Ablation study of correction methods in Polish language Retrieval-Augmented Generation chatbots

S. FLOREK
ORCID: 0009-0000-5568-6984
szymon.florek@ardium.pl

Ardium sp. z o.o.
Ceramików Str. 21a, 44-151 Gliwice, Poland

P. GÓRNY
ORCID: 0000-0003-3181-6320
piotr.gorny@wat.edu.pl

Military University of Technology, Faculty of Cybernetics
Kaliskiego Str. 2, 00-908 Warsaw, Poland

Retrieval-Augmented Generation (RAG) mitigates hallucinations in large language models by grounding responses in retrieved evidence, yet the interplay of correction techniques across the RAG pipeline remains underexplored – especially for non-English languages. We present a modular RAG system evaluated through a systematic ablation study of **12 pipeline configurations** on a **1,000-question Polish-language test set** derived from a 1M-article Wikipedia corpus and manually validated. Four correction families are compared: (i) cross--encoder reranking, (ii) query decomposition with multi-hop reranking and diversity enforcement, (iii) native Chain-of-Thought (CoT) reasoning, and (iv) post-generative Chain-of-Verification (CoVe). Results evaluated with RAGAS metrics show that **multi-hop reranking yields the largest quality gains** (Δ Faithfulness +15.6%, Δ Context Recall +38.4%, Δ Relevancy +20.3%) at acceptable latency (18.4 s), while CoVe in auto-correction mode can paradoxically *degrade* faithfulness at high computational cost (+41.6 s). Cross-encoder reranking achieves the best cost quality ratio (+16.7% Recall, +1.2 s). We discuss practical configuration tiers for real--time, interactive, and offline deployments.

Keywords: Retrieval-Augmented Generation, ablation study, multi-hop reranking, Chain-of-Thought, Chain-of-Verification, Polish NLP, chatbot evaluation.

DOI: 10.5604/01.3001.0055.9903

1. Introduction

Large language model (LLM)-based chatbots achieve unprecedented conversational fluency, yet their deployment in production environments is constrained by three fundamental problems: factual hallucinations, knowledge staleness beyond the training cutoff, and inability to provide verifiable source attribution [1, 2]. Recent work on detecting such hallucinations via semantic entropy [11] highlights the severity of the problem. Empirical studies confirm that hallucinations directly erode user trust – Larsen et al. [16] show that exposure to confidently stated falsehoods is the primary driver of user dissatisfaction, while Massenon et al. [17] report that users actively flag hallucinations as the main reason for abandoning LLM-powered mobile applications.

Retrieval-Augmented Generation (RAG) introduced by Lewis et al. [1], addresses these limitations by combining parametric knowledge encoded in model weights with non-parametric access to an external knowledge base at inference time. The architecture enables knowledge updates without retraining, reduces hallucinations through evidence grounding, and provides source provenance for generated claims. However, naïve RAG implementations – single-pass retrieval followed by direct context injection – still suffer from noisy retrieval, poor context precision, and inadequate handling of complex multi-hop queries that require synthesizing information from multiple sources [4, 15].

A growing body of work proposes individual improvements at different pipeline stages: cross-encoder reranking for retrieval precision [5], Chain-of-Thought (CoT) prompting for structured reasoning [6], query decomposition for closing the compositionality gap in multi-hop questions [7], and Chain-of-Verification (CoVe) for post-generative hallucination detection [8]. Yet few studies *systematically compare* these techniques within a single controlled pipeline to isolate their individual and combined effects on end-to-end quality. Furthermore, virtually all existing evaluations target English-language corpora, leaving the effectiveness of these methods on morphologically rich languages such as Polish unexplored.

Contributions. We make the following contributions:

1. A modular RAG architecture implemented in FastAPI with plug-and-play activation of correction modules at four pipeline stages: pre-retrieval (query analysis and decomposition), retrieval (cross-encoder reranking with diversity enforcement), generation (enhanced prompting with native CoT), and post-generation (claim-level verification and correction).
2. A 12-configuration ablation study on 1,000 synthetic Polish-language questions grounded in a 1M-article Wikipedia corpus, evaluated with RAGAS metrics (Faithfulness, Answer Relevancy, Context Precision, Context Recall).
3. An empirical demonstration that retrieval-stage interventions – specifically multihop reranking with query decomposition dominate generation-stage and post--generation-stage corrections in end-to-end quality across all measured metrics.
4. Practical latency – quality deployment tiers ranging from real-time (<5 s) to offline batch processing, enabling practitioners to select configurations matched to their use case.

2. Related Work

RAG Fundamentals

Lewis et al. [1] formalize RAG as a marginalization over retrieved documents:

$$P_{RAG}(y \mid x) \approx \Sigma_{z \in Z_K(x)} P_\eta(z \mid x) \cdot P_\theta(y \mid x, z) \quad (1)$$

where:
x – is the user query,
y-the – generated answer,
z – a context passage from the knowledge base,
$Z_K(x)$ – the top-K retrieved documents,
$P_\eta(z \mid x)$ – the retriever parameterized by η, and
$P_\theta(y \mid x, z)$ – the generator parameterized by θ.
The authors distinguish two variants: RAG-Sequence (one document conditions the entire output) and RAG-Token (different documents may condition different tokens), with the former dominating practical deployments due to lower computational cost.

In engineering terms, the RAG pipeline consists of an offline phase (document segmentation into chunks, embedding computation, and approximate nearest neighbor indexing via algorithms such as HNSW [19]) and an online phase (dense retrieval via bi-encoder similarity, context augmentation, and grounded generation) [4]. The bi-encoder architecture, formalized in Dense Passage Retrieval (DPR) [9], maps queries and documents independently into a shared vector space, enabling efficient offline indexing. However, bi-encoders lack explicit token-level interaction modeling, motivating a second-stage cross-encoder reranker that jointly encodes the (query, document) pair for more precise relevance scoring [5].

For evaluation, the RAGAS framework [10] decomposes end-to-end quality into retrieval metrics (Context Precision, Context Recall) and generation metrics (Faithfulness – the proportion of claims supported by retrieved context, and Answer Relevancy – the degree to which the response addresses the query). This layered evaluation is essential because retrieval and generation failures are independent: a perfect generator cannot compensate for missing evidence, and accurate retrieval does not guarantee faithful synthesis [4].

Correction Techniques for RAG

Prompt engineering and grounding rules. In RAG systems, the generator prompt typically includes explicit grounding rules instructing the model to answer exclusively from provided sources, cite evidence for factual claims, and signal uncertainty when context is insufficient. Ouyang et al. [22] demonstrate that instruction--tuned models are highly sensitive to prompt structure, making prompt design an integral and inseparable component of each pipeline module.

