DeepDoctor-RAG: A Knowledge-Graph and Literature-Augmented RAG System for Explainable Medical QA

Dongyi Lu, **Yiwen Xu**

Abstract— **DeepDoctor-RAG is an engineering-oriented medical QA system that grounds large language model (LLM) generation in a structured evidence pack. The system integrates a Neo4j-backed medical knowledge graph with an action-driven retrieval loop constrained by ontology-aligned query templates, deterministic terminology normalization via Medical Subject Headings (MeSH), and best-effort online literature augmentation via PubMed. These design choices prioritize simplicity, determinism, and auditability, enabling explainable retrieval paths and traceable provenance for generated statements. We document the end-to-end architecture, repository implementation highlights, and a pragmatic evaluation protocol based on RAGAS metrics for applied research settings.**

Index Terms—**Evidence grounding, knowledge graph, medical QA, RAG.**

I. Introduction

Large language models (LLMs) have demonstrated remarkable capabilities in natural language understanding and generation, enabling fluent and contextually coherent interactions. These advances are rooted in the Transformer architecture [1], which introduced scalable attention-based sequence modeling and catalyzed subsequent large-scale pretraining paradigms. Models such as Qwen3-Plus [2] exemplify this trajectory by providing high-quality, general-purpose responses and serving as practical backbones for downstream applications. However, despite their fluency and broad coverage, LLMs remain prone to hallucinations, generating plausible but factually incorrect or unsupported statements. In medical question-answering scenarios, such hallucinations can lead to serious harms for patient safety and clinical decision-making. Consequently, systems intended for clinical use must prioritize factual correctness, evidence grounding, and the explicit ability to abstain or defer when reliable information is not available.

II. Method

A. RAG Architecture

DeepDoctor-RAG follows a three-layer retrieval architecture. At the core is a Neo4j-backed medical knowledge graph (KG) that encodes curated relations between diseases, symptoms, diagnostic tests, drugs, and non-pharmacological interventions. Above this graph layer, a terminology layer based on Medical Subject Headings (MeSH) [5] normalizes heterogeneous expressions of clinical entities, and an evi dence layer issues constrained literature queries against PubMed [6] when additional primary sources are required. The overview is shown in Fig 1.

In this subsection we focus on the knowledge-graph layer and its retrieval loop, which provide the main structured backbone for retrieval-augmented generation. Terminology normalization and literature augmentation are discussed in later subsections of the Methods section.

(1) Knowledge-graph schema

The medical knowledge graph is intentionally kept compact and ontology-constrained rather than attempting to replicate a full clinical terminology. The graph is implemented in Neo4j and exposes a small set of typed nodes:

- Disease
- Symptom
- Test
- Drug
- Cure (non-pharmacological intervention)

Each node stores a human-readable *name* attribute. Disease nodes may additionally contain a free-text *description* field summarizing the disease's typical presentation and management. This description is reused directly as evidence during retrieval.

Relationships are centered on disease nodes and encode clinically interpretable links:

- (Disease)–[:HAS_SYMPTOM]–>(Symptom)
- (Disease)–[:RECOMMEND_TEST]–>(Test)
- (Disease)–[:RECOMMEND_TEST]–>(Test)
- (Disease)–[:RECOMMEND_CURE]–>(Cure)
- (Disease)–[:HAS_COMORBIDITY]–>(Disease)

In deployment, the retrieval policy intentionally restricts the action space to disease-centered relations. Symptoms, tests, and treatments are used to infer candidate diseases, and recommendations are derived from disease neighbors. This design limits spurious "shortcut" retrieval behaviors and yields more structured, explainable paths aligned with differential-diagnosis-style reasoning [7].

