Wikipedia-grounded Dataset for Question Answering, Verification, and Text Generation for Ukrainian Language

Volodymyr Taranukha[1,*,†], Oleksandr Marchenko[1,†], Anatoly Anisimov[1,†] and Emil Nasirov[1,†]

[1] *Institute of Information Technologies and Systems, 40, Acad. Glushkova av., Kyiv, 03187, Ukraine*

Abstract

There are few high-quality corpora for the Ukrainian language. Moreover, while modern LLMs benefit from scale, applications benefit more from task-centric corpora aligned with downstream tasks. Wikipedia is a large, community-curated, well-structured and richly cited resource. This article explains how we leverage Wikipedia to build task-centric corpora for the Ukrainian language.

Keywords

dataset, Ukrainian language, Wikipedia, ODQA, fact verification, text generation

1. Introduction

Modern LLMs benefit from scale but applications benefit more from task-centric corpora aligned with downstream use - open-domain question answering (ODQA), multi-hop reasoning, fact verification, and data-to-text generation. Wikipedia is a uniquely suitable source: community-curated, richly structured (sections, hyperlinks, infoboxes, tables), and licensable for research. There are some good modern benchmarks and resources for this task.

Wikipedia-anchored benchmarks established viable retrieval-augmented and evidence-grounded pipelines at scale. WIKITABLET [1] show how to condition generation on section structure and table snippets, so models practice section-aware content planning on encyclopedic inputs. DrQA/Reading Wikipedia to Answer Open-Domain ToTTo[2] gives example how table-to-text items in training sample has to include highlighted cell selections and lexical constraints, which reduces hallucinations and teaches controlled generation from structured data. KILT [3] shows that each item can be bound to a stable Wikipedia snapshot via page IDs and character offsets, enabling deterministic rebuilds, disambiguated entity targets, and leakage checks between train/dev/test. PAQ [4] demonstrates scale and paraphrase coverage from auto-generated questions. Such generation must be used carefully by keeping source passages for each question, deduplicating near-matches, and preserving paraphrase clusters allowing to expand coverage without spurious patterns. BEIR offers a heterogeneous retrieval benchmark spanning fact checking, QA, and more, suitable for zero-shot retrieval evaluation [5]. MIRACL provides expert-

Information Technology and Implementation (IT&I-2025), November 20-21, 2025, Kyiv, Ukraine

* Corresponding author.

† These authors contributed equally.

volodymyr.taranukha@gmail.com (V. Taranukha); omarchenko@univ.kiev.ua (O. Marchenko); anatoly.v.anisimov@gmail.com (A. Anisimov); enasirov@gmail.com (E.Nasirov)

0000-0002-9888-4144 (V. Taranukha); 0000-0002-5408-5279 (O. Marchenko); 0000-0002-1467-2006 (A. Anisimov); 0009-0006-9016-2602 (E.Nasirov)

annotated, multilingual monolingual retrieval over Wikipedia across 18 languages (not including Ukrainian), aligning with modern IR pipelines[6]. MuSiQue constructs harder, less "shortcut-prone" multi-hop questions, matching LLM reasoning evaluation[7]. AmbiQA and successors model questions with multiple valid interpretations and answers, recent work on generative disambiguation sharpens evaluation under ambiguity [8]. FEVEROUS extends FEVER to require structured + unstructured evidence, closer to real Wikipedia use [9]. AmbiFC introduces 10k claims with fine-grained evidence, advancing robustness beyond single-evidence claims [10]. However, all mentioned works do not support Ukrainian language.

The authors have experience in designing linguistic resources and datasets [11, 12, 13]. In order to facilitate further research in Computational Linguistics for Ukrainian language we aim to design and implement a reproducible Ukrainian language centered Wikipedia-to-Tasks pipeline that will do the following:

- robustly extracts and cleans content from MediaWiki snapshots;
- performs adaptive chunking that respects section/paragraph boundaries and augments queries with retrieval context;
- builds lexical, dense, and hybrid indices with explicit provenance;
- generates and normalizes instruction and QA pairs supported by retrieval evidence; and
- filters low-quality or duplicate items (including context–answer mismatches by embedding similarity) with targeted human review.

2. Related Works

True one-to-one Ukrainian counterparts to English ODQA-style datasets remain scarce. Nevertheless, several published resources can serve as practical surrogates.

MKQA provides aligned question–answer pairs across 26 languages (including Ukrainian) for open-retrieval evaluation [14]. Multilingual QA pairs not bound to a specific Wikipedia snapshot or to passage-level provenance. Useful for question form templates and answer normalization rules. For our task, we must mine evidence on Ukrainian Wikipedia and bind each item to page/section IDs to make it usable for training, as an evaluation slice it is immediately helpful for out-of-domain and cross-lingual checks.

UNLP 2024's shared task materializes community QA efforts in Ukrainian using exam-style questions (ZNO) [15]. It is not not Wikipedia-grounded, therefore not a direct training source for passage-anchored QA. It can be used as an out-of-domain probe to test factual robustness and instruction-following in Ukrainian.

The Eval-UA-tion 1.0 suite publishes Ukrainian evaluation tasks for sanity checks and model selection [16]. It is a broad Ukrainian evaluation battery with no Ukrainian Wikipedia passage provenance. It can be useful after training to check tokenization/morphology and to detect regressions unrelated to retrieval.

For summarization, XL-Sum includes Ukrainian and is published in ACL Findings 2021 [17]. It is news-style summarization pairs (non-encyclopedic). It is useful as auxiliary at best: it provides brevity/faithfulness signals before fine-tuning on ukwiki, will not work a direct source of passage-grounded QA.

NER-UK 2.0 supplies manual entity annotations supporting entity-centric retrieval and highlighting [18]. It is a high-quality Ukrainian entity annotations. Project onto ukwiki chunks to enrich entity fields in the index, supervise evidence span highlighting, and evaluate entity fidelity in generated answers.

TRWU contributes Ukrainian news messages with stance/sentiment/bias labels for robustness experiments [19]. It contains non-encyclopedic data, it can be kept for robustness (behavior on

noisy inputs). Label schemes can inspire optional classification sub-tasks (e.g., talk/controversy pages) if we later extend beyond QA.