Chain-of-Thought (CoT). Wei et al. [6] show that prompting LLMs to produce intermediate reasoning steps before the final answer significantly improves performance on complex tasks. In RAG, CoT forces the model to explicitly analyze retrieved sources before synthesis, which reduces hallucination. Native reasoning modes in newer models (e.g., Qwen3's

<*think*> tags) enable CoT without manual few-shot exemplars [23].

Query decomposition and multihop reasoning. Complex queries requiring information from multiple documents expose a *compositionality gap* – the model can answer sub-questions individually but fails on their composition [7]. Decomposing such queries into sub-questions, retrieving evidence for each, and fusing results addresses this gap. Yang et al. [21] formalize multi-hop QA evaluation with the HotpotQA benchmark.

Chain-of-Verification (CoVe). Dhuliawala et al. [8] propose a post-generative verification pipeline: (1) extract individual claims from the generated response, (2) formulate verification questions for each claim, (3) check claims against retrieved evidence using natural language inference, and (4) correct or remove unsupported claims. The approach targets hallucinations that survive retrieval-grounded generation.

Research gap. While each technique has been evaluated in isolation, no prior work provides a controlled ablation comparing all four families within a unified pipeline. Furthermore, existing multihop reranking approaches typically treat sub-query results uniformly; to our knowledge, no prior work introduces query-type--aware weighted fusion with diversity enforcement as proposed in our Algorithm 2. Moreover, existing evaluations focus predominantly on English, leaving morphologically complex languages such as Polish – with rich inflection, free word order, and limited NLP tooling – entirely unaddressed.

3. System Architecture

Layered Overview

Figure 1 presents the system architecture. The **presentation layer** exposes a REST API via FastAPI with streaming support for real-time response delivery. The **orchestration layer** implements a strategy pattern that routes queries through the appropriate pipeline variant (baseline vs. enhanced) based on runtime configuration flags, enabling the ablation study described in Section 4.3. The **pre-retrieval layer** performs linguistic analysis and, when multihop mode is active, decomposes complex queries into focused sub-queries (Algorithm 1). The **retrieval layer** executes dense vector search via Qdrant followed by optional cross-encoder reranking and diversity enforcement (Algorithm 2). The **generation layer** constructs grounded prompts from YAML templates and invokes the LLM with optional native CoT reasoning. The **post-generation layer** implements the CoVe verification pipeline (Algorithm 3).

Each correction module can be independently toggled, and key architectural decisions include dependency injection with LRU caching for model singletons (avoiding repeated model loading), YAML-based prompt templates for reproducibility, and Qdrant with retry logic as the vector store.

Correction Modules

Pre-retrieval: Adaptive Query Rewriting. The AdaptiveQueryRewriter module (Algorithm 1) implements a two-phase process (Figure 2): linguistic feature extraction followed by LLM--driven classification and optional decomposition. In the first phase, spaCy [13] (*pl_core_news_md*) analyzes the query to extract a feature vector comprising: named entity count, dependency tree depth, presence of coordinating conjunctions (*i*, *oraz*, *a*), temporal markers (*kiedy*, *przed*, *po*), and comparative operators (*większy*, *niż*, *porównaj*). These linguistic features are formatted into a structured analysis report and passed to the LLM via a classification prompt.

In the second phase, the LLM evaluates the feature report and classifies the query as *simple* or *multihop*. Simple queries proceed directly to retrieval unchanged. Multihop queries trigger decomposition: the LLM generates 2–4 focused sub-queries, each targeting a specific information need. The decomposition prompt enforces strict rules: (1) each sub-query must be self-contained and independently retrievable, (2) entity names must be preserved in their nominative form without morphological modification – critical for Polish, where declension alters surface forms (e.g., *Warszawa* → *Warszawy*) and degrades bi-encoder retrieval recall, (3) sub-queries must collectively cover all information needs of the original query, and (4) each sub-query is tagged with a type label used for fusion weight (α) assignment in Algorithm 2 (Table 1).

