Agentic Architectures in Retrieval-Augmented Generation

Tomasz Makowski

January 20, 2026

Table of contents

Problem & Motivation

Problem & Motivation

- Classical RAG can still hallucinate when retrieval is weak or incomplete.
- Retrieval is often one-shot; complex questions need **multi-step search**.
- Agentic approaches can **plan, reflect, debate, and adapt retrieval**.

Key idea: dynamic retrieval and multi-agent reasoning can improve grounding and answer quality.

Research Goals

- **Main question:** Do agentic RAG architectures reduce hallucination and improve answer quality vs vanilla RAG?
- Compare **single-agent** vs **multi-agent** and **iterative** vs **one-shot** retrieval.
- Evaluate across **controlled synthetic data** and **realistic benchmarks**.

Agentic RAG Architectures

Agentic RAG Architectures (in this project)

- **vanilla** – single retrieval + answer
- **self_reflective** – answer → critique → refine
- **query_decomposition** – split multi-hop questions, then aggregate
- **chain_of_verification** – verify extracted claims, rewrite with evidence
- **active_retrieval** – iterative query rewriting until sufficient
- **marag** – Researcher → Analyst → Synthesizer
- **madam_rag** – debaters + moderator for conflict resolution
- **routing** – planner selects best pipeline per question

Implementation

Implementation & Stack

- Python, modular pipeline in `src/`
- LLM providers: OpenAI / Gemini (configurable)
- Retrieval: hybrid dense + lexical, optional reranker
- Metrics: keyword & semantic precision/recall, grounding score, latency, tokens
- Experiment runner: `scripts/run_experiments.py` (YAML-driven)

Synthetic “Future Poland” dataset (controlled)

Purpose: ensure the model **cannot answer from pre-trained knowledge**. Documents were intentionally crafted with cross-references, subtle contradictions, and multi-hop dependencies.

A generator agent was implemented to create the source files; each file contained deliberate inconsistencies or links to other files, and each document was constrained to a target length.

Files contained information about the future of Poland (2025-2125). Each file contained 1000-2000 words. Topics: e.g. Energy Transformation, Education Reform, AI Expansion.

Artifact	Description
`raw/*.md`	Handwritten knowledge base
`questions.json`	JSON with questions and reference answers
`chunks.json`	Chunked corpus used for retrieval

Synthetic Dataset Creation Process

1. **Design themes** (policy timeline + sports domains)
2. **Write source docs** with cross-references and contradictions
3. **Create QA pairs** to cover multi-hop, fact-checking, routing stress tests
4. **Chunking** with overlap + embedding index
5. **Evaluation pipeline** built for reproducible experiments

Why: ensures that answers are grounded in provided documents, not LLM memory.

Benchmark dataset (external)

Source: **Hugging Face `rag-datasets`**
Converted to local `benchmark_files/*.md` + `benchmark_questions.json`.

Scale: **3,200** Markdown files and **918** questions.
Each file is a single Wikipedia paragraph, covering diverse topics (geography, history, sports, etc.).
This setting is harder for retrieval because the corpus is much larger and thematically broader.

Why benchmark? To test if agentic pipelines generalize beyond synthetic data.

Experimental Setup

- Configured through YAML (`configs/experiment.yaml`)
- Runs each architecture for each question
- Asynchronous execution with checkpoints

Metrics:

Metric	Meaning
Keyword precision/recall@k	lexical overlap with retrieved chunks
Semantic precision/recall@k	embedding similarity to answer
Grounding score	token overlap between answer and context
Latency / Tokens	efficiency & cost proxy

