisions. This setup combines the governability of a production

pipeline with the flexibility of linguistic methods and supports portability of the tagger across sections and related portals.

3.5 Crawler methods

To evaluate the effectiveness of different crawling strategies, we consider three configurations: Static crawler, Headless-All, and the proposed hybrid method (Crawler6).

The static crawler treats the public portal as a directed graph and traverses it using breadth-first search (BFS) to a fixed depth while fetching raw HTML without executing JavaScript. Field extraction (URL, category, title, description) relies on deterministic selectors and lightweight heuristics. While computationally efficient, this approach may lead to incomplete extraction for pages where content is rendered dynamically. BFS ensures systematic level-by-level traversal with a queue-managed frontier and guarantees coverage within the configured depth, making it suitable for structured portal exploration [26].

The Headless-All configuration follows the same traversal strategy but renders every page using a headless browser (e.g., Selenium or Playwright), allowing full execution of client-side scripts and access to the final DOM. This approach significantly improves extraction completeness, particularly for dynamic content such as tabs, forms, and interactive elements, but introduces substantial computational overhead due to per-page rendering cost [27].

The proposed method, Crawler6, introduces a budget-aware hybrid strategy that selectively applies headless rendering. The name “Crawler6” refers to the sixth experimental configuration evaluated in this study, corresponding to the optimal policy under resource constraints. Each page is assigned a priority score based on estimated DOM complexity or expected extraction gain, and headless rendering is applied greedily to the highest-priority pages until the predefined computational budget B is exhausted.

This decision process can be formulated as a resource allocation problem, analogous to a 0–1 knapsack optimization, where the objective is to maximize a weighted F1 score under time or cost constraints, with expected extraction gain representing value and rendering cost representing weight [28]. As a result, Crawler6 achieves a balance between extraction completeness and computational efficiency, making it suitable for large-scale corpus construction.

For evaluation, we report precision, recall, and F1 score per field, following standard information retrieval practices; the F1 score is defined as the harmonic mean of precision and recall [29]. To quantify uncertainty, non-parametric bootstrap confidence intervals are computed for F1 [30]. Statistical significance between crawling strategies on the same dataset is assessed using McNemar’s test on paired binary outcomes.

Results

To ensure a reliable and reproducible evaluation, the dataset of 213 service pages was constructed from the full corpus of 915 services across 12 categories of the egov.kz portal using a stratified random sampling strategy.

Specifically, pages were first grouped by category, and then a proportional number of pages was randomly selected from each category according to its size. This approach preserves the original distribution of categories in the corpus and avoids overrepresentation of high-resource categories such as Education and Real Estate.

At the same time, all categories were included in the evaluation to ensure coverage of both large and small groups (e.g., Consular services). As a result, the evaluation dataset reflects the structural and content diversity of the portal, providing a representative benchmark for assessing extraction quality and ensuring the validity and reproducibility of the reported results.

Using this evaluation dataset, we assessed field-extraction accuracy on 213 service pages using precision, recall, and F1-score (Equations (1)–(3)) together with budget-constrained optimization (Equation (4)).

With equal weights for TITLE and DESCRIPTION, the static pipeline achieved a weighted (F1) of 0.771 (95% CI: 0.734–0.804), whereas the fully dynamic Headless-All variant reached 0.939 (95% CI: 0.921–0.957). The proposed Crawler6 policy (the sixth experimental configuration

corresponding to the budget-optimized strategy), which allocates headless rendering to pages with the highest expected utility under a time budget (B), attained the same aggregate quality at ($B \approx 0.80$): running headless on 170 of 213 pages yielded a weighted (F1) of 0.939. The budget–quality frontier rises steeply from the static baseline and plateaus near the headless upper bound around ($B \approx 0.8$), indicating that Crawler6 approaches maximal quality while incurring substantially lower average rendering cost than Headless-All.