3. Source acquisition and filtering

The Wikimedia Dumps portal documents the pages-articles.xml.bz2 files, directory structure, and checksums. We pinned the corpus to a single Wikimedia dump snapshot of the target Wikipedia (e.g., ukwiki-20250401-pages-articles.xml.bz2) to guarantee reproducibility and leakage control. We pulled the dump from https://dumps.wikimedia.org/ukwiki/20250401/ using wget with resumed transfers and verified integrity against the published checksums. We recorded the snapshot date, dump file names, file sizes, and checksums in a machine-readable manifest (snapshot.json). We chose the pages-articles dump (namespace 0) so only encyclopedic articles are included by default. We also archived the stub-meta-current.xml.gz for quick metadata lookups. The manifest stores {project: "ukwiki", snapshot: SNAP, dump: filename, md5} and is checked into version control.

Grounding all tasks in a single Wikipedia snapshot is the standard practice that underpins reproducibility, comparability, and leakage control in knowledge-intensive benchmarks (e.g., KILT). Binding our pipeline to a specific snapshot ensures that retrieval sets and generated examples are stable across reruns and across research groups. Compared to pipelines that mix API reads with live content, our snapshot discipline yields deterministic corpora, mirroring best practice in KILT and BEIR while focusing on Ukrainian. This improves auditability (we can always point to the exact article revision) and allows fair head-to-head retriever evaluations.

We converted wikitext to cleaned plain text using WikiExtractor[20] with JSON output enabled, then performed a light second-pass cleanup with mwparserfromhell for edge cases. Each output record contains {id, title, url, lang, revision, section_path, text}. WikiExtractor is the de-facto tool for transforming Wikipedia dumps into plain text, handling templates and producing scalable shards, mwparserfromhell[21] adds robust programmatic markup handling for edge cases. Using both minimizes noise without losing provenance. Some public corpora ship only monolithic text or rely on ad-hoc regex cleaning. Our extractor pipeline outputs structured JSONL with section paths, enabling section-aware retrieval and generation. This improves downstream RAG quality and facilitates manual audits (reviewers can jump to the exact section).

Although pages-articles already targets namespace 0, we applied additional page-type and quality filters before indexing:

1. Redirects & disambiguation: dropped pages beginning with #REDIRECT and those containing common disambiguation templates, we also filtered pages whose entire body is a bulleted list of links (list-pages and set-index pages).
2. Minimum content: required ≥1,000 bytes of cleaned text, ≥2 headings, and ≥3 sentences, we removed articles dominated by tables/lists (>70% non-prose).
3. Structural sanity: excluded pages with extreme template density (ratio of markup tokens to words) and pages with no alphabetic tokens in the first paragraph.

Retrieval systems are sensitive to noisy or weakly structured pages, filtering out non-prose content improves dense retriever effectiveness and reduces hallucination in RAG. Benchmarks such as BEIR and MIRACL curate Wikipedia content with quality controls (native judgments, explicit exclusions), which supports our conservative filtering stance. Relative to English-centric pipelines (KILT/BEIR), our Ukrainian-first filtering aggressively removes list-pages/disambiguation pages that often poison top-k retrieval. This is expected to lower the proportion of non-answerable passages retrieved by >10% while retaining coverage of long-tail topics.

Next sub-step is provenance & audit trail. Precise provenance enables error analysis and deterministic rebuilds—key requirements echoed across KILT, BEIR or FEVEROUS. Maintaining revision IDs supports future diff-based updates without full re-extraction. Now every passage

carries strong provenance: {page_id, revision_id, title, url, section_path, snapshot_id, timestamp, lang}. We computed a stable document hash (SHA-1 of normalized wikitext) and stored it in both the JSONL[22] and the manifest. During ingestion we wrote structured logs per step (download, extract, clean, filter), each with counters and timing information. Many open releases expose only raw text or opaque indices. Our intermediate results are inspection-friendly and ready for hybrid retrieval, with enough metadata to reproduce any result or review any generated answer.

4. Chunking and Tokenization

Our segmentation design converts article-level text into retrieval-ready passages with the following requirements: indexability under subword token limits of modern encoders, contextual coherence sufficient for sentence-level factual questions, and consistency across languages (Ukrainian primary, multilingual optional) if possible.

We segment each cleaned article into contiguous, fixed-length token windows of size (L={100, 200, 300}) with overlap O (default O=0.2L) using the tokenizer aligned to the downstream encoder, so consecutive windows start at token indices.

We adopted the tokenizer of the encoder actually used for passage representation to eliminate train–test mismatches in subword boundaries. Our default tokenizer is BERT-family WordPiece[23] (30k vocabulary, using [CLS]/[SEP] specials), but the pipeline is model-agnostic, so other models can be used also, such as SentencePiece (Unigram/BPE), byte-level BPE, and variants required by multilingual encoders (e.g., XLM-R[24], mDeBERTaV3[25]) and sentence-embedding models (e.g., E5[26]/BGE[27]). (Discussion on other tokenizers is placed in Discussion section). In masked-LM encoders, subword tokenizers (such as WordPiece, SentencePiece) stabilize OOV handling in morphologically rich languages and reduce vocabulary size without sacrificing coverage.

The window family is motivated by prior retrieval-augmented systems that operate on short passages (approx. 100–200 tokens), and by recent analyses of long-context behavior indicating sensitivity to the position of evidence. Overlap protects discourse continuity across boundaries, which mitigates the well-documented tendency of current LMs to under-utilize information located away from the beginning or end of the prompt, by ensuring that the same sentence can occur near the edge in one chunk and near the center in the next, we reduce position-sensitivity during retrieval and generation. For sections shorter than L it is accepted that they form a single passage, very long sentences are allowed to straddle boundaries to preserve token-level determinism. However, we ensure at least one boundary falls at whitespace. We record token_span and char_span for every passage to retain reproducibility and facilitate highlighting.