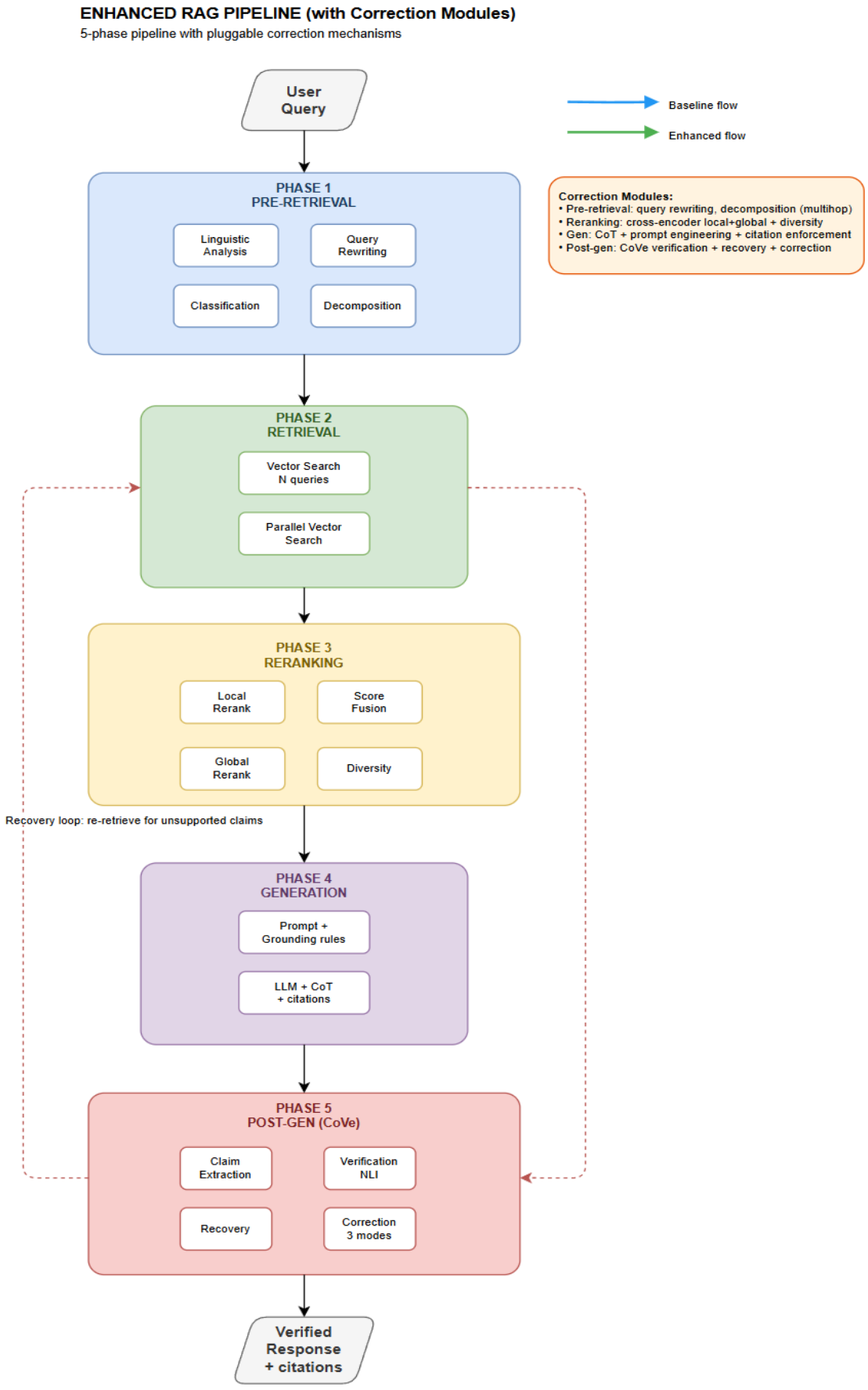


Fig. 1. System architecture. Correction modules (shaded) can be independently toggled per configuration. Arrows indicate data flow from user query (top) through pre-retrieval analysis, dense retrieval with reranking, grounded generation, and optional post-generative verification to the final response (bottom)

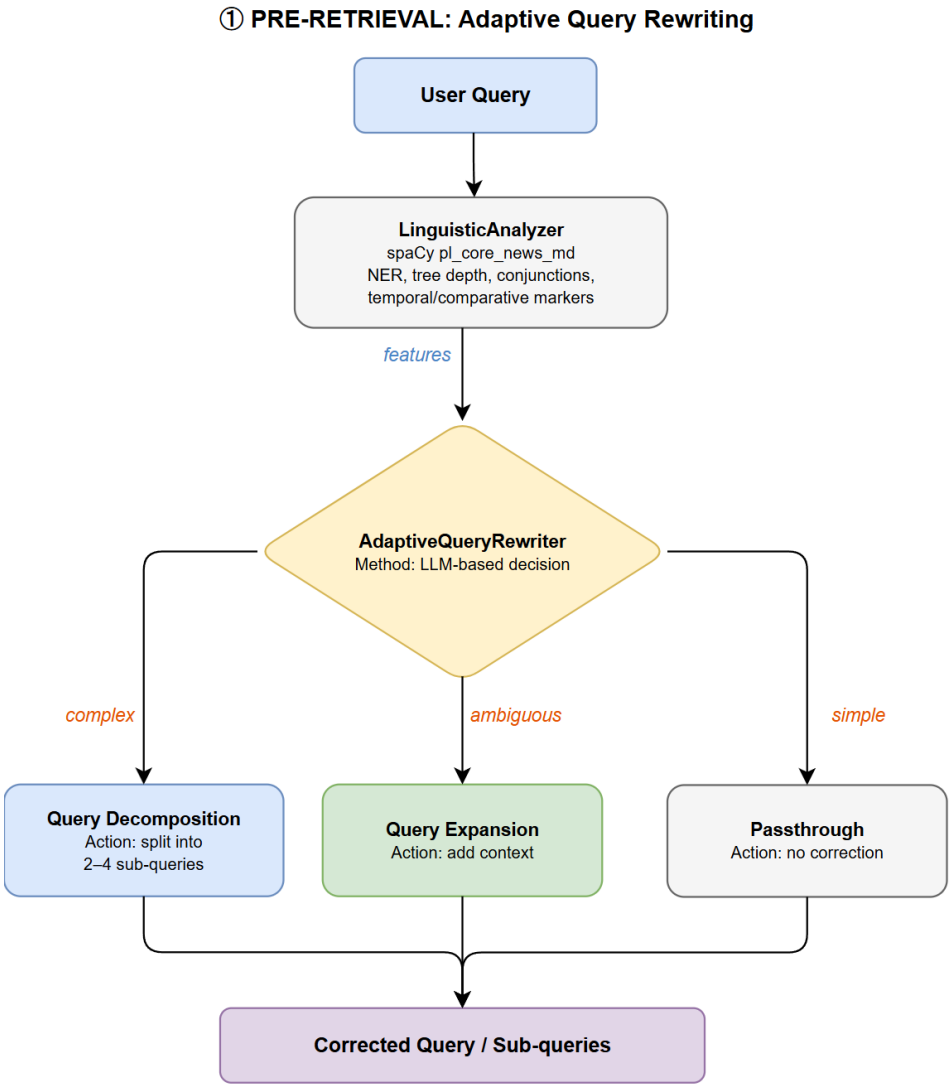


Fig. 2. Adaptive query rewriting pipeline. The spaCy linguistic analyzer extracts features (NER count, tree depth, conjunction/temporal/comparative markers) from the user query. The feature vector is passed to the LLM classifier which decides between direct retrieval (simple) and decomposition into 2–4 typed sub-queries (multihop). Entity names are preserved in nominative form to maintain retrieval recall in Polish

Algorithm 1: Adaptive Query Classification and Decomposition

```
Require: User query x, spaCy model S, LLM M
Ensure: Query type t ∈ {simple, multihop}, sub-queries Q
Phase 1 — Linguistic feature extraction:
    doc ← S(x)
    f_ner ← |{e ∈ doc.ents}|
    f_depth ← max_tree_depth(doc)
    f_conj ← has_conjunction(doc)
    f_temp ← has_temporal_marker(doc)
    f_comp ← has_comparative_op(doc)
    f ← [f_ner, f_depth, f_conj, f_temp, f_comp]
Phase 2 — LLM classification and decomposition:
    t ← M(classify_prompt(x, f))
    if t = simple then return t, {x}
    else
        Q ← M(decompose_prompt(x, f))
        for each q_i ∈ Q do
            Validate entity preservation; Assign type label
        end for
        return t, Q
    end if
```

Retrieval: Cross-Encoder Reranking and Multihop Fusion

For simple queries, retrieved candidates are reranked using a cross-encoder (jina-reranker-v2--base-multilingual [14]) that jointly encodes the (query, passage) pair and returns a fine-grained relevance score. For multihop queries, Algorithm 2 implements a three-stage reranking process with diversity enforcement. Table 1 lists the query-type-specific fusion weights α_t. The retrieval budgets are set to $N = 200$ bi-encoder candidates per sub-query, $k_i = 10$

after per-subquery reranking, and K = 15 final context chunks.