A per-field analysis shows that TITLE is extracted perfectly by all three pipelines (F_1= 1.000), while performance diverges on DESCRIPTION. The static pipeline obtains (F_1= 0.541) (95% CI: 0.468–0.608), Headless-All achieves ($F_1 = 0.879$) (95% CI: 0.842–0.910), and Crawler6 closely tracks the dynamic upper bound with ($F_1 \approx 0.878$), reflecting the policy's focus on pages most susceptible to incomplete DOMs under static loading. Paired McNemar tests against the static baseline show no difference on TITLE ($p \approx 1.0$) but a highly significant improvement for DESCRIPTION for both Headless-All and Crawler6 ($p < 0.001$). Collectively, these results demonstrate that selective headless rendering, as operationalized by Crawler6, restores completeness on the difficult field and delivers Headless-All–level quality at a reduced computational budget, making it a practical choice for production settings with resource constraints.

Table 2. Quality and Budget Trade-offs of Three Crawling Policies (Static, Headless-All, and Crawler6) on N=213 Pages

№	Crawler parameters	Methods		
		Static baseline	Headless-All	Crawler6
1.	N pages	213	213	213
2.	Pages_headless	0	213	170
3.	Budget_fraction	0	1	0,8
4.	TITLE F_1	1	1	1
5.	TITLE F_1 CI_{low}	1	1	1
6.	TITLE $F_1 CI_{high}$	1	1	1
7.	DESC F_1	0,541	0,879	0,878
8.	DESC F_1 CI_{low}	0,468	0,842	0,842
9.	DESC F_1 CI_{high}	0,608	0,91	0,91
10.	Weighted F_1	0,771	0,939	0,939
11.	Weighted F_1 CI_{low}	0,734	0,921	0,921
12.	Weighted F_1 CI_{high}	0,804	0,957	0,957
13.	$McNemar_p$ TITLE vs Static		1	1
14.	$McNemar_p$ DESC vs Static		0	0

The goal of the experiment was to evaluate how the proposed pipeline – crawler → parsing → preprocessing → automatic tagging → JSON export – handles real, heterogeneous layouts typical of government web resources. Within the experiment, only safe, unauthenticated HTTP GET requests were executed; response encodings were forcibly set to UTF-8, and HTML was parsed with BeautifulSoup. The system is deployed as a Django application and performs the full cycle: from providing a starting URL to obtaining structured JSON with annotations of key service-card fields. Figure 4 shows the initial page of the web-scraping application.

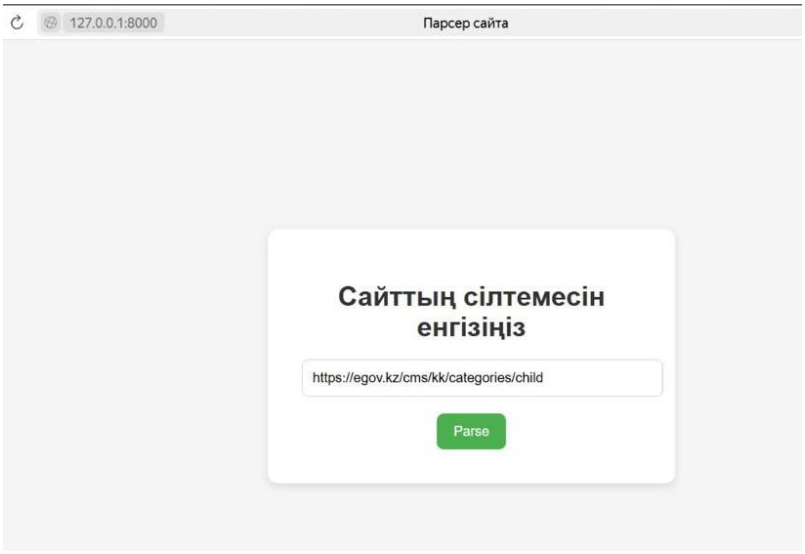

Figure 4. Initial page of the web-scraping application

Architecturally, the solution is modular: request routing (Django URL dispatcher), a crawler/parser business-logic layer (Python + requests/BeautifulSoup), a presentation layer (templates for URL input and result viewing), and session-based management of transient state. This separation of concerns isolates data-extraction experiments from the web wrapper and ensures reproducibility.