We performed simple test on a small subset of our corpus with bag-of-words representation and queries derived from section leads. We compared different values of L (100,200,300) with different overlaps O (0.1, 0.2, 0.3) under a simple retriever. Shorter windows (100–200) consistently showed better retrieval precision at short queries, while 300-token windows slightly improved recall for multi-sentence questions. Overlap at 0.2 was found as optimal point between index growth and recall. Going for 0.3 offered limited additional benefit. These patterns accord with prior literature that emphasizes short, self-contained passages for retrieval-augmented QA.

Fast tokenizers provide deterministic normalization and alignment maps, enabling exact reconstruction of text spans for audits. This matters for retrieval because span misalignment degrades dense similarity and complicates evidence highlighting. Finally, overlap increases index size, we mitigate this with deduplication keyed by (page_id, token_span) and by pruning windows that are near-empty after cleaning.

There is a certain risk in such approach since fixed windows can split discourse phenomena (anaphora, enumerations). There are other options but we decided against using them in spite of mentioned risks due to their disadvantages.

Sentence-aware splitting uses sentence boundaries to avoid mid-sentence cuts, optionally packing adjacent sentences until a token budget is reached. It improves readability and fewer dangling coreferences, but there are disadvantages: reliance on sentence segmenters that may degrade on titles, lists, or scientific notation and introduce language-specific errors.

Semantic (embedding-aware) splitting provides adaptive boundaries chosen by embedding similarity between adjacent sentences, chunks expand or contract to keep intra-chunk coherence high. It often produces fewer irrelevant tokens per chunk, but it has disadvantages: higher preprocessing cost and potential domain drift if the splitter's embeddings differ from the retriever's.

Hierarchical or hybrid splitting organizes passages at multiple granularities (section to paragraph to token windows) to enable coarse-to-fine retrieval. It is flexible with recall/precision trade-offs, but it has such disadvantages as index complexity and fusion logic at query time.

We adopted fixed token windows as the primary method for their determinism, speed, and alignment with the chosen encoder's tokenization. However, our code path keeps sentence-aware and semantic splitters as pluggable components for ablations.

5. Storing Data

5.1. Storage of chunks in an indexed JSON format

We adopted a line-oriented JSONL corpus as the basis store for all passage-level artifacts produced after tokenization and segmentation. JSONL's line-granularity supports incremental processing, graceful degradation (malformed lines quarantined without corrupting neighbors), and easy diffing across releases. It also simplifies re-tokenization experiments: the same JSONL payload can be re-tokenized into alternative chunkings without re-extracting Wikipedia content, preserving provenance fields verbatim across regenerations.

Although JSONL lacks intrinsic indexing, line boundaries permit offset indexing without parsing the entire file. During writing we captured tell() positions and compressed-block boundaries, at finalize time we produced an index table (SQLite) keyed by doc_id. This way, retrieving k passages from disparate shards requires only k random seeks and decompression of at most k gzip members.

Each passage is recorded as a self-contained JSON object on a separate line, enabling stream processing and line-granular recovery. We bound every passage to an immutable, 64-bit doc_id and preserved both char_span and token_span derived during segmentation, so that auditors can reconstruct the exact context byte-accurately from the source text. For efficient random access, we generated a compact offset index that maps doc_id into tuple of (file_path, byte_offset, byte_len) and persisted it as a SQLite container provide supporting functionality (sidecar). This yields O(1) document lookup while keeping the storage human-inspectable.

The extraction pipeline produced gzipped shards (approx. 1–3 GB each) partitioned by lang/snapshot_id and by the first digits of doc_id. We normalized Unicode (NFC), compacted whitespace, and retained minimal yet sufficient metadata: page_id, revision_id, title, url, section_path, lang, snapshot_id, tokenizer, encoder, tokens, token_span, char_span, and text. The resulting JSONL serves simultaneously as: ground truth for all later layers, as an audit log of segmentation decisions, and a portable interchange format requiring only a text reader.

Schema (abbreviated). {doc_id:int64, page_id:int64, revision_id:int64, title:str, url:str, section_path:list<str>, lang:str, snapshot_id:str, tokenizer:str, encoder:str, tokens:int32, token_span:list<int32>, char_span:list<int32>, text:str}. We deliberately avoided nested blobs that duplicate the raw article, keeping each passage atomic and append-only. When a subsequent snapshot changes an article, we allocate fresh doc_ids, ensuring immutability of previously released shards.

5.2. Building a vector index for RAG from JSON

We constructed the initial retrieval subsystem entirely from the JSONL corpus, omitting any external vector database. Each passage's text field was embedded using basic BERT [23] (default: bert-base-multilingual-cased) with mean pooling over the last hidden layer gated by the attention mask. We then L2-normalized the resulting vectors and stored them as a contiguous matrix (float32, NumPy .npy). At query time, we compute a BERT embedding for the query with the same procedure and perform brute-force cosine search against the matrix. Although asymptotically more expensive than specialized ANN structures, this JSON-only baseline provides deterministic alignment with the tokenizer used in segmentation, high auditability, and a strong sanity-check for later, more complex indices.

For this purpose JSONL shards were streamed in batches, texts were prefixed with no special markers (BERT does not require instruction prefixes), tokenized with the exact tokenizer pinned in Part 2, and encoded on GPU where available. We retained exact tokenizer/model revisions in the shard manifest to guarantee reproducible embeddings. The embedding matrix and a parallel doc_ids.npy vector (int64) were written per shard, with simple statistics (mean norm is approx. 1.0, zero NaNs) logged for each batch.

While specialized retrieval encoders exist, BERT remains a robust baseline for multilingual semantics when paired with an appropriate pooling strategy. Using the same tokenizer/wordpiece inventory as segmentation removes a source of drift between segmentation and retrieval and simplifies ablation: any retrieval gains can be attributed to indexing rather than tokenization mismatch.

For a query string, we apply the same BERT tokenizer and mean-pooling, normalize the vector, and compute cosine similarities against the passage matrix using blocked matrix multiplication to keep CPU caches warm. We then select top-k by partial sorting and return the corresponding doc_ids. Empirically, on a single 16-core server, brute-force cosine over approx. 250k passages remains within interactive latencies (approx. 10–60 ms) due to efficient BLAS kernels. Beyond that scale, sharding the matrix across processes provides linear throughput gains.