Algorithm 2: Multihop Reranking with Diversity Enforcement

```
Require: Original query x, sub-queries {q_1, …, q_m}, budget N, final budget K, fusion weights α_t
Stage 1 — Per-subquery retrieval and reranking:
    for each sub-query q_i do
        Retrieve top-N candidates from Qdrant via bi-encoder
        Rerank candidates with cross-encoder, keep top-k_i
    end for
Stage 2 — Global reranking with original query:
    Pool all per-subquery results into candidate set C
    Rerank C against original query x with cross-encoder
    Compute final score: s_j = α_t · s_cross(x, c_j) + (1-α_t) · max_i s_cross(q_i, c_j)
Stage 3 — Diversity enforcement:
    Sort candidates by final score; R ← ∅
    for each sub-query q_i not yet represented in R do
        Add highest-scoring candidate from q_i to R
    end for
    Fill remaining K − |R| slots with top-scored unselected candidates
    return Top-K diverse context set R
```

Tab. 1. Query-type-specific fusion weight α_t for global reranking. Higher α_t favors the original query (global coherence); lower α_t favors sub-query scores (local independence)

Sub-query type	α_t	Rationale
Verification	0.20	Sub-queries collect facts independently
Comparison	0.25	Each entity analyzed separately
Similarity	0.25	Entity features are independent
Aggregation	0.50	Balance: local collection, global aggregation
Chaining	0.60	Answer must fit the full reasoning chain
Temporal	0.60	Temporal sequence requires global coherence
Superlative	0.65	Global perspective needed (best/worst)
Simple	0.70	Fallback – should not be multihop

The diversity enforcement step (lines 12–15) ensures that the final context contains at least one document from each sub-query, preventing any single information need from dominating the context window. This directly addresses the "Lost in the Middle" phenomenon [18], where relevant passages buried deep in the context are ignored by the generator.

Generation: Enhanced Prompting with Native CoT. The generation module constructs prompts from YAML templates with three levels of sophistication. The baseline template provides a minimal instruction. The enhanced template adds explicit grounding rules ("answer ONLY based on provided sources"), anti-hallucination instructions, and citation formatting requirements using source markers [N]. For multihop queries, a dedicated multihop template includes sections for decomposition context, grouped source display per sub-query, fusion rules, and a coverage gate verifying that all sub-questions are addressed. When CoT is enabled, the model's native reasoning mode is activated (Qwen3's /think mode), producing structured analysis within <think>…</think> tags before generating the user-facing response.

Post-generation: Chain-of-Verification (CoVe). The CoVe module implements a three-phase post-generative verification pipeline (Algorithm 3) inspired by Dhuliawala et al. [8], adapted for Polish-language processing and integrated with the retrieved context.

Phase 1 – Claim Extraction.

The ClaimExtractor receives the generated answer along with the original context passages and decomposes it into atomic factual claims. Each claim is output as a structured JSON object containing (a) the claim text, (b) a list of source chunk identifiers that the generator referenced, and (c) the sentence index within the original answer. The extraction prompt is designed to preserve the original phrasing of the answer and to split compound sentences into individual verifiable propositions. For Polish-language answers, the prompt additionally instructs the model to normalize inflected entity names back to nominative form for downstream matching. The number of extracted claims typically ranges from 3 to 12 per answer, depending on query complexity.

Phase 2 – Claim Verification.

The ClaimVerifier performs batch natural language inference (NLI) by pairing each extracted claim with the retrieved context passages. For each claim, the verifier constructs a premise-hypothesis pair where the premise is the concatenation of the top-k retrieved chunks (sorted by reranker score) and the hypothesis is the claim itself. The model classifies each pair

into one of three verdicts: *supports* (the evidence entails or strongly corroborates the claim), *refutes* (the evidence contradicts the claim), or *insufficient* (no evidence is found to confirm or deny the claim). Each verdict is accompanied by a confidence score $c \in [0,1]$ and a brief justification string. A confidence threshold $\tau = 0.6$ is applied: verdicts below this threshold are downgraded to *insufficient* to avoid propagating low-confidence corrections.

Phase 3 – Answer Correction.

The AnswerCorrector receives the original answer, the list of annotated claims with their verdicts, and operates in one of three modes controlled by a configuration parameter: *auto* (automatically rewrite the answer, removing refuted claims and correcting insufficient ones based on retrieved evidence), *suggest* (propose corrections as inline annotations for human review), or *metadata* (annotate the response with uncertainty flags and claim-level verdicts without modifying content). The correction strategy differs by verdict: refuted claims are replaced with source-backed alternative formulations, insufficient claims are removed entirely with a note indicating missing evidence, and supported claims are preserved unchanged. In *auto* mode, the corrector generates a fully rewritten answer in a single LLM call, using a structured prompt that specifies which claims to keep, remove, or replace.

Algorithm 3: Chain-of-Verification (CoVe) Pipeline

```
Require: Answer A, context C = {c_1, …, c_k}, mode m ∈ {auto, suggest, metadata}, threshold τ
Ensure: Verified answer A′
Phase 1: Claim Extraction
    Q ← ClaimExtractor(A, C)
    for each claim q_i ∈ Q do
        Extract: q_i.text, q_i.sources, q_i.idx
    end for
Phase 2: Claim Verification (Batch NLI)
    for each claim q_i ∈ Q do
        premise ← concat(C[top-k])
        (v_i, c_i, j_i) ← ClaimVerifier(premise, q_i.text)
        if c_i < τ then v_i ← insufficient
    end for
Phase 3: Answer Correction
    if m = metadata then A′ ← AnnotateVerdicts(A, Q)
    else if m = suggest then A′ ← InlineAnnotations(A, Q, C)
    else (auto mode)
        Q_keep ← {q_i : v_i = supports}; Q_fix ← {q_i : v_i = refutes}; Q_drop ← {q_i : v_i = insufficient}
        A′ ← AnswerCorrector(A, Q_keep, Q_fix, Q_drop, C)
    end if
    return A′
```

Knowledge Base Pipeline

The knowledge base consists of approximately 1 million Polish-language Wikipedia articles obtained from a Wikimedia dump (licensed under CC BY-SA 3.0). Articles were filtered by minimum length (200 characters) and language (langdetect), then processed through a semantic chunking pipeline that segments text based on embedding similarity between consecutive sentences, using a cosine similarity drop threshold of $\delta = 0.3$ to detect topic boundaries. Chunks were embedded using the multilingual gte-multilingual-base model [20] (768 dimensions) and indexed in a self-hosted Qdrant instance using the HNSW algorithm [19] with parameters $M = 16$ and efConstruct = 128. The final index contains over 9.4 million vectors.