(2) Knowledge-graph retrieval loop

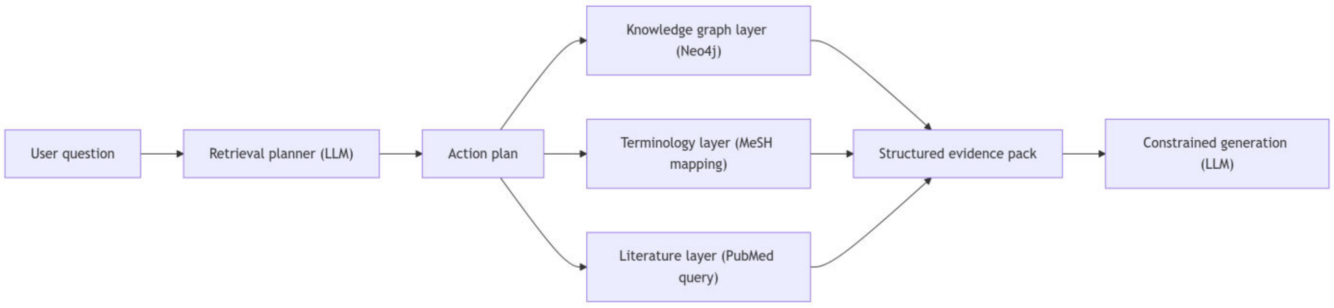
Given a user question, DeepDoctor-RAG performs an iterative, action-driven retrieval loop rather than issuing a single monolithic graph query. Each round proceeds as follows:

1. The current retrieval state is summarized as a textual representation of all previously collected graph triples and disease descriptions.
2. The question, the current retrieval state, and a

Manuscript received January 27, 2026

Dongyi Lu, School of Software, Tiangong University, Tianjin, China,
Yiwen Xu, School of Artificial Intelligence, Tiangong University, Tianjin, China,

Fig 1. DeepDoctor-RAG retrieval layers and evidence flow


User question
Retrieval planner (LLM)
Action plan
Knowledge graph layer (Neo4j)
Terminology layer (MeSH mapping)
Literature layer (PubMed query)
Structured evidence pack
Constrained generation (LLM)

description of the available actions are passed to a small language model that produces a structured retrieval plan.

3. The system executes each action in the plan using fixed, template-based Cypher queries against Neo4j, aggregates the resulting evidence, and updates the retrieval state.

The available actions form a small, symmetric set around disease nodes. Forward actions start from a disease and expand its neighborhood:

- GET_SYMPTOMS_OF_DISEASE
- GET_RECOMMEND_TESTS_OF_DISEASE
- GET_RECOMMEND_DRUGS_OF_DISEASE
- GET_RECOMMEND_CURES_OF_DISEASE
- GET_COMORBIDITIES_OF_DISEASE

Each forward action takes a disease name keyword and returns a bounded set of neighbor nodes. Whenever a disease name appears as a keyword, the system records it in a *mentioned_diseases* set and uses the same keyword to query the graph for an available disease *description*. When present, this description is added to the retrieval result as free-text evidence.

To handle cases where the primary disease is unknown in the user query, the loop exposes inverse inference actions that start from observed symptoms, investigations, or treatments and infer plausible diseases:

- INFER_DISEASES_BY_SYMPTOMS
- INFER_DISEASES_BY_TESTS
- INFER_DISEASES_BY_DRUGS
- INFER_DISEASES_BY_CURES
- INFER_DISEASES_BY_COMORBIDITIES

Each inverse action matches a set of symptom, test, drug, treatment, or comorbidity disease names against the corresponding neighbors in the graph and ranks candidate diseases by a simple, explainable score based on the number of matched neighbors. A configurable *min_match_count* parameter (default 1) specifies how many distinct neighbors must match before a disease is considered a candidate, trading off recall and specificity in a transparent way.

Formally, for an inverse action with an observed neighbor set X (e.g., symptoms), the candidate score is defined as:
$\text{score}\langle d|X\rangle$=($\{x \in X$: (d,x) $\in \mathcal{R}\}$)
and d is retained only if $\text{score}\langle d|X\rangle \geq m$, where m is the configured *min_match_count*.

Across rounds, the retrieval planner uses a moderately high sampling temperature so that the language model can explore different combinations of actions and keywords, analogous to a clinician trying several search phrases in an electronic knowledge base. However, the LLM never directly issues Cypher queries: all database access is constrained to a fixed set of ontology-aligned templates [4]. This separation preserves exploration flexibility at the planning level while keeping retrieval deterministic, auditable, and robust to hallucinated graph paths.