In parallel to embeddings, we distilled a small SQLite sidecar keyed by doc_id holding title, url, section_path, lang, and char_span. Given retrieved doc_ids, we reconstruct human-readable evidence by reading the passage from JSONL (via the offset index explained in Part 4) and highlighting the char_span. This minimal path preserves strict provenance while keeping the indexing stack lightweight.

The JSON-only, BERT-based retrieval provides a transparent and reproducible baseline that reaches acceptable Recall@k for definition and entity-centric queries on our snapshot while maintaining deterministic evidence reconstruction. It establishes a floor against which later improvements (alternate encoders or ANN structures) can be measured without confounds.

5.3. HNSW: index construction and optimization for fast nearest-neighbor search

We transitioned from the JSON-only, brute-force cosine baseline to a graph-based approximate nearest-neighbor (ANN) index using Hierarchical Navigable Small Worlds (HNSW)[28]. The goal was to maintain high Recall@k while driving down tail latency for interactive RAG. HNSW organizes vectors into a multi-layer navigable small-world graph; queries descend levels greedily and then perform local best-first search at the ground layer.

We trained no additional model: with BERT-derived, L2-normalized passage embeddings produced before we adopted inner product as equivalent to cosine on unit-norm vectors and constructed HNSW indexes per shard (approx. 1–5 M passages). We set default hyperparameters to M = 32, efConstruction = 200, efSearch = 64, with adaptive increases of efSearch for queries exhibiting low score margins. Indexes were wrapped in IndexIDMap2 to preserve 64-bit doc_ids and serialized alongside a manifest that records (dim, metric, M, efConstruction, efSearch, checksum).

HNSW yields a favorable latency–recall frontier on sentence-level embeddings and supports incremental additions. Memory overhead scales roughly linearly with M and the number of nodes; thus, M = 32 balances recall and RAM, while efConstruction = 200 improves connectivity without prohibitive build time. Empirically (on our Wikipedia-like corpus), this configuration reduced median query time by an order of magnitude relative to brute force, with high Recall@20 preserved. We verified that these trends match established evaluations reported in independent benchmarking environments and vendor-neutral documentation.

We parallelized construction across shards. Each worker (CPU-bound for HNSW) mapped embeddings_bert.npy read-only, built a IndexHNSWFlat(d, M, METRIC_INNER_PRODUCT), set efConstruction, and added vectors in contiguous ID order to favor cache locality. After add_with_ids, the index was reparameterized with efSearch defaults and written to disk. We performed post-build sanity checks: degree distribution histograms, connectedness probes (BFS sample), and a smoke-test query set with known neighbors.

Queries encoded by BERT are normalized and searched with HNSW k-NN. We monitor margin statistics (Δ between top-1 and top-2) and empty-hit rate. If a margin is small or the hit set is nearly duplicate, efSearch is raised (e.g., 64 is changed to 128) for a single retry. We cap k≤50 to bound reranker cost in downstream generation.

Search returns doc_ids, which we join to the SQLite sidecar to recover title, url, section_path, lang, and exact char_span. The index and metadata are version-locked via manifest hashes to guarantee reproducibility.

HNSW is robust to moderate noise and local clustering; however, performance can degrade on purely random vectors. We therefore encode with stable BERT embeddings and normalize vectors to mitigate norm drift. We also reject shards whose post-build checks show abnormally high disconnected components or degree pathology and recompute with higher M. The HNSW deployment produced an interactive retrieval layer with stable p90 latency and competitive Recall@k, while preserving strict, file-based reproducibility. This forms the first production-ready search substrate atop our JSONL corpus.

6. Templates for retrieval-augmented generation

6.1. Template choice

We evaluated several variants of templated with understanding that they are not mutually exclusive.

QA style (Question/Context/Answer) - replaces Instruction with Question. It's natural for factoid QA and pairs well with DPR retrievers[29]. However, it's less expressive for summarization or style-constrained tasks.

ReAct-style RAG interleaves Thought and Act(Search) steps to fetch more passages during inference. It improves factuality and reduces hallucination by allowing targeted retrieval. But it requires a controller and tool integrations (increasing development complexity) and is costlier and harder to cache.

Demonstrate-Search-Predict bootstraps few-shot demonstrations with retrieval in the loop approach is strong for tasks that benefit from pipeline-aware exemplars. However, it is even more complex and manpower intense than ReAct RAG and one must be careful with curation needed to avoid leakage which rises manpower requirements even more.

XML-constrained RAG wraps segments in tags:

```
<task><instruction>{instruction}</instruction>
<context>{...}</context>
<answer></answer></task>
```

It is machine-readable and plays well with validators, also it is helpful for structured output requirements. Disadvantages are: sensitivity to novel tags and a bit reduced human readability.

JSON schema RAG when the model fills a JSON object with fields {"answer":..., "citations":[...]}. It's directly machine-readable and simplifies downstream scoring (same as XML-constrained). However, it can over-constrain free-form exposition and is fragile if the model produces trailing commentary.

We decided to use the Instruction/Context/Answer (I/C/A) for ground generation in retrieved evidence because it is sufficient for the task at hand. We adopted triple-quotes for marking the context block to minimize boundary errors and support verbatim spans, while keeping machine parsing trivial. Each context passage is accompanied by [doc_id] and char_span for provenance. Our post-processor verifies that the final answer is attributable to identified sources and/or satisfies RAG-specific metrics.

Our baseline template for expository instructions (e.g., "Explain the significance of photosynthesis in simple terms."):

System: You are a careful assistant. Use only the Context. If the Context is insufficient, say so.
Instruction:{instruction}
Context:
"""
[{doc_id_1}:{char0_1}-{char1_1}]{passage_1}
[{doc_id_2}:{char0_2}-{char1_2}]{passage_2}
…
"""
Answer:

This format decouples Instruction (task control) from Context (evidence) and exposes explicit provenance tags that our evaluator can match against citations.