Navigation is implemented as breadth-first search: starting from the user-provided seed URL, the system builds a link queue, then sequentially fetches pages, extracts new <a href> targets from the DOM, and enqueues them until the frontier is exhausted (see Figure 5). For robustness, response-status checks, graceful handling of network/HTTP errors, and skipping of pages with malformed layouts are provided. When needed for dynamic sections, a headless rendering layer is enabled, and the final DOM is analyzed thereafter.

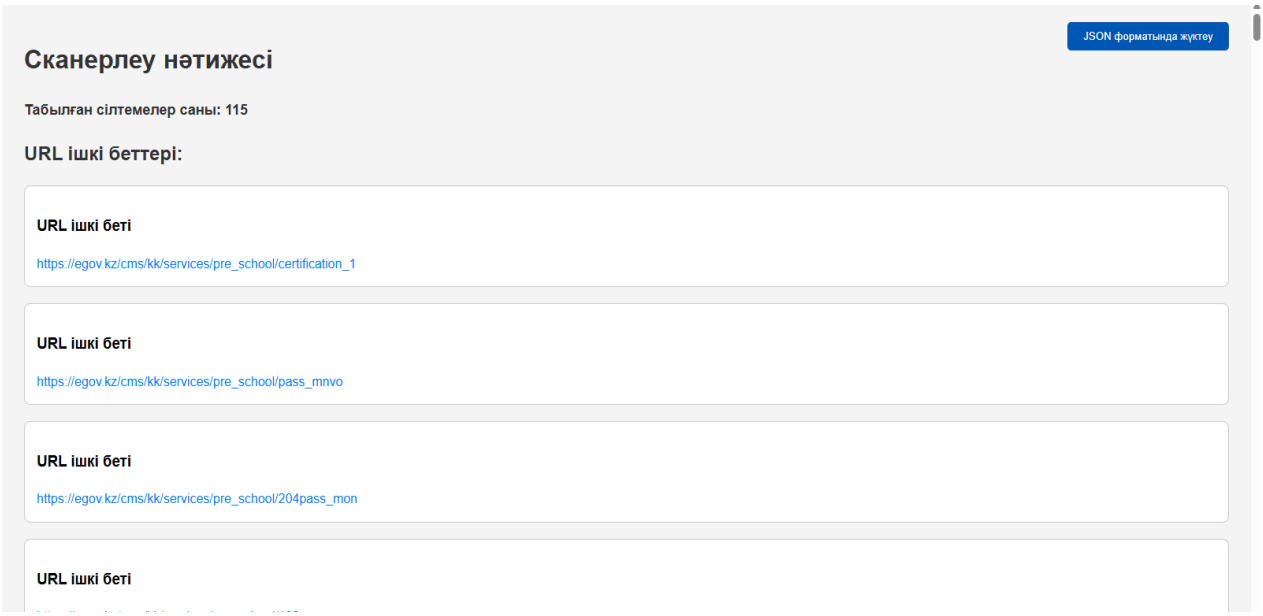

Figure 5. List of links from the user-specified start URL

After scanning all the links, you need to click on the button in the upper right corner to download the extracted data in json format. Figure 6 shows structured data in json format, here you need to take into account that all services from 1 categories are collected in 1 json.

```
{
    "title": "Алты объектіден он объектіге дейін объектілерден тұратын меншік, шаруашылық
    "url": "https://egov.kz/cms/kk/online-services/for_citizen/epay006_gs",
    "description": "Алты объектіден он объектіге дейін объектілерден тұратын меншік, шару
    "categories": ["Жылжымайтын мүлік"],
    "keywords": ["мүлік","құқықтық тіркеу","баж салығы"],
    "instructions": "Онлайн төлеу\\n\\nОнлайн төлеу батырмасы арқылы өтіңіз.\\nСалық орга
},
{
    "title": "Жылжымайтын мүлік объектісінің құқық иеленушісі үшін мемлекеттік баж салығы
    "url": "https://egov.kz/cms/kk/online-services/for_citizen/epay013_gs",
    "description": "Жылжымайтын мүлік объектісінің құқық иеленушісі деректерінің, сәйкест
    "categories": ["Жылжымайтын мүлік"],
    "keywords": ["жылжымайтын мүлік","иеленуші","баж салығы"],
    "instructions": "Қалай төлеуге болады\\n\\n«Электрондық үкімет» төлем шлюзінің бетінд
},
{
    "title": "Меншік иесі, жерді пайдалану, жер учаскесіне өзге құқықтарды тіркеу үшін ме
    "url": "https://egov.kz/cms/kk/online-services/for_citizen/epay009_gs",
    "description": "Меншік иесі, жерді пайдалану, жер учаскесіне өзге құқықтарды (құқықта
    "categories": ["Жылжымайтын мүлік"],
    "keywords": ["Меншік иесі","тіркеу","баж салығы"],
    "instructions": "Онлайн төлеу\\n\\n«Электрондық үкімет» төлем шлюзінің бетіндегі «Төл
},
```