Results

Results - Future Poland

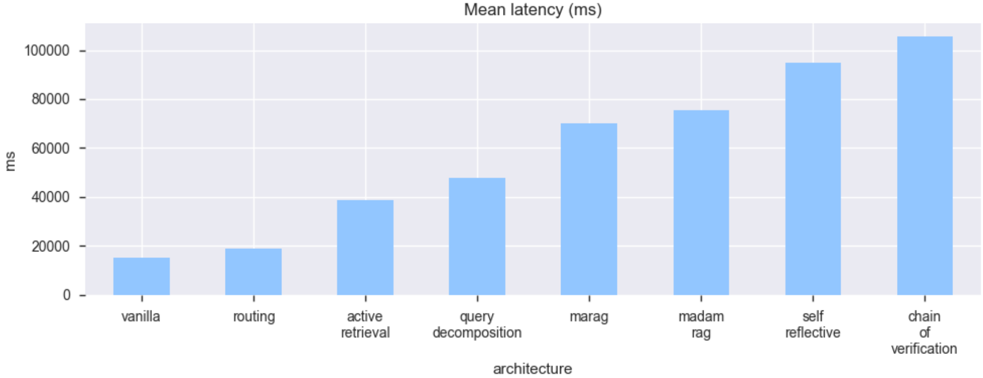

Figure 1: Future Poland: Latency

Results - Future Poland

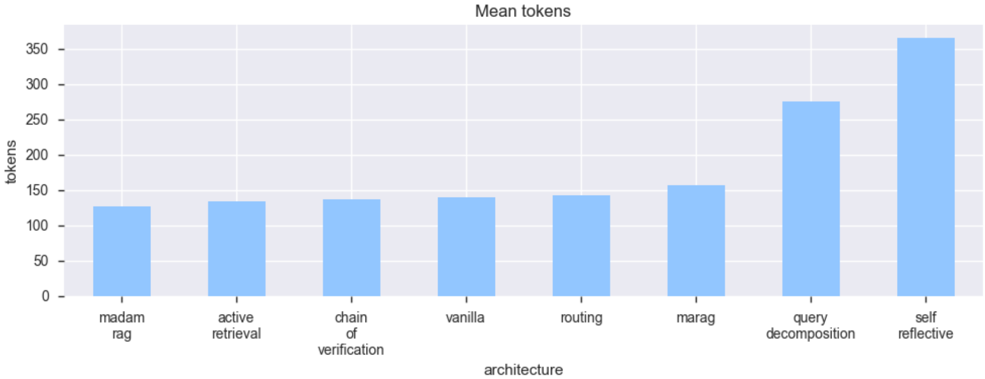

Figure 2: Future Poland: Mean Tokens

Results - Future Poland

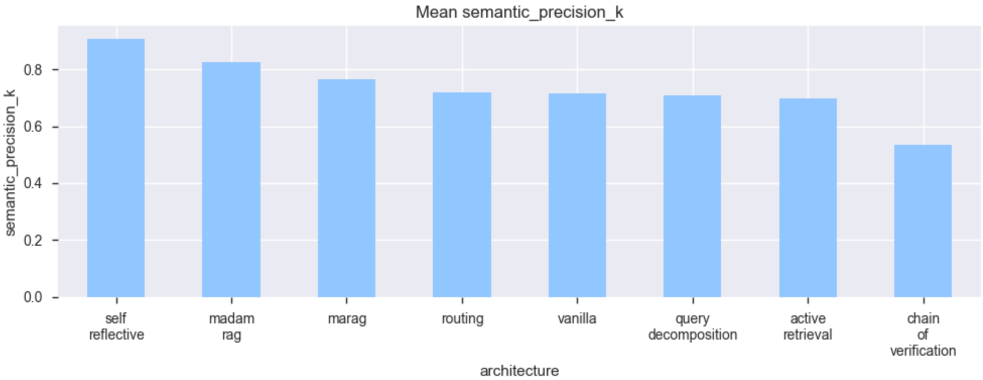

Figure 3: Future Poland: Mean Tokens

Results - Future Poland

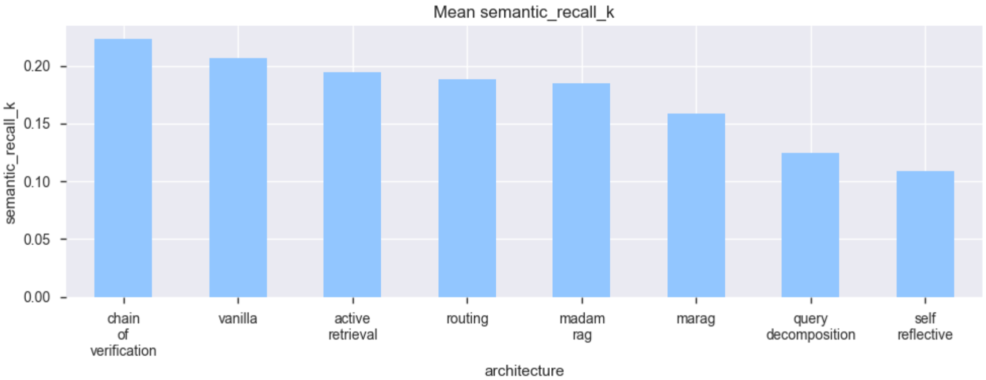

Figure 4: Future Poland: Mean Tokens

Results - Future Poland

Comparison by precision, recall, and grounding score: the number of times each agent was best across these metrics was counted.