The loop terminates when the LLM returns an empty action list—indicating that the accumulated triples and disease descriptions are sufficient for answer generation—or when a maximum number of rounds is reached. The union of all retrieved relations, together with disease descriptions and later MeSH and literature evidence, then forms the structured context passed to the generation component.

Table. I Action set used by the KG retrieval policy

(For readability, the table uses abbreviated action labels; the full action identifiers are listed above)

Action (abbr.)	Meaning
GET_SYMPTOMS	D→S (retrieve symptom names)
GET_TESTS	D→T (retrieve test names)
GET_DRUGS	D→Rx (retrieve drug names)
GET_CURES	D→C (retrieve cure/treatment names)
GET_COMORBIDITIES	D→Co (retrieve comorbidity disease names)
INFER_BY_SYMPTOMS	S→D (rank diseases)
INFER_BY_TESTS	T→D (rank diseases)
INFER_BY_DRUGS	Rx→D (rank diseases)
INFER_BY_CURES	C→D (rank diseases)
INFER_BY_COMORBIDITIES	Co→D (rank diseases)

Notation: D = disease, S = symptom, T = test, Rx = drug, C = cure/treatment, Co = comorbidity disease.

Algorithm I: KG retrieval loop

```
def retrieve_graph(question, kg, max_rounds, top_k_items):
    state = RetrievalState()
    for _ in range(max_rounds):
        plan = plan_query(question=question, current_retrieval=str(state))
        if not plan.actions:
            break
        for act in plan.actions:
            state = state.merge(kg.execute_action(act, top_k_items=top_k_items))
    return state
```

(3) MeSH and literature evidence

Beyond the core knowledge-graph layer, DeepDoctor-RAG optionally enriches the retrieval state with (i) terminology evidence derived from Medical Subject Headings (MeSH) and (ii) primary literature evidence retrieved from PubMed. This auxiliary layer serves two purposes. First, it provides a standardized and auditable representation of key clinical concepts, mitigating the brittleness of free-form surface names in Chinese clinical text. Second, it supplies high-authority citations that can corroborate graph-derived evidence when such citations exist.

For terminology normalization, the system deterministically maps disease names to MeSH descriptors and attaches structured mapping artifacts to the retrieval output. Each mapping is represented by a stable MeSH identifier (UI) and a traceable decision record containing candidate descriptors, thresholds, and database provenance. When a mapping is selected, the retriever additionally emits terminologydefinition evidence items such as the MeSH preferred name, scope note, and tree numbers. These definition snippets are not used to invent new conclusions; instead, they provide audit trails and consistent concept anchors that can be reused by downstream components [5].

For literature enrichment, the system performs best-effort online retrieval against PubMed [6]. The literature layer is explicitly non-blocking: failure to retrieve publications must not change the system behavior of the knowledge-graph pipeline. When successful, the layer appends a small set of structured citations (including query, PMID, URL, and a compact title/abstract snippet) as additional evidence to be consumed by the constrained generation stage. This design acknowledges the practical limitation that many patient questions will not have directly matching literature, while still allowing authoritative references to improve trustworthiness when available.

B. Data Collection

(1) Knowledge-graph sources

DeepDoctor-RAG targets Chinese clinical question answering; therefore, we build the Neo4j knowledge graph primarily from Chinese resources and preserve Chinese surface forms as node names.

The knowledge graph is constructed from two complementary sources. First, we ingest a curated Chinese medical knowledge graph on common family diseases released by Southeast University [7]. This source provides a reliable backbone of disease-centered relations that are clinically interpretable and relatively low-noise. We programmatically extract eligible nodes and relations from the upstream dataset and remove isolated nodes that do not participate in any relation, ensuring that the final graph remains compact and traversal-friendly.