Figure 6. Collected data in JSON format

Running the pipeline across the 12 categories yielded a normalized set of service cards. For each card, the system automatically annotates a minimal yet sufficient set of fields for downstream analysis: url, category, Title, and descriptions. The records are serialized into a single JSON array and are available both for visual inspection in the interface and for download via a utility endpoint.

Thus, the experiment confirmed the viability and portability of the proposed scheme: the modular Django-based pipeline enables controlled traversal, robust parsing of heterogeneous DOM structures, and reproducible tagging of target fields with subsequent export to JSON – providing a solid foundation for further linguistic processing and analytics.

Discussion

To position the proposed approach relative to existing web crawling frameworks, Table 3 presents a comparison between traditional static crawlers, fully headless browser-based methods, and the proposed hybrid pipeline.

Table 3. Comparison of crawling approaches: static, fully headless, and hybrid (proposed)

Method	Dynamic Content Support	Computational Cost	Extraction Completeness	Adaptation for NLP Corpus	Reproducibility
Static crawler (Scrapy, BS)	No	Low	Medium	Limited	High
Headless-All (Selenium)	Yes (full rendering)	High	High	Limited	Medium
Hybrid (proposed, Crawler6)	Yes (selective)	Medium	High	High	High

As shown in Table 3, static crawlers are computationally efficient but often fail to capture dynamically generated content, resulting in incomplete extraction. Fully headless approaches ensure high completeness but incur significant computational overhead. In contrast, the proposed hybrid strategy (Crawler6) achieves a balance by selectively applying headless rendering only

when necessary, allowing the system to maintain high extraction quality while reducing resource consumption. Furthermore, the pipeline is specifically adapted for corpus construction in the Kazakh language, incorporating normalization and structured output aligned with downstream NLP tasks.

The presented pipeline demonstrates that, for Kazakh-language e-government domains, it is feasible to build a reproducible scheme for collection, normalization, and initial linguistic annotation without default reliance on heavy browser automation. In contrast to fully dynamic crawling approaches, the proposed method selectively activates headless rendering only for pages where static parsing under-captures content, thereby reducing computational cost while preserving extraction completeness.

In terms of coverage, the site is treated as a directed graph, and breadth-first traversal systematically "maps" sections while preserving page hierarchy. In a demonstration run on the "Real Estate" category, 115 service cards were obtained; overall, the pipeline spans 12 categories and aggregates 915 service descriptions, forming a targeted subcorpus of official-administrative register. Thus, the "Django + requests/BeautifulSoup (+ Playwright/Selenium on demand) + JSON" architecture proves suitable for reproducible collection and rapid integration with downstream analytics [23–25].

From a data-quality perspective, three components were decisive. First, the tag taxonomy and auto-labeling rules – grounded in DOM patterns and breadcrumbs – provide robust extraction of url, category, Title, and descriptions despite moderate frontend changes. Second, normalization (NFC/NFKC, de-transliteration "qazaqstan → Қазақстан," unified phone formats, boilerplate removal) reduces noise and prepares material for morphological processing and subsequent training annotations for LLMs [11-12]. Third, deduplication based on fragment similarity limits "soft duplicates" arising from navigational overlaps. Together, these steps increase the useful density of the corpus and make it more suitable for entity extraction, classification, and stylistic adaptation.