4. Experimental Setup

Test Set Construction

Rather than relying on standard benchmarks (e.g., PolQA [24], MKQA [25]), we generated 1,000 synthetic questions directly from the indexed Wikipedia corpus to ensure full grounding control. The question generator sampled 2,847 random articles and used Qwen3-32B with dedicated prompts per question type (factual, comparative, temporal, multi-hop) to extract questions. Each question was validated via cross-encoder grounding verification: the ground truth passage must appear in the indexed corpus with a reranker score above a minimum threshold. Additionally, all 1,000 generated questions were manually reviewed and corrected by a human annotator to ensure grammatical correctness, unambiguous phrasing, and factual alignment with the source passages. The final dataset is stored in JSONL format with fields for ground truth answer, question type, and source context identifiers.

Evaluation Metrics

We adopt the RAGAS framework [10] with six metrics summarized in Table 2.

Note on Context Precision: since questions were generated from ~2,800 articles but the knowledge base contains ~1M articles, the system may retrieve equally correct passages from different source articles. RAGAS marks these as "irrelevant" because they do not match the original ground truth source, making Context Precision a conservative lower bound of true precision.

Tab. 2. Evaluation metrics used in the ablation study

Metric	Measures	Range
Faithfulness (F)	Claims supported by context	[0,1]
Relevancy (Rel)	Answer addresses the query	[0,1]
Precision (Prec)	Relevant docs in context	[0,1]
Recall (Rec)	Ground truth covered by context	[0,1]
Latency	End-to-end processing time	seconds
Coverage (Cov)	Sub-query diversity in context	[0,1]

Configurations

We designed 12 configurations (Table 3) enabling both isolated evaluation of single components and assessment of their combinations.

Tab. 3. Experimental configurations. ✓ = enabled, × = disabled. Enh. = enhanced grounding prompt with anti-hallucination rules

Config	Enh.	QA	Rer.	CoT	CoVe	Description
baseline	×	×	×	×	off	Naïve RAG
enhanced	✓	×	×	×	off	+ grounding rules
cot_only	×	×	×	✓	off	+ native CoT
reranker	×	×	✓	×	off	+ cross-encoder
cove_auto	✓	×	×	✓	auto	+ CoVe auto
cot_enh	✓	×	×	✓	off	CoT + grounding
multihop	×	✓	✓	×	off	Decomp. + 3-stage
multi+cot	×	✓	✓	✓	off	+ CoT
full	✓	✓	✓	✓	off	All, no verif.
full+auto	✓	✓	✓	✓	auto	+ auto correct
full+meta	✓	✓	✓	✓	meta	+ metadata only
full+sugg	✓	✓	✓	✓	sugg	+ suggestions

Configurations are grouped into three categories: baseline (minimal pipeline), single-component (one module activated relative to baseline), and full pipeline (multiple modules with different CoVe modes). Each configuration uses prompt templates matched to its active modules – enhanced configurations use grounding rules, multihop configurations use decomposition-aware prompts, and CoT configurations activate native reasoning. This coupling is intentional: in RAG systems, prompt engineering is an integral part of each component, not an independent dimension amenable to separate ablation.

Note: The distinction between multi+cot and full lies in the prompt template: full activates the enhanced grounding prompt (explicit anti-hallucination rules, citation formatting, coverage gate), whereas multi+cot uses only the standard multihop template with native CoT enabled.

Infrastructure

All ablation experiments ran on an NVIDIA H100 (96 GB VRAM) virtual machine provided by the Military University of Technology cloud laboratory, with 128 GB DDR5 RAM and Ubuntu 22.04. The LLM was Qwen3-32B (Apache 2.0 license) [23] in 4-bit quantization served via vLLM [12], used both as the RAG generator (for decomposition, answer generation, and CoVe verification) and as the RAGAS (v0.1.10) [10] LLM-as-a-judge, ensuring zero external API cost. Auxiliary models were identical across all configurations: gte-multilingual-base [20] (Apache 2.0) for embeddings, jina-reranker-v2-base-multilingual [14] (CC-BY-NC-4.0) for cross-encoder reranking, and spaCy [13] pl_core_news_md (MIT) for linguistic analysis. Preliminary validation was performed on a 500-question subset with 5 independent repetitions per configuration, confirming low inter-run variance (standard deviation below 0.02 for all RAGAS metrics across all configurations) and stable metric rankings. This preliminary phase established that a single full-scale run yields representative results. Following this validation, the full evaluation was conducted on the complete 1,000-question test set. Processing all 12 configurations on 1,000 questions required approximately 11 days of continuous computation at 12 hours per day, precluding multiple full-scale repetitions within the available resource allocation.

5. Results

Ablation Results

Table 4 presents the full results across all 12 configurations. Several patterns emerge. First, every correction module improves over the

baseline in Faithfulness, Relevancy, and Context Recall, confirming the value of pipeline augmentation. Context Precision, however, shows minor decreases for some single-component configurations (e.g., enhanced: 0.443 vs. baseline: 0.463); as noted in Section 4.2, this metric is a conservative lower bound because the open-domain corpus may contain equally valid passages absent from the per-question ground truth, causing RAGAS to penalize correct but unexpected retrievals. Second, the multihop configurations dominate in Faithfulness (0.891) and Context Recall (0.829), while the combination of multihop with CoT achieves the highest Answer Relevancy (0.762). Third, CoVe-augmented configurations show the highest Context Precision (0.522 for the suggest mode) but at very high latency cost (60–64 s) and, surprisingly, reduced Faithfulness compared to configurations without verification.

Tab. 4. Full ablation results on 1,000 Polish-language questions. Best per metric in bold

Config	Faith.	Rel.	Prec.	Recall	Lat. (s)	Cov.
baseline	0.768	0.594	0.463	0.600	2.7	–
enhanced	0.850	0.646	0.443	0.622	7.6	–
cot_only	0.884	0.641	0.440	0.614	9.6	–
reranker	0.838	0.680	0.501	0.698	3.9	–
cove_auto	0.872	0.621	0.448	0.613	44.3	–
cot_enh	0.823	0.653	0.431	0.610	12.7	–
multihop	**0.891**	0.714	0.494	**0.829**	18.4	0.64
multi+cot	0.870	**0.762**	0.493	0.828	25.7	0.62
full	0.881	0.721	0.506	0.823	24.5	0.61
full+auto	0.855	0.732	0.516	0.810	62.8	0.64
full+meta	0.858	0.743	0.498	0.827	60.4	0.62
full+sugg	0.832	0.756	**0.522**	0.810	63.6	0.60

Per-Component Impact

To isolate the contribution of each technique, Table 5 compares single-component configurations against the baseline.