6.2. Test implementation

On the technical side we implemented a context packer that assembles the top-k passages returned by HNSW into a bounded-length window. Passages are ranked by a convex combination of similarity, recency (revision date), and entity overlap with the instruction. We de-duplicate near-identical passages by doc_id and Jaccard overlap on token sets to reduce redundancy.

We used Mistral-7B-Instruct [30] as the default generator owing to its latency–quality balance on commodity GPUs. The serving stack treats the generator as stateless: prompts are assembled per request. To test that generated text is supported by provided evidence, we integrated very simple attribution-focused metrics. Each answer is accompanied by a list of predicted citations (doc IDs); our scorer computes faithfulness and context relevance using per token Jakard.

7. Adaptive prompts for QA: template design, instantiation, and ablation

Having established a high-recall nearest-neighbor index over BERT embeddings, we designed a family of adaptive prompt templates that expose the user's question in a normalized slot, optionally elicit reasoning traces for more demanding tasks, and standardize the answer channel to facilitate evaluation. We additionally support a Template insertion control (e.g., "Explain step by step: [Question]") that toggles the latent reasoning mode without enforcing a particular trace length.

7.1. Formats and design trade-offs

Structured prompts reduce ambiguity and provide latent affordances for reasoning strategies (decomposition, plan/execution separation). Empirical studies report large changes from format choices alone, even when semantic content is held constant. Reasoning-eliciting templates

consistently improve success on compositional and arithmetic tasks. Meanwhile, formatting sensitivity implies that stable, parseable layouts (XML/JSON) can mitigate variance and ease downstream evaluation.

We tried several forms to test robustness and accommodate different inference regimes (not templates):

Minimalist (Question - Answer):

#Question:{question}
#Answer: <final answer here>.

Such approach allows for lowest token overhead, strong for simple factoid QA when retriever is accurate. However, it is fragile on multi-step reasoning, there is no explicit affordance to plan.

Self-Ask

#Question:{question}
#Sub-question 1: <question> #Answer 1: <answer>
#Sub-question 2: <question> #Answer 2: <answer>
Final Answer: …

It encourages explicit decomposition and can integrate search actions between sub-questions. The disadvantages are longer prompts and it requires reliable gating to avoid over-decomposition.

Plan-and-Solve header

#Plan
#Step1:…
#Step2:…
#Solve <reasoned answer>

This approach separates planning from execution and reduces missing-step errors. But, it also adds 10–25% token overhead and is sensitive to header labels.

Least-to-Most

#Problem:{question}
#Easier subproblems: [s1,s2,…]
#Solve in order and combine.

Such approach helps with compositional generalization. Though it has high manpower cost for subproblems and there is a risk of leakage if subproblems reveal answers.

Tree-of-Thought controller

#Goal:{question}
#Think in branches of up to B alternatives per step; depth D; evaluate with heuristic H.
#Return only the final answer.

It enables breadth exploration and backtracking, but it has highest inference cost and requires custom controller and stopping rules.

Schema-constrained / XML-tagged

#<q>{question}</q><r/>
#<format>final-only</format>

It has best advantages: it is parseable, it aligns with downstream extraction and plays well with safety filters. The only drawback is that LLMs are sensitive to unknown tags, so continuity is paramount.

In our ablations, minimalist and Plan-and-Solve variants yielded the best latency–accuracy balance under tight budgets, while Self-Ask and Tree-of-Thought offered higher ceilings on multi-hop tasks at increased token cost. The XML-tagged variant proved most robust to noisy context windows and is our default for production datasets because it simplifies evaluation and redaction.

7.2. Template spec and serialization

We formalize an Instance as a 4-tuple (question, reasoning, answer, template_label) with optional reasoning and template_label. Each instance is serialized into two orthogonal views:

- a human-readable "paper" view (for this manuscript), and
- a machine template with explicit delimiters for injection into LLM contexts.

Our default machine template for adaptive prompts:

```
<task>
<question>{question}</question>
<reasoning>{reasoning}</reasoning><!--optional-->
<answer>{answer}</answer>
</task>

<controls>
<template>{template_label}</template> <!--e.g., Explain step by step: [{question}]-->
</controls>
```

The XML-like markers act as sentinel delimiters to reduce boundary errors during concatenation and to ease post-hoc parsing. We preserve the underlying JSONL record as ground truth by mirroring these fields as keys {"q":"…","r":"…","a":"…","template":"…"}.

Prompt-format sensitivity is substantial, so structure and delimiters are not cosmetic. Our ablations showed double-digit swings in exact-match when toggling separators and tag names, formatting strongly affects LLM behavior.

For each question we normalized the user questions by lowercasing only function words and preserving named entities. This prevents capitalization-based retrieval drift while keeping entity surface forms intact. Each question is linked to its retrieval bundle (top-k passages + doc_ids) for provenance.

For easy questions (factual, single-hop), we leave <reasoning> field empty and rely on evidence citations during generation. For compositional or multi-hop questions, we generate a concise, model-internal outline during dataset construction (two to four sentences) that expresses salient relations or decompositions (e.g., "Rayleigh scattering; sky perceived blue due to short-wavelength scattering"). We use the <template_label> to switch on an instruction phrase (e.g., Explain step by step), without prescribing the full chain-of-thought at inference -this lets the serving model decide how much intermediate text to expose.

We keep <answer> empty at prompting time (for test splits) or fill it for training instances used in instruction tuning. In both cases the template reserves the slot and the post-processor knows where to harvest the model's output.

This kind of templates naturally support hidden reasoning or outline-level hints but allows to evaluate only the final answer against evidence.

By consolidating on the XML-tagged adaptive template with a "step-by-step" control, we achieved consistent improvements on multi-hop questions without exposing long reasoning traces, lowered annotation overhead than with Self-Ask and Tree-of-Thought, and created robust parsing in our evaluation harness. The template also integrates seamlessly with our RAG context, as tags cleanly delimit query, evidence, and expected outputs.

8. Validation and filtering of examples

After all is done the remaining bottleneck for dataset quality is example hygiene: overly general queries must be removed, near-duplicate instructions must be collapsed, and examples whose answers are not attributable to the retrieved context must be rejected.