Comparison with prior work confirms general trends: static scraping remains the workhorse for sites with stable HTML, while browser automation is best applied selectively – to dynamic content and interactive widgets [1–3]. For building thematic subcorpora of government portals, such "on-demand hybridization" is more practical than ubiquitous headless rendering, especially given ethical constraints and "polite" access policies.

The approach has limitations and threats to validity. The architecture is sensitive to DOM restructurings and CSS classes; multi-heuristic container search mitigates this only partially, and major redesigns require rule updates. Late-loaded dynamic content can still be missed by static passes; enabling the headless layer addresses this at a performance cost. Linguistic factors – code-switching (ru/kaz), orthographic variability, agglutinative morphology – impose requirements on later stages: reliable NER/POS needs domain lexicons and disambiguation, plus expanded normalizers for regional variants [11-12]. Finally, legal and ethical aspects call for formalized procedures: automated robots.txt checks, throttling with jitter, conditional GETs, and an access-transparency log should be part of a production deployment, as recommended by industry guidance [23–25].

We assess portability as high for portals with similar information architecture: the presence of breadcrumbs, repeatable cards, stable headings, and tables makes extraction rules broadly reusable, with source changes reduced to selector and anchor-lexicon retuning. Practical scenarios include regular subcorpus refresh for adapting LLMs to the official register, building search and QA indices, and monitoring changes in regulatory texts. Over the long term, the quality and transparency of the collection pipeline will be the limiting factor for downstream models: improvements in access ethics, quality control, and metadata documentation will translate directly into robustness and reproducibility.

The path forward is clear. We plan to institutionalize access policy and reporting (identifiable User-Agent, contact, purpose of use), expand automatic noise filters and JSON-

schema validation, add retries with exponential backoff and caching, and incorporate completeness assessment against category-level gold slices. On the language side, we will strengthen normalizers, add rules for dialectal forms, and integrate Kazakh morphological resources into the annotation loop. This roadmap will turn the demonstration prototype into a durable production platform for curating Kazakh-language e-government corpora.

Conclusion

The paper presents a reproducible pipeline for collecting and linguistically processing Kazakh-language texts from the governmental domain. We design a modular architecture in which Django orchestrates execution and logging; the extraction core is built on requests/BeautifulSoup with an attachable headless layer (Playwright/Selenium) for dynamic elements; and results are normalized and published under a unified JSON schema. Site traversal is treated as exploration of a directed graph; a breadth-first search strategy provides systematic section coverage, preserves hierarchy, and enables controlled corpus growth. Automatic tag annotation relies on DOM patterns, breadcrumbs, and block-level heuristics, yielding the minimally sufficient field set (url, category, Title, descriptions) for downstream analytics and LLM adaptation.

Experimental evaluation on the e-services portal shows that the proposed "static parsing + on-demand headless" hybrid reduces costs while improving recall on complex table/list structures. The pipeline covers 12 categories and 915 service cards; for the "Real Estate" category, 115 pages were automatically collected and processed. Built-in normalization (NFC/NFKC, de-transliteration, boilerplate filtering) and deduplication increase the useful data density and prepare the material for morphological disambiguation and entity-extraction tasks.

Limitations stem from sensitivity to DOM changes, residual dynamic content, and linguistic specifics of Kazakh (code-switching, orthographic variation, agglutination), as well as access-ethics requirements. Nevertheless, the architecture transfers readily to portals with similar information structures and can serve as a backbone for regular updates of Kazakh-language subcorpora in the official-administrative register.

Future work includes formalizing access and transparency policies (robots.txt compliance, throttling with jitter, conditional GETs, retries), expanding the tag taxonomy with an LLM assistant, strengthening normalizers and disambiguation rules, enforcing JSON-Schema validation, and releasing representative gold standards for measuring coverage/precision. Implementing these steps will turn the prototype into a robust production platform for e-gov analytics and Kazakh-language NLP, ensuring high data quality and reproducible results.

Data Availability

The source code, crawling scripts, and sample dataset used in this study are publicly available in the following repository: https://github.com/AsselOspan/Crawler6_EGOV_KZ

Acknowledgment

This research was funded by the Science Committee of the Ministry of Science and Higher Education of the Republic of Kazakhstan, grant number BR24993001 "Creation of a large language model (LLM) to maintain the implementation of Kazakh language and increase the technological progress".