Second, to increase coverage and capture additional long-tail knowledge, we augment the graph with structured relations extracted from the Huatuo-26M dataset [8]. Specifically, we process the *huatuo_knowledge_graph_qa/train_datasets.jsonl* split [1] (796,444 question–answer items) and use an LLM-based extractor (DeepSeek-V3.2 [10], *deepseek-chat*) to convert free-form text into typed entities and ontology-aligned relations. To control extraction cost and maintain stable throughput, the dataset is processed in fixed-size pages of 1,000 lines per request.

LLM extraction is inherently imperfect and occasionally produces malformed edges (e.g., swapping the subject and object, or assigning a relation to an incompatible pair of node types). Rather than attempting heuristic auto-corrections, we adopt a conservative filtering policy: only relations that match the schema constraints are accepted, and suspicious edges are discarded. This choice avoids injecting speculative structure into the graph and reduces the risk of hallucinated relations. A side effect is that some extracted nodes become isolated; these nodes are removed as a post-processing step.

Table. II Node statistics of the final KG

Node label	Count
Disease	11264
Symptom	15718
Test	5469
Drug	3828
Cure	544

Table. III Relationship statistics of the final KG

Relationship type	Count
HAS_SYMPTOM	71252
RECOMMEND_TEST	42621
RECOMMEND_DRUG	59736
RECOMMEND_CURE	21047
HAS_COMORBIDITY	13844

(2) MeSH and literature resources

For the terminology layer, we download the official

[1] https://huggingface.co/datasets/FreedomIntelligence/huatuo_knowledge_graph_qa/blob/main/train_datasets.jsonl

MeSH descriptor release (*desc2025*) in XML format and build a local SQLite index for deterministic lookup and mapping [5]. For the literature layer, we query PubMed online at inference time and do not pre-download or pre-index the corpus; this keeps the system lightweight while retaining access to updated biomedical publications [6].

III. Experiment

A. Evaluation Overview

We evaluate DeepDoctor-RAG by comparing two question-answering (QA) systems: a retrievalaugmented system (DeepDoctor-RAG) and a non-retrieval baseline. The two systems share the same generation model, decoding parameters (e.g., temperature), and response formatting constraints. Their only difference lies in the final generation call: DeepDoctor-RAG conditions the generator on the structured retrieval state produced by the KG/MeSH/literature pipeline, while the baseline does not.

To quantify answer quality, we use the RAGAS *NumericMetric* [9] to score each system's output on a normalized 0.0–1.0 scale. Metric computation is performed by a separate evaluator model that is stronger than the tested generators; this reduces the risk of self-evaluation bias and better reflects evidence-groundedness.

B. Experimental Method

We use Qwen3-4B-Instruct-2507 [2] as the generator for both systems, running in a 4-bit quantized format (Q4_K_M) with non-thinking decoding. This choice balances three constraints: (i) the model is sufficiently capable to summarize and synthesize retrieval evidence, (ii) it is not so strong that the baseline ceiling effect obscures the marginal utility of retrieval, and (iii) it is fast enough to run repeated evaluations over a 1,000-example test set.

Prompting policy differs between deployment and evaluation. In deployment, the system can use a stricter prompt that requires the LLM to ground conclusions in retrieval outputs and discourages relying on the model's potentially unsupported parametric knowledge, thereby reducing hallucinations. However, this strict prompting is unsuitable for our controlled comparison: the non-retrieval baseline intentionally receives no evidence context, and under a strict "answer only from evidence" policy it would frequently abstain or return near-empty answers, trivially collapsing scores. Therefore, for evaluation we use an unstrict prompt variant shared across both systems so that differences in performance primarily reflect the presence or absence of retrieval evidence rather than prompt-induced refusal behavior.

The QA benchmark is the Huatuo-26M knowledge-graph QA split, huatuo_knowledge_graph_qa/ test_datasets.jsonl[2] , which contains 1,000 question–answer pairs. For each example, we feed the question to each QA system and obtain a final answer. We then provide the question, the reference answer, and the system output to the RAGAS NumericMetric evaluator. The evaluator assigns a score between 0.0 and 1.0, primarily reflecting helpfulness and factual accuracy. To operationalize this scale, outputs that are severely incorrect or unhelpful are mapped close to 0.0, whereas outputs that are both accurate and clinically useful approach 1.0. We record per-item scores to support aggregate comparison and qualitative error analysis.