We adopt a three-gate policy executed in the following order: (G1) query specificity, (G2) instruction de-duplication, (G3) context-to-answer consistency. The ordering is motivated by cost: pre-retrieval processing is (G1) cheapest, near-duplicate detection (G2) amortizes costs over entire shards, and faithfulness checks (G3) is the hardest. All gates leave a machine-readable audit trail (JSON) that explains the decision with scalar scores and thresholds.

All gates produce structured audit records that log features, scores, thresholds, and the reason for accept/reject. This allows re-running the validator across snapshots with different hyperparameters without re-encoding texts. We deliberately bias toward precision (filtering too much) in G1 and G2, because failures escalate to a rewrite/curation queue; in G3 we balance precision/recall so as not to discard legitimate paraphrastic answers.

Of course there are some issues. Entity-free queries that are still specific in closed domains (e.g., mathematics) may be over-filtered. Lexically diverse paraphrases may evade MinHash. Embedding proximity misses subtle contradictions. Nevertheless automation greatly reduces amount of required manual filtering.

8.1. Gate 1 - Overly general queries

The objective is to identify and remove prompts whose information need is under-specified (e.g., "Tell me about science"), which systematically depress retrieval and confound downstream evaluation. We formalize under-specification as low lexical specificity.

We combine pre-retrieval predictors from [31] with lightweight lexical features:

- avgIDF and sumIDF over content tokens computed from our passage collection, low values indicate generic terms.
- Stopword ratio and entity density (named entities per token), to penalize boilerplate phrasings while retaining entity-anchored requests.
- Length controls (minimum 3 content tokens after stopword removal) to rule out one-term commands.

We compute these signals per query and apply a learned decision rule trained on a small, hand-labeled dev set. The classifier is a calibrated logistic regression over standardized features (avgIDF, stopword ratio, lexical specificity, content length), chosen for transparency.

Applying G1 removes a consistent tail of vague prompts while preserving entity-centric and compositional questions. This aligns with reports that pre-retrieval specificity predictors are strong proxies for query quality and reduces the frequency of ungroundable generations in our RAG evaluation.

8.2. Gate 2 - Duplicate instruction detection

The objective is to collapse near-duplicate Instruction and Question variants to avoid training and test leakage, evaluation bias, and over-representation of popular intents.

We combine lexical features MinHash to capture surface-form duplicates and semantic cosine with the same BERT mean-pooled embeddings used elsewhere. So, each normalized instruction is shingled (word 5-grams), hashed into a MinHash signature, and indexed. The BERT embedding matrix of instructions is searched with HNSW to collect cosine neighbors. We merge close items into clusters. This hybrid approach catches both trivial paraphrases and deep semantic duplicates.

It directly follows evidence that deduplication improves generalization and lowers memorization in LLMs, and mirrors web-scale dataset construction practices that rely on MinHash at scale.

8.3. Gate 3 - Context-answer consistency

The objective is to detect and reject examples whose Answer is not supported by the Context returned by the retriever, thereby enforcing attribution and reducing hallucinations.

We combine a vector-proximity test with a textual alignment score:

- Attribution compliance. If the template requires citations, we verify that cited doc_ids are a subset of the retrieved doc_ids. This is formal procedure.
- Embedding proximity test. We split the answer into sentences, encode each with BERT mean-pooling, and compute its maximum cosine similarity to any sentence from the packed context. An example passes if sufficient number of scores of answer sentences exceed a threshold. This test is indexing-agnostic and efficiently batched.
- For curated subsets we compute a factual consistency score with an alignment model between the context and the answer. Low alignment flags unsupported claims even when embeddings yield false positives.

We first run the proximity test, failing items are dropped immediately. Passing items are passed to the alignment scorer. The two-stage test rejects obviously unsupported answers cheaply and reserves expensive alignment modeling for ambiguous cases. In practice this reduces hallucination rate and raises attribution scores in our RAG evaluations without inflating latency during dataset construction.

9. Discussion and future work

While BERT-compatible WordPiece tokenization is our default for interoperability with BEIR-style retrievers, we considered several alternatives. Each of them has strong advantages and they are listed below, but disadvantages made them untenable at this stage of development.

XLM-R [24] is a SentencePiece Unigram miodel. Its advantages are: strong cross-lingual performance, robust to morphology, trained on large CommonCrawl in 100 languages. The disadvantages consist of: 512-token limit, unigram segmentation differs from WordPiece, leading to different window statistics, larger models incur higher latency.

mDeBERTaV3 [25] is a multilingual DeBERTa. Its advantages are: improved pretraining objective (RTD) and parameter sharing, strong multilingual transfer, available HF checkpoints. The disadvantages consist of: tokenizer is SentencePiece-like, sequence limits and memory use comparable to BERT-base, fewer production-grade embedding heads than E5/BGE.

MPNet[32]/RoBERTa-family work as byte-level BPE. Its advantages are: byte-level resilience, strong sentence-embedding variants (e.g., all-mpnet-base-v2), larger maximum input length and fast tokenization. The disadvantages consist of: mismatch with WordPiece passage stats, byte-level normalization can change token counts for Cyrillic punctuation.

Advanced embedding models such as E5-v2[33] and BGE-M3 [27]. Their advantages are: state-of-the-art retrieval transfer, multilingual coverage, explicit instructions for query/passage encoding, often tolerant to moderately longer chunks. The disadvantages consist of: require careful prompt formats (e.g., query:/passage: prefixes) and may expect different maximum lengths (e.g., 512–1024 tokens) and pooling strategies.

As for performance, there is another option for memory-sensitive applications. IVF+PQ [34] is a technique that combines two indexing methods for efficient search in high-dimensional vector data: Inverted File (IVF), which structures data based on similarity, and Product Quantization (PQ), which compresses vectors by dividing them into subvectors and quantizing them. This allows for

significantly faster search and reduced memory requirements for storing large data sets. It excels in memory efficiency while HNSW excels in recall–latency balance on CPU so it can be done.