References

[1] Lotfi, C., Srinivasan, S., Ertz, M., & Latrous, I. (2021). Web scraping techniques and applications: A literature review. *In Advances in Data Science and Management* (pp. 381–394). https://doi.org/10.52458/978-93-91842-08-6-38

[2] Pichiyan, V., Muthulingam, S., Sathar, G., Nalajala, S., Ch, A., & Das, M. N. (2023). Web scraping using natural language processing: Exploiting unstructured text for data extraction and analysis. *Procedia Computer Science*, 230, 193–202. https://doi.org/10.1016/j.procs.2023.12.074

[3] A. Ospan, A. Mussa, M. Mansurova and T. Sarsembayeva, "LLM Agents for Enhanced Tabular Data Interpretation: A Perspective," *2025 IEEE 5th International Conference on Smart Information Systems and Technologies (SIST)*, Astana, Kazakhstan, 2025, pp. 1-6, doi: 10.1109/SIST61657.2025.11139242 .

[4] Abodayeh, A., Hejazi, R., Najjar, W., Shihadeh, L., & Latif, R. (2023). Web scraping for data analytics: A BeautifulSoup implementation. *Proceedings of the Sixth International Conference of Women in Data Science at Prince Sultan University* (WiDS PSU), 65–69. https://doi.org/10.1109/WiDS-PSU57071.2023.00025

[5] Kazmali, A. S., & Sayar, A. (2025). Web scraping: Legal and ethical considerations in general and local context – A review. *Procedia Computer Science*, 259, 1563–1572. https://doi.org/10.1016/j.procs.2025.04.111

[6] García, B., del Alamo, J. M., Leotta, M., & Ricca, F. (2024). Exploring browser automation: A comparative study of Selenium, Cypress, Puppeteer, and Playwright. In A. Bertolino, J. Pascoal Faria, P. Lago, & L. Semini (Eds.), Quality of Information and Communications Technology (QUATIC 2024), Communications in Computer and Information Science (Vol. 2178, pp. xx–xx). *Springer, Cham*. https://doi.org/10.1007/978-3-031-70245-7_10

[7] Mansurova, M., Barakhnin, V., Ospan, A., & Titkov, R. (2023). Ontology-driven semantic analysis of tabular data. *Applied Sciences*, 13(19), 10918. https://doi.org/10.3390/app131910918

[8] Colla, D., Mensa, E., & Radicioni, D. P. (2020). LessLex: Linking multilingual embeddings to sense representations of lexical items. *Computational Linguistics*, 46(2), 289–333. https://doi.org/10.1162/coli_a_00375

[9] Ehrmann, M., Hamdi, A., Pontes, E. L., Romanello, M., & Doucet, A. (2023). Named entity recognition and classification in historical documents: A survey. ACM Computing Surveys, 56(2), *Article* 27, 1–47. https://doi.org/10.1145/3604931

[10] Supriyono, Wibawa, A. P., Suyono, & Kurniawan, F. (2024). A survey of text summarization: Techniques, evaluation and challenges. *Natural Language Processing Journal*, 7, 100070. https://doi.org/10.1016/j.nlp.2024.100070

[11] Widyassari, A. P., Rustad, S., Shidik, G. F., Noersasongko, E., Syukur, A., Affandy, A., & Setiadi, D. R. I. M. (2022). Review of automatic text summarization techniques & methods. *Journal of King Saud University – Computer and Information Sciences*, 34(4), 1029–1046. https://doi.org/10.1016/j.jksuci.2020.05.006

[12] Ferrara, E., De Meo, P., Fiumara, G., & Baumgartner, R. (2014). Web data extraction, applications and techniques: A survey. *Knowledge-Based Systems*, 70, 301–323. https://doi.org/10.1016/j.knosys.2014.07.007

[13] Abdelrazek, A., Eid, Y., Gawish, E., Medhat, W., & Hassan, A. (2023). Topic modeling algorithms and applications: A survey. *Information Systems*, 112, 102131. https://doi.org/10.1016/j.is.2022.102131