Tab. 5. Single-component deltas relative to baseline

Component	Δ Faith.	Δ Rel.	Δ Recall	Δ Lat.
Reranker	+9.1%	+15.3%	+16.7%	+1.2 s
CoT	+14.2%	+8.5%	+1.6%	+6.9 s
Multihop	+15.6%	+20.3%	+38.4%	+15.7 s
CoVe (auto)	+11.6%	+5.1%	+1.6%	+41.6 s

Note: Deltas were computed from full-precision (3-decimal) metric values before rounding; minor discrepancies may appear when recomputing from the rounded values in Table 4.

Cross-encoder reranking delivers the best cost–quality ratio among all components: +16.7% Context Recall and +15.3% Relevancy at only +1.2 s additional latency. The improvement confirms that promoting more relevant passages to the top of the context window directly benefits both retrieval completeness and downstream generation quality.

Chain-of-Thought achieves the second-highest Faithfulness gain (+14.2%) by forcing structured source analysis before answer generation. Its impact on Context Recall is minimal (+1.6%), which is expected since CoT operates at the generation stage and does not affect retrieval. The moderate latency increase (+6.9 s) reflects the cost of producing reasoning tokens within the <think> block.

Multihop reranking produces the largest improvements across all metrics: +15.6% Faithfulness, +20.3% Relevancy, and a dramatic +38.4% Context Recall. The decomposition mechanism splits complex queries into focused sub-queries, each retrieving targeted evidence that the baseline single-pass retrieval misses entirely. The diversity enforcement step ensures broad coverage, with the Coverage metric confirming that 60–64% of final context documents originate from different sub-queries. The +15.7 s latency overhead is justified by the magnitude of quality improvement.

Chain-of-Verification in isolation yields moderate Faithfulness gains (+11.6%) at very high cost (+41.6 s). Critically, when added to the full pipeline (comparing full at 0.881 vs. full+auto at 0.855), CoVe *reduces* Faithfulness by 2.6 percentage points. This paradoxical degradation is analyzed in Section 6.

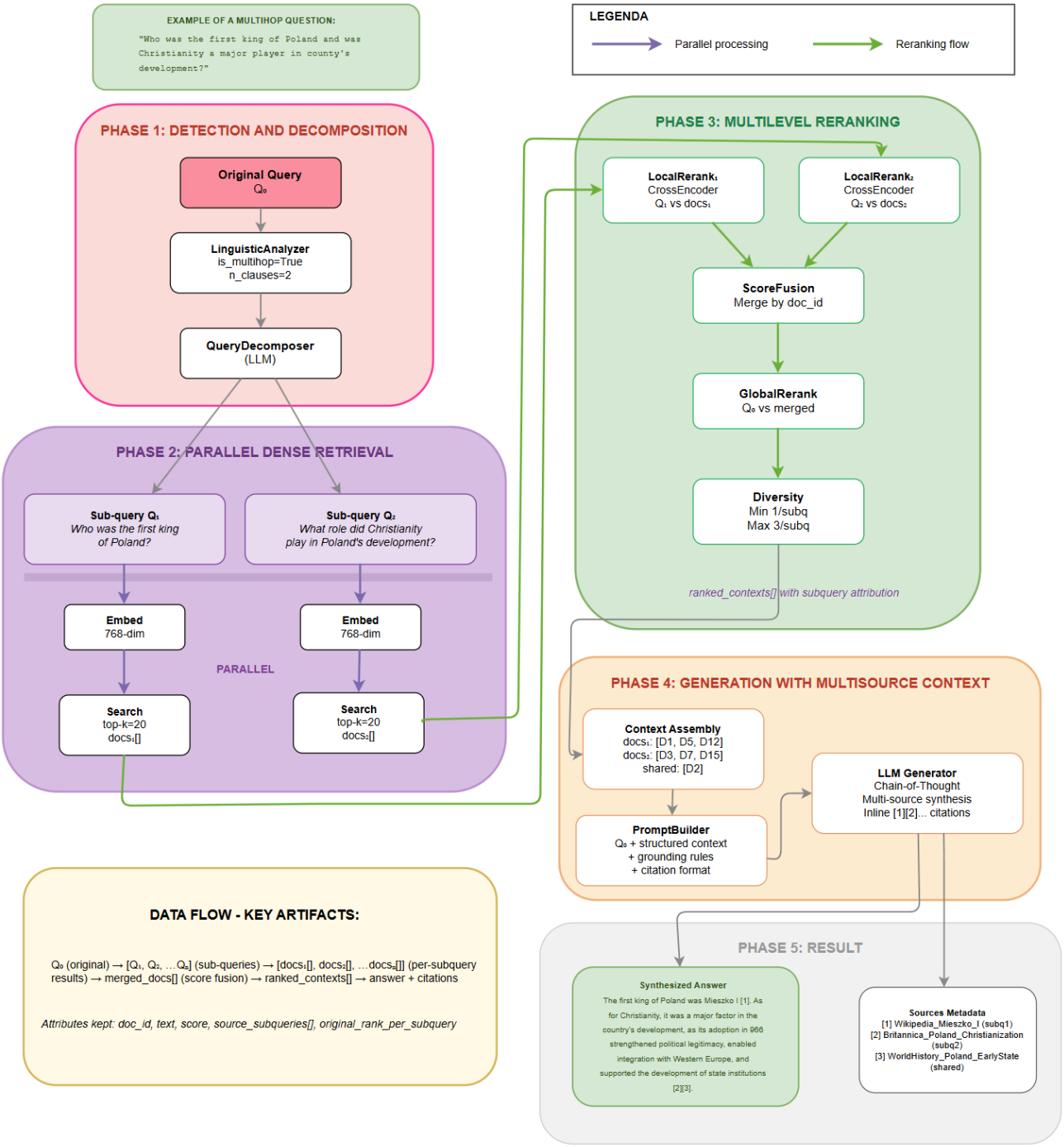


Fig. 4. Multihop reranking pipeline illustrated on an example query. Phase 1: the query is decomposed into focused sub-queries. Phase 2: parallel vector search with embedding retrieval. Phase 3: three-stage reranking (local cross-encoder per sub-query, score fusion, global reranking with diversity enforcement). Phase 4: multi-source generation with context assembly, grounding rules, and citation mapping

Multihop Analysis

Figure 4 illustrates the complete multihop reranking pipeline on an example query. Across all multihop-enabled configurations, the adaptive query rewriter classified 62–64% of test questions as requiring decomposition, with slight variations (615–645 out of 1,000) attributable to LLM non-determinism during classification. The Coverage metric (60–64%) confirms that the diversity enforcement mechanism successfully ensures representation of documents from multiple sub-queries in the final context, validating the design of Algorithm 2's Stage 3.