architecture	best_count
self_reflective	23
marag	19
active_retrieval	14
routing	13
vanilla	10
chain_of_verification	8
madam_rag	7
query_decomposition	6

Results - Future Poland

A ranking was also computed for each question, and `np.log(s).sum` was calculated, where `s` is the rank position for that question.
The log reduces the impact of extreme low ranks, and the sum yields a single aggregate score across all questions.

architecture	place_product_logsum
marag	106.188469
self_reflective	110.169673
active_retrieval	121.567201
vanilla	122.350255
routing	127.495255
madam_rag	147.584518
query_decomposition	156.248498
chain_of_verification	161.094112

Results - Future Poland

Average rank (lower is better) was computed per question type for each architecture. Cells with **1.0** mark the best result for a given question type.

architecture	abstractive summary	aggregation reasoning	causal reasoning	comparative reasoning	conflict resolution	counterfactual	multi-hop factual	temporal reasoning	vanilla factual
AR	4.0	5.0	5.0	5.0	3.0	2.0	6.0	4.0	2.0
CoV	7.0	8.0	6.0	7.0	8.0	8.0	8.0	8.0	8.0
MADAM	8.0	5.0	7.0	3.0	7.0	4.0	5.0	5.0	7.0
MARAG	**1.0**	2.0	**1.0**	**1.0**	2.0	5.0	**1.0**	**1.0**	3.0
QD	6.0	5.0	8.0	8.0	4.0	7.0	7.0	5.0	6.0
Routing	2.0	4.0	4.0	2.0	5.0	5.0	4.0	5.0	**1.0**
SR	5.0	**1.0**	3.0	4.0	**1.0**	**1.0**	2.0	2.0	5.0
Vanilla	3.0	2.0	2.0	6.0	5.0	3.0	3.0	3.0	4.0

Results - Benchmark

The benchmark questions were generally simpler (e.g. “What is the largest religious group in Canada?”), and despite a larger number of chunks being searched, responses were produced faster and at lower cost; precision was lower while recall was higher, which indicates broader retrieval coverage with more noise.

Results - Benchmark

Lower values indicate better overall ranking (log-sum of per-question ranks); `vanilla` and `self_reflective` were the strongest on this benchmark.

architecture	place_product_logsum
vanilla	17.945994
self_reflective	18.505610
active_retrieval	20.649590
marag	20.852531
routing	24.918288
madam_rag	30.666406
chain_of_verification	31.793606
query_decomposition	36.884528

Conclusions & Future Work

Insights

- Quality gains were observed on multi-hop and conflict questions with agentic pipelines.
- Routing was found to be effective for token management: complex questions were best handled by `self_reflective`, simple factual queries by `vanilla`, and multi-step reasoning by `query_decomposition`.
- Higher quality was achieved at higher cost, so a trade-off was required.

Limitations

- RAG performance on large datasets was difficult to evaluate reliably, and no single metric was found to be sufficient.
- Benchmark questions were often simple, which limited the stress on advanced agent behavior.
- Cost and latency increased sharply for deeper multi-agent pipelines.

Future Work

- Larger and more diverse synthetic datasets should be created.
- Stronger automatic evaluation (e.g., RAGAS, factuality) should be explored.
- Smarter routing should be refined to balance quality and token cost.

Thank you for your attention

Questions

References

References

- MA-RAG (2025): collaborative multi-agent RAG (Planner/Step-Definer/Extractor/QA), on-demand agents, no fine-tuning; wins on multi-hop and ambiguous QA. https://arxiv.org/abs/2505.20096
- MADAM-RAG (2025): multi-agent debate to handle ambiguity, misinformation, and noise; introduces RAMDocs; +11.4 pp EM (AmbigDocs) and +15.8 pp on FaithEval vs strong baselines. https://arxiv.org/abs/2504.13079
- HM-RAG (2025): hierarchical, multimodal RAG (decompose -> parallel retrievers -> decision fusion); plug-and-play sources (text/graph/web). https://arxiv.org/abs/2504.12330
- Survey: Agentic RAG (2025): taxonomy of reflection, planning, tool use, and multi-agent collaboration; good figure source for the "space of methods". https://arxiv.org/abs/2501.09136
- RAG datasets: Hugging Face `rag-datasets` hub. https://huggingface.co/rag-datasets

References