[14] Kozhirbayev, Z., & Yessenbayev, Z. (2020). Kazakh text normalization using machine translation approaches. *CEUR Workshop Proceedings*, 2780, 115–122. http://ceur-ws.org/Vol-2780/paper10.pdf

[15] Rapisheva, Zh. D., Rakhymberlina, S. A., Akisheva, Zh. S., & Akshabaeva, L. M. (2023). Features of application of single- and multi-component terms in the Kazakh official style. *Bulletin of the Karaganda University. Philology Series*, 112(4), 67–72. https://doi.org/10.31489/2023ph4/67-72

[16] Tolegen, G., Toleu, A., & Mussabayev, R. (2024). Contrastive learning for morphological disambiguation. *Applied Sciences*, 14(21), 9992. https://doi.org/10.3390/app14219992

[17] Kim, S., Park, H., & Lee, J. (2020). Word2vec-based latent semantic analysis (W2V-LSA) for topic modeling: A study on blockchain technology trend analysis. *Expert Systems with Applications*, 152, 113401. https://doi.org/10.1016/j.eswa.2020.113401

[18] Yoon, S. H., & Kim, K. H. (2021). Expansion of Topic Modeling with Word2Vec and Case Analysis. *The Journal of Information Systems*, *30*(1), 45–64. https://doi.org/10.5859/KAIS.2021.30.1.45

[19] Yang, J., Yang, B., Sun, Q., Yan, S., & Miao, Y. (2022). Research on the key technology of web data extraction and mining based on the probability distribution. *Wireless Communications and Mobile Computing,* 2022, 6714785. https://doi.org/10.1155/2022/6714785

[20] Bhatt, C., Bisht, A., Chauhan, R., Vishvakarma, A., Kumar, M., & Sharma, S. (2023). Web scraping techniques and its applications: A review. *In Proceedings of the 3rd International Conference on Innovative Sustainable Computational Technologies (CISCT)* (pp. 1–8). https://doi.org/10.1109/CISCT57197.2023.10351298

[21] Kumar, N., Lohani, D., & Acharya, D. (2022). Vehicle accident sub-classification modeling using stacked generalization: A multisensor fusion approach. Future Generation *Computer Systems*, 133, 39–52. https://doi.org/10.1016/j.future.2022.03.005

[22] Mansurova, M., Barakhnin, V., Ospan, A., & Titkov, R. (2023). Ontology-Driven Semantic Analysis of Tabular Data: An Iterative Approach with Advanced Entity Recognition. *Applied Sciences*, *13*(19), 10918. https://doi.org/10.3390/app131910918

[23] Krotov, V., Johnson, L., & Silva, L. (2020). Legality and ethics of web scraping. *Communications of the Association for Information Systems*, 47, 539–563. https://doi.org/10.17705/1CAIS.04724

[24] Nguyen, T. H. (2018). Deep learning for information extraction. *In Proceedings of the Conference on Empirical Methods in Natural Language Processing*. https://api.semanticscholar.org/CorpusID:248557100

[25] Chai, C. P. (2023). Comparison of text preprocessing methods. *Natural Language Engineering*, *29*(3), 509–553. doi:10.1017/S1351324922000213

[26] Gupta, R., et al. (2024). Generative AI: A systematic review using topic modelling techniques. *Data and Information Management*, 8(2), 100066. https://doi.org/10.1016/j.dim.2024.100066

[27] Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., & Amodei, D. (2020). Language models are few-shot learners. *arXiv*. https://doi.org/10.48550/arXiv.2005.14165

[28] Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., Rodriguez, A., Joulin, A., Grave, E., & Lample, G. (2023). LLaMA: Open and efficient foundation language models. *arXiv*. https://doi.org/10.48550/arXiv.2302.13971

[29] OpenAI. (2023). GPT-4 technical report. *arXiv.* https://doi.org/10.48550/arXiv.2303.08774

[30] Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-T., Rocktäschel, T., Riedel, S., & Kiela, D. (2020). Retrieval-augmented generation for knowledge-intensive NLP tasks. *arXiv.* https://doi.org/10.48550/arXiv.2005.11401