Latency–Quality Trade-off

Table 6 organizes configurations into practical deployment tiers based on observed latency ranges.

Tab. 6. Deployment tiers based on latency–quality trade-off

Tier	Configs	Latency	Use case
Fast	baseline, reranker	2.7–3.9 s	Real--time chat
Medium	enhanced, cot_only, cot_enh	7.6–12.7 s	Interactive QA
Slow	multihop, multi+cot, full	18.4–25.7 s	Deep search
Very slow	cove_auto, full+auto/meta/sugg	44–64 s	Offline / batch

Only the baseline and reranker configurations meet real-time requirements (<5 s). Multihop configurations without CoVe offer the best quality within a 20–26 s budget acceptable for interactive deep-search applications. CoVe adds approximately 38 s to any configuration, making it impractical for interactive use but potentially valuable for offline batch processing where accuracy auditing is prioritized over speed.

6. Discussion

Retrieval quality dominates generation-stage interventions. The central finding of this study is that improving context quality at the retrieval stage has a substantially larger impact on end-to-end answer quality than manipulating the generation or post-generation stages. Multihop reranking achieves +38.4% Context Recall - meaning the model receives far more complete evidence — which cascades into improvements in Faithfulness (+15.6%) and Relevancy (+20.3%). By contrast, CoT improves Faithfulness through better reasoning over existing context but cannot compensate for missing evidence (+1.6% Recall), and CoVe can only verify claims against already-retrieved passages. This suggests a design principle for RAG systems: *invest first in retrieval completeness and precision before optimizing generation behavior*.

A concrete example illustrates this effect. Consider the comparative query *"Compare the contributions of Mieszko I and Bolesław Chrobry to the Christianization of Poland"* (Figure 4). The baseline single-pass retriever searches for passages semantically similar to the full query, favoring documents that mention both rulers and Christianity together - typically overview articles that lack specific detail about either figure. The multihop module instead decomposes the query into two focused sub-queries (one per ruler), retrieves targeted evidence for each (e.g., the Baptism of 966, Dobrawa's role for Mieszko I; the Congress of Gniezno, establishment of the archbishopric for Bolesław Chrobry), and fuses the results with diversity enforcement ensuring both information needs are represented. The resulting context is both more complete and more precise, enabling the generator to produce a structured comparison grounded in specific historical facts rather than vague summaries.

CoVe auto-correction can degrade quality. The counterintuitive finding that CoVe auto-correction *reduces* Faithfulness in the full pipeline (0.881 → 0.855) deserves careful analysis. We hypothesize two contributing factors: (1) the claim verification step, executed by the same 4-bit quantized Qwen3-32B model, may itself produce classification errors - marking correct claims as "insufficient" and triggering unnecessary removals; (2) the automatic correction phase introduces new text that may not be perfectly grounded in sources, effectively trading original hallucinations for correction-induced ones. The metadata mode (0.858) and suggest mode (0.832) confirm this pattern: less aggressive intervention preserves more of the original answer's faithfulness. We recommend the metadata mode for production deployments, as it flags uncertain claims without risking quality degradation through automated rewriting.

Prompt engineering is inseparable from module ablation. A methodological observation: each correction module relies on dedicated prompt templates (grounding rules for enhanced mode, decomposition instructions for multihop, structured verification prompts for CoVe). Ablating a module simultaneously ablates its associated prompts, meaning the measured deltas capture the *joint* effect of the algorithmic component and its prompt design. This coupling is inherent to RAG systems - prompt engineering is not an orthogonal dimension but an integral part of each component's effectiveness. This finding confirms a key conclusion from our work: carefully designed prompt templates with grounding rules, context structuring, and clear model instructions are necessary conditions for any correction module to produce meaningful improvements.

Limitations. The study has several limitations. First, evaluation is restricted to

Polish, and results may not generalize to other languages with different morphological properties. Second, the RAGAS LLM-as-a-judge uses the same Qwen3-32B model as the generator. We mitigate this concern in three ways:
(a) each RAGAS evaluation call operates as an independent zero-shot assessment with fresh context and dedicated evaluation prompts, so the judge shares no conversational state with the generator;
(b) Zheng et al. [26] demonstrate that strong open-source LLMs achieve high agreement with human annotators when used as judges, supporting the validity of this approach;
(c) the entire project was constrained to open-source models executable on consumer hardware (NVIDIA RTX 5070 Ti for development, H100 for the full ablation), precluding the use of commercial API-served models.
Nevertheless, systematic stylistic biases shared between generation and evaluation cannot be fully excluded, and cross-model validation remains desirable in future work. Third, Context Precision represents a conservative lower bound because the open-domain corpus may contain equally valid passages not present in the per-question ground truth. Fourth, the full ablation was conducted as a single run on 1,000 questions per configuration due to computational constraints (approximately 11 days of continuous processing for all 12 configurations); however, preliminary validation on a 500-question subset with 5 independent repetitions confirmed low inter-run variance and stable metric rankings across all configurations. Finally, use of commercially available API-served models (e.g., GPT-4, Claude) could yield different results for CoVe, where verification quality is directly limited by the model's reasoning capability.

7. Conclusion

We presented a systematic ablation study comparing four families of RAG correction techniques – cross-encoder reranking, multihop query decomposition with diversity enforcement, native Chain-of-Thought reasoning, and Chain-of-Verification – across 12 pipeline configurations evaluated on 1,000 Polish language open-domain questions derived from a 1M-article Wikipedia corpus. Our principal finding is that **multihop reranking with query decomposition** provides the largest end-to-end quality improvement (+15.6% Faithfulness, +38.4% Context Recall, +20.3% Relevancy) at moderate latency (18.4 s), substantially outperforming both generation-stage (CoT: +14.2% Faithfulness) and post-generation interventions (CoVe: +11.6% Faithfulness at +41.6 s). Cross-encoder reranking offers the best cost–quality ratio for latency-sensitive applications (+16.7% Recall at +1.2 s). Paradoxically, automated post-generative verification can degrade answer quality when performed by quantized models, recommending metadata-only annotation for production use.

These results establish a clear design principle: in RAG chatbot systems, *improving context quality at the retrieval stage yields greater end-to-end benefits than post-hoc generation corrections*. Future work should extend this evaluation along several axes: cross-lingual replication on typologically diverse languages (e.g., agglutinative Turkish, isolating Mandarin) to test the generality of retrieval-first design principles; latency optimization of the multihop module through parallel sub-query retrieval and speculative decoding for near-real-time deployment; adaptive pipeline configuration selection driven by query complexity predictors and user feedback; and cross-model judge validation to further strengthen evaluation reliability.