C. Results and discussion

We evaluate 1,000 paired QA examples and compare the per-item RAGAS *NumericMetric* scores between DeepDoctor-RAG and the non-retrieval baseline. Table Table 4 summarizes the key aggregate statistics.

Table. IV Summary statistics of RAGAS *NumericMetric* on 1,000 paired test examples

Metric	**Value**
Sample size	1000
DeepDoctor-RAG mean (std)	0.6620 (0.2570)
Baseline mean (std)	0.4583 (0.3326)
Mean difference (RAG–Base)	0.2
Std. dev. of differences	0.38
Paired *t*-test (*t*, *p*)	16.8990, $p < 10^{-4}$
Wins / ties / losses	610 / 130 / 260
Relative improvement (mean)	0.44
Cohen's *d* (paired)	0.53
P(RAG > Base)	0.61

Overall, DeepDoctor-RAG improves the mean score by 0.2037 absolute points over the baseline, corresponding to a 44.46% relative gain. The paired effect size (Cohen's *d*) is 0.5344, indicating a moderate improvement under this metric. On an item-by-item basis, DeepDoctor-RAG outperforms the baseline on 61.00% of the test questions.

Importantly, a 61.00% win rate is not "high" in absolute terms: it is only moderately above the 50% level expected when retrieval provides no useful signal and the two systems behave similarly. This observation is critical for interpreting the results: the aggregate win rate can be viewed as a mixture of (i) questions where retrieval fails to contribute (win probability near 50%), and (ii) questions where retrieval meaningfully matches the query (where the win probability should be substantially higher than 61.00%). Under a simple two-regime model, $P(\text{RAG}>\text{Base})=(1-p)\cdot 0.5+\text{p}\cdot w$ the observed 0.6100 implies $p\cdot(w-0.5)=0.1100$. For example, even if matched retrieval yields a strong per-item win rate of w=0.8, this would correspond to only $p \approx 36.7\%$ of questions receiving effective retrieval support.

Despite this clear aggregate gain, the per-item variance remains substantial. This behavior is consistent with a coverage-limited retrieval setting: for a non-trivial fraction of questions, the KG does not contain sufficiently specific facts to materially change the final answer, and the retrieval-augmented system degenerates to behavior similar to the baseline generator.

This result also highlights a potential mismatch between the evaluation task and the system's target use case. The Huatuo knowledge-graph QA benchmark is closer to medical knowledge examination questions than to patient-centered consultation dialogues. DeepDoctor-RAG is designed to support explainable and auditable reasoning over patient complaints and differential diagnosis cues; therefore, purely knowledge-centric QA may underestimate

[2]https://huggingface.co/datasets/FreedomIntelligence/huatuo_knowledge_graph_qa/blob/main/test_datasets.jsonl

the benefit of structured retrieval evidence in more realistic scenarios.

Nevertheless, the relative score improvement of the retrieval-augmented system is 44.46% over the baseline. Qualitative inspection indicates that when the question is covered by the KG (especially for non-trivial multi-entity queries) and the baseline model is forced to rely on implicit parametric knowledge, DeepDoctor-RAG can derive higher-scoring answers by grounding statements in retrieved relations and disease descriptions. This observation motivates future work on domain-matched evaluation sets and targeted coverage analyses.

IV. Related Work

A. Retrieval-augmented Generation

Retrieval-augmented generation (RAG) couples a parametric language model with a non-parametric retrieval module that fetches external context for conditioning the final generation. The original RAG formulation of Lewis et al. demonstrates that retrieving relevant passages for knowledge-intensive tasks can improve factuality and reduce unsupported statements by grounding generation in evidence [3]. Subsequent clinical and biomedical QA systems largely follow this paradigm, varying primarily in the choice of retrieval corpus (e.g., general web, biomedical literature, institutional documents) and the mechanism for integrating retrieved context into prompts.