10. Conclusions

We successfully designed and implement a reproducible Ukrainian language centered Wikipedia-to-Tasks pipeline that can do the following:

- robustly extracts and cleans content from MediaWiki snapshots;
- performs adaptive chunking that respects section/paragraph boundaries and augments queries with retrieval context;
- builds lexical, dense, and hybrid indices with explicit provenance;
- generates and normalizes instruction and QA pairs supported by retrieval evidence; and
- filters low-quality or duplicate items (including context–answer mismatches by embedding similarity) with targeted human review.

Acknowledgements

We are very grateful for support we received during this research from Department 165 of Institute of Information Technologies and Systems and Department of Mathematical Informatics from Faculty of Computer Sciences and Cybernetics of Taras Shevchenko National University of Kyiv. The research was funded by the National Research Foundation of Ukraine.

Declaration on Generative AI

During the preparation of this work, the authors used ChatGPT-5 in order to: Text Translation, Grammar and spelling check. After using this tool, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

References

[1] Chen, M.; Wiseman, S.; Gimpel, K. WikiTableT: A Large-Scale Data-to-Text Dataset for Generating Wikipedia Article Sections. In: Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021. Association for Computational Linguistics, 2021, pp. 193–209. doi:10.18653/v1/2021.findings-acl.17. URL: https://aclanthology.org/2021.findings-acl.17/.

[2] Parikh, A.; Wang, X.; Gehrmann, S.; Faruqui, M.; Dhingra, B.; Yang, D.; Das, D. ToTTo: A Controlled Table-to-Text Generation Dataset. In: Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP 2020). Association for Computational Linguistics, 2020, pp. 1173–1186. doi:10.18653/v1/2020.emnlp-main.89. URL: https://aclanthology.org/2020.emnlp-main.89/.

[3] Petroni, F.; Piktus, A.; Fan, A.; Lewis, P.; Yazdani, M.; De Cao, N.; Thorne, J.; Jernite, Y.; Karpukhin, V.; Maillard, J.; Plachouras, V.; Rocktäschel, T.; Riedel, S. KILT: a Benchmark for Knowledge-Intensive Language Tasks. In: Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT 2021). Association for Computational Linguistics, 2021, pp. 2523–2544. doi:10.18653/v1/2021.naacl-main.200. URL: https://aclanthology.org/2021.naacl-main.200/.

[4] Lewis, P.; Wu, Y.; Liu, L.; Minervini, P.; Küttler, H.; Piktus, A.; Stenetorp, P.; Riedel, S. PAQ: 65 Million Probably-Asked Questions and What You Can Do With Them. Transactions of the Association for Computational Linguistics, 9, 2021, pp. 1098–1115. doi:10.1162/tacl_a_00415. URL: https://aclanthology.org/2021.tacl-1.65/.

[5] Thakur, N., Reimers, N., Rücklé, A., Srivastava, A., Gurevych, I. (2021). BEIR: A Heterogeneous Benchmark for Zero-shot Evaluation of Information Retrieval Models. NeurIPS 2021 Datasets and Benchmarks. URL: https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/file/65b9eea6e1cc6bb9f0cd2a47751a186f-Paper-round2.pdf.
[6] Zhang, X., Thakur, N., Ogundepo, O., Kamalloo, E., Alfonso-Hermelo, D., Li, X., Liu, Q., Rezagholizadeh, M., Lin, J. (2023). MIRACL: A Multilingual Retrieval Dataset Covering 18 Diverse Languages. TACL, 11, 1114–1131. doi:10.1162/tacl_a_00595.
[7] Trivedi, H., Khot, T., Gupta, N., Sabharwal, A., Clark, P. (2022). MuSiQue: Multihop Questions via Single-hop Question Composition. TACL, 10, 539–554. URL: https://aclanthology.org/2022.tacl-1.31.pdf.
[8] Gao, Y., Zhang, N. L., Chen, D., Lin, J., Zhang, Y., Berant, J. (2021). Answering Ambiguous Questions through Generative Evidence Fusion and Disambiguation. In Proceedings of ACL 2021 (Long). URL: https://aclanthology.org/2021.acl-long.253.pdf.
[9] Aly, R., et al. (2021). FEVEROUS: Fact Extraction and VERification Over Unstructured and Structured information. NeurIPS 2021 Datasets and Benchmarks. URL: https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/file/68d30a9594728bc39aa24be94b319d21-Paper-round1.pdf.
[10] Glockner, M., et al. (2024). AmbiFC: Fact-Checking Ambiguous Claims with Evidence. TACL, 12, 1792–1811. doi:10.1162/tacl_a_00629.
[11] Marchenko, O.; Anisimov, A.; Nykonenko, A.; Rossada, T.; Melnikov, E. Authorship Attribution System. In: Frasincar, F.; Ittoo, A.; Nguyen, L.; Métais, E. (eds) Natural Language Processing and Information Systems. NLDB 2017. Lecture Notes in Computer Science, vol. 10260. Springer, Cham, 2017, pp. 227–231. doi:10.1007/978-3-319-59569-6_27.
[12] Anisimov, A. V.; Marchenko, O. O.; Vozniuk, T. G. Determining Semantic Valences of Ontology Concepts by Means of Nonnegative Factorization of Tensors of Large Text Corpora. Cybernetics and Systems Analysis, 50, 327–337, 2014. doi:10.1007/s10559-014-9621-9.
[13] Taranukha, V.; Horokhova, T.; Linder, Y. On Semi-Automatic Creation of Dataset for Multi-Document Automatic Summarization of News Articles and Forum Threads. In: Selected Papers of the VIII International Scientific Conference "Information Technology and Implementation" (IT&I-2021), Workshop Proceedings (CEUR Workshop Proceedings, vol. 3179). CEUR-WS.org, 2022, pp. 15–24.
[14] Longpre, S., Lu, Y., Daiber, J. (2021). MKQA: A Linguistically Diverse Benchmark for Multilingual Open-Domain Question Answering. TACL, 9, 1389–1406. doi:10.1162/tacl_a_00433.
[15] Boroş, T., et al. (2024). Fine-Tuning and Retrieval Augmented Generation for the UNLP 2024 Task on Ukrainian Question Answering. In Proceedings of UNLP @ LREC-COLING 2024. URL: https://aclanthology.org/2024.unlp-1.10.pdf.
[16] Hamotskyi, S., et al. (2024). Eval-UA-tion 1.0: Benchmark for Evaluating Ukrainian Language Models. In Proceedings of UNLP 2024. URL: https://aclanthology.org/2024.unlp-1.13.pdf.
[17] Hasan, T., Bhattacharjee, A., Islam, M. S., Mubasshir, K., Li, Y.-F., Kang, Y.-B., Rahman, M. S., Shahriyar, R. (2021). XL-Sum: Large-Scale Multilingual Abstractive Summarization for 44 Languages. Findings of ACL-IJCNLP 2021. URL: https://aclanthology.org/2021.findings-acl.413/.
[18] Chaplynskyi, D., Romanyshyn, M. (2024). Introducing NER-UK 2.0: A Rich Corpus of Named Entities for Ukrainian. In Proceedings of UNLP 2024. URL: https://aclanthology.org/2024.unlp-1.4.pdf.
[19] Barbosa, D., Ustyianovych, T. (2024). TRWU: Instant Messaging Platforms News Multi-Task Classification for Stance, Sentiment, and Discrimination Detection. In Proceedings of UNLP 2024. URL: https://aclanthology.org/2024.unlp-1.5.pdf.
[20] Attardi, G. WikiExtractor (project site). GitHub Pages. 2012–2025. URL: https://attardi.github.io/wikiextractor/.
[21] Kurtovic, B. MWParserFromHell v0.7: Documentation. Read the Docs. URL: https://mwparserfromhell.readthedocs.io/.