Acknowledgments
The authors gratefully acknowledge the Military University of Technology cloud laboratory for providing the NVIDIA H100 GPU infrastructure used in all experiments. We also thank the anonymous reviewers for their constructive feedback.

8. Bibliography

[1] Lewis P., et al., "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks", *Proc. NeurIPS*, Vancouver, Canada, 2020.
[2] Ji Z., et al., "Survey of Hallucinations in Natural Language Generation", *ACM Comput. Surv.*, vol. 55(12), art. 248 (2023).
[3] Jurafsky D., Martin J.H., *Speech and Language Processing*, 3rd ed. draft, Stanford University, 2024.
[4] Gao Y., et al., "Retrieval-Augmented Generation for Large Language Models: A Survey", *arXiv:2312.10997*, 2023.
[5] Reimers N., Gurevych I., "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks", *Proc. EMNLP*, 3982–3992, Hong-Kong, China, 2019, doi: 10.18653/v1/D19-1410.

[6] Wei J., et al., “Chain-of-Thought Prompting Elicits Reasoning in Large Language Models”, *Proc. NeurIPS’22*, 24824–24837, 2022.
[7] Press O., et al., “Measuring and Narrowing the Compositionality Gap in Language Models”, *arXiv:2210.03350*, 2022.
[8] Dhuliawala S., et al., “Chain-of-Verification Reduces Hallucination in Large Language Models”, *arXiv:2309.11495*, 2023.
[9] Karpukhin V., et al., “Dense Passage Retrieval for Open-Domain Question Answering”, *Proc. EMNLP*, 6769–6781, ACL 2020.
[10] Es S., et al., “RAGAS: Automated Evaluation of Retrieval Augmented Generation”, *arXiv:2309.15217*, 2023.
[11] Farquhar S., et al., “Detecting hallucinations in large language models using semantic entropy”, *Nature*, vol. 630, 625–630 (2024).
[12] Kwon W., et al., “Efficient Memory Management for Large Language Model Serving with PagedAttention”, *Proc. SOSP’23*, 611–626, ACM 2023, doi: 10.1145/3600006.36131.
[13] Honnibal M., et al., “spaCy: Industrial-strength Natural Language Processing in Python”, Zenodo, 2020, doi: 10.5281/zenodo.1212303.
[14] *Jina Reranker v2: Multilingual Cross-Encoder for Text Reranking, Hugging Face model card*, Jina AI, 2024.
[15] Barnett S., et al., “Seven Failure Points When Engineering a Retrieval Augmented Generation System”, *Proc. IEEE/ACM CAIN*, 194–199, ACM 2024, doi: 10.1145/3644815.3644945.
[16] Larsen A.G., et al., “LLM Hallucinations in Conversational AI for Customer Service”, *Int. J. Hum.–Comput. Interact.*, 1–22 (2025), doi: 10.1080/10447318.2025.2580540.
[17] Massenon R., et al., “My AI is Lying to Me: User-reported LLM hallucinations in AI mobile apps reviews”, *Sci. Rep.*, vol. 15 (2025).
[18] Liu N.F., et al., “Lost in the Middle: How Language Models Use Long Contexts”, *arXiv:2307.03172*, 2023.
[19] Malkov Y.A. Yashunin D.A., “Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs”, *arXiv:1603.09320*, 2016.
[20] Zhang X., et al., “mGTE: Generalized Long-Context Text Representation and Reranking Models for Multilingual Text Retrieval”, *Proc. EMNLP*, 393–1412, 2024.
[21] Yang Z., et al., “HotpotQA: A Dataset for Diverse, Explainable Multi-hop Question Answering”, *Proc. EMNLP*, 2369–2380, 2018, doi: 10.18653/v1/D18-1259.
[22] Ouyang L., et al., “Training language models to follow instructions with human feedback”, *Proc. NeurIPS*, 2022, doi: 10.48550/arXiv.2203.02155.
[23] Yang A., et al., “Qwen3 Technical Report”, *arXiv:2505.09388*, 2025.
[24] Rybak P., et al., “PolQA – Polish Question Answering Dataset”, *Proc. LREC*, 12846–12855, 2022.
[25] Longpre S., et al., “MKQA: A Linguistically Diverse Benchmark for Multilingual Open Domain Question Answering”, *Trans. ACL*, vol. 9, 1389–1406 (2021).
[26] Zheng L., et al., “Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena”, *Proc. NeurIPS*, 2023.

Badanie ablacyjne metod korekcji w polskojęzycznych chatbotach opartych na Retrieval-Augmented Generation

S. FLOREK, P. GÓRNY

Generowanie wspomagane wyszukiwaniem (RAG) ogranicza halucynacje dużych modeli językowych poprzez osadzanie odpowiedzi w pobranym kontekście, jednak wzajemne oddziaływanie technik korekcji w potoku RAG pozostaje słabo zbadane - szczególnie w odniesieniu do języków innych niż angielski. Przedstawiamy modułowy system RAG poddany systematycznemu badaniu ablacyjnemu obejmującemu **12 konfiguracji potoku** na **1000 pytaniach testowych w języku polskim** wygenerowanych z korpusu 1 mln artykułów Wikipedii i zwalidowanych ręcznie. Porównano cztery rodziny metod korekcji: (i) reranking z enkoderem krzyżowym, (ii) dekompozycję zapytań z rerankingiem wieloetapowym i wymuszaniem różnorodności, (iii) natywne rozumowanie łańcuchowe (CoT) oraz (iv) postgeneracyjną weryfikację łańcuchową (CoVe). Wyniki ocenione metrykami RAGAS wskazują, że **reranking wieloetapowy przynosi największe zyski jakościowe** (Δ Faithfulness +15,6%, Δ Context Recall +38,4%, Δ Relevancy +20,3%) przy akceptowalnej latencji (18,4 s), podczas gdy CoVe w trybie autokorekty może paradoksalnie *pogorszyć* wierność odpowiedzi przy wysokim koszcie obliczeniowym (+41,6 s). Reranking z enkoderem krzyżowym osiąga najlepszy stosunek kosztu do jakości (+16,7% Recall, +1,2 s). Omówiono praktyczne poziomy konfiguracji dla wdrożeń czasu rzeczywistego, interaktywnych i wsadowych.

Słowa kluczowe: generowanie wspomagane wyszukiwaniem, badanie ablacyjne, reranking wieloetapowy, rozumowanie łańcuchowe, weryfikacja łańcuchowa, przetwarzanie języka polskiego, ewaluacja chatbotów.