DeepDoctor-RAG follows the same evidence-grounding principle, but is designed around structured, ontology-aligned retrieval rather than unconstrained passage retrieval alone. In particular, we prioritize deterministic and auditable retrieval behaviors by restricting graph access to a small set of fixed templates and by representing retrieved context as a structured evidence pack that can be inspected independently of the generator.

B. Graph-based Medical RAG and MedGraphRAG

Graph-based RAG methods extend classical RAG by organizing knowledge into entities and relations, enabling multi-hop retrieval and evidence aggregation for relation-centric questions. In the medical domain, Wu et al. propose MedGraphRAG, which combines triple graph construction with a U-shaped retrieval strategy (U-Retrieval) that balances top-down precise retrieval and bottom-up response refinement to improve safety and reliability in medical generation [4]. The framework additionally emphasizes controlled vocabularies and source documentation to support evidence-based responses.

DeepDoctor-RAG is directly inspired by the MedGraphRAG design goals—namely, to reduce hallucinations and improve auditability by grounding medical answers in structured evidence. However, our system intentionally simplifies the retrieval logic and interfaces to support engineering deployment: rather than building a complex triple-linked graph over heterogeneous documents, we use a compact disease-centered Neo4j schema, a small action space with fixed query templates, and best-effort literature augmentation. As a result, DeepDoctor-RAG does not claim superiority over MedGraphRAG on all benchmarks; instead, it offers a pragmatic, deterministic alternative that is easier to implement, debug, and evaluate in applied settings.

V. Conclusion

DeepDoctor-RAG is a practical medical QA system that reduces hallucinations by grounding generation in a structured evidence pack built from a Neo4j medical knowledge graph, deterministic MeSH normalization, and best-effort PubMed citations. By constraining retrieval to an ontology-aligned action space with fixed query templates, the system remains interpretable and auditable while retaining the flexibility of multi-turn exploration at the planning level. Our evaluation shows that retrieval conditioning improves evidence-grounded answer quality relative to a matched non-retrieval baseline. The results also highlight a task–coverage mismatch: on knowledge-intensive, professional-style medical questions, a compact disease-centered KG may fail to retrieve sufficiently specific evidence, limiting the marginal benefit of retrieval. Future work will focus on expanding and refreshing KG coverage, strengthening retrieval matching, and validating the system on more patient-centered consultation scenarios.

References

[1] A. Vaswani *et al.*, “Attention is all you need”, in *Advances in neural information processing systems*, vol. 30, 2017.

[2] A. Yang *et al.*, “Qwen3 technical report”, *arXiv preprint arXiv:2505.09388*, 2025.

[3] P. Lewis *et al.*, “Retrieval-augmented generation for knowledge-intensive nlp tasks”, *Advances in neural information processing systems*, vol. 33, pp. 9459–9474, 2020.

[4] J. Wu *et al.*, “Medical graph rag: Towards safe medical large language model via graph retrieval-augmented generation”, *arXiv preprint arXiv:2408.04187*, 2024.

[5] National Library of Medicine, “Annual MeSH Processing: November–December 2024”, 2024. [Online]. Available: https://www.nlm.nih.gov/mesh/annual_process/annual_process_overview.html

[6] National Library of Medicine (US), “PubMed”, 2026. [Online]. Available: https://pubmed.ncbi.nlm.nih.gov

[7] Z. Wang *et al.*, “Chinese medical knowledge graph on common family diseases”, 2022. [Online]. Available: http://data.openkg.cn/dataset/medicalgraph (in Chinese)

[8] J. Li *et al.*, “Huatuo-26m, a large-scale chinese medical qa dataset”, *arXiv preprint arXiv:2305.01526*, 2023.

[9] VibrantLabs, “Ragas: Supercharge Your LLM Application Evaluations”, 2024. [Online]. Available: https://github.com/vibrantlabsai/ragas

[10] A. Liu et al., “Deepseek-v3. 2: Pushing the frontier of open large language models”, *arXiv preprint arXiv:2512.02556, 2025.*