[22] JSON Lines. JSON Lines: A Text Format for Storing Structured Data. 2014–2025. URL: https://jsonlines.org/
[23] Devlin, J.; Chang, M.-W.; Lee, K.; Toutanova, K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT 2019), Volume 1 (Long and Short Papers). Association for Computational Linguistics, 2019, pp. 4171–4186. doi:10.18653/v1/N19-1423. URL: https://aclanthology.org/N19-1423/.
[24] Conneau, A.; Khandelwal, K.; Goyal, N.; Chaudhary, V.; Wenzek, G.; Guzmán, F.; Grave, E.; Ott, M.; Zettlemoyer, L.; Stoyanov, V. Unsupervised Cross-lingual Representation Learning at Scale. In: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL 2020). Association for Computational Linguistics, 2020, pp. 8440–8451. doi:10.18653/v1/2020.acl-main.747. URL: https://aclanthology.org/2020.acl-main.747/. Also arXiv:1911.02116.
[25] He, P.; Gao, J.; Chen, W. DeBERTaV3: Improving DeBERTa using ELECTRA-Style Pre-Training with Gradient-Disentangled Embedding Sharing. In: The Eleventh International Conference on Learning Representations (ICLR 2023), Kigali, Rwanda, May 1–5, 2023. OpenReview.net, 2023. URL: https://openreview.net/forum?id=sE7-XhLxHA
[26] Wang, L.; Yang, N.; Huang, X.; Yang, L.; Majumder, R.; Wei, F. Multilingual E5 Text Embeddings: A Technical Report. arXiv:2402.05672, 2024. doi:10.48550/arXiv.2402.05672
[27] Chen, J.; Xiao, S.; Zhang, P.; Luo, K.; Lian, D.; Liu, Z. BGE M3-Embedding: Multi-Lingual, Multi-Functionality, Multi-Granularity Text Embeddings Through Self-Knowledge Distillation. arXiv:2402.03216, 2024. doi:10.48550/arXiv.2402.03216
[28] Malkov, Y. A.; Yashunin, D. A. Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. IEEE Transactions on Pattern Analysis and Machine Intelligence, 42(4), 824–836, 2020. doi:10.1109/TPAMI.2018.2889473.
[29] Karpukhin, V.; Oğuz, B.; Min, S.; Lewis, P.; Wu, L.; Edunov, S.; Chen, D.; Yih, W.-t. Dense Passage Retrieval for Open-Domain Question Answering. In: Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP 2020). Association for Computational Linguistics, 2020, pp. 6769–6781. doi:10.18653/v1/2020.emnlp-main.550. URL: https://aclanthology.org/2020.emnlp-main.550/.
[30] Jiang, A. Q.; Sablayrolles, A.; Mensch, A.; Bamford, C.; Chaplot, D. S.; de las Casas, D.; Bressand, F.; Lengyel, G.; Lample, G.; Saulnier, L.; Lavaud, L. R.; Lachaux, M.-A.; Stock, P.; Le Scao, T.; Lavril, T.; Wang, T.; Lacroix, T.; El Sayed, W. Mistral 7B. arXiv:2310.06825, 2023. doi:10.48550/arXiv.2310.06825. URL: https://arxiv.org/abs/2310.06825
[31] Zendel, O.; et al. QPPTK@TIREx: Simplified Query Performance Prediction Toolkit. In: Proceedings of TIREx@CLEF 2024, CEUR-WS, 2024. URL: https://ceur-ws.org/Vol-3689/WOWS_2024_paper_6.pdf
[32] Song, K.; Tan, X.; Qin, T.; Lu, J.; Liu, T.-Y. MPNet: Masked and Permuted Pre-training for Language Understanding. In: Advances in Neural Information Processing Systems 33 (NeurIPS 2020). 2020. URL: https://proceedings.neurips.cc/paper/2020/file/c3a690be93aa602ee2dc0ccab5b7b67e-Paper.pdf. Also arXiv:2004.09297.
[33] Wang, L.; Yang, N.; Huang, X.; Yang, L.; Majumder, R.; Wei, F. Improving Text Embeddings with Large Language Models. In: Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). Association for Computational Linguistics, 2024, pp. 11897–11916. doi:10.18653/v1/2024.acl-long.642. URL: https://aclanthology.org/2024.acl-long.642/
[34] Facebook Research. The index_factory. FAISS Wiki (GitHub), updated 2025. URL: https://github.com/facebookresearch/faiss/wiki/The-index-factory (accessed 20 Sep 2025).