Developing a Benchmarking Framework for RAG Systems Feedback Bericht

von Jan Albrecht

I. Zusammenfassung der Präsentation

Ich habe in der Präsentation das Framework für RAG-Systeme vorgestellt und bin auf die Besonderheiten eingegangen, die bei der Umsetzung relevant sind, insbesondere die modulare Architektur eines RAG-Systems und die verschiedenen Aspekte der Evaluation. Ersteres fokussiert sich darauf, dass möglichst viele RAG-Varianten und IRA-Methoden (Iterativ, Rekursiv, Adaptiv) mit dem Framework abgebildet werden können. Die Evaluation kann in End-to-End-Evaluation und Komponenten-Evaluation unterschieden werden. Während End-to-End-Evaluation für den Endnutzer relevanter ist, kann Komponentenevaluation zur Fehleranalyse und Hyperparameter Tuning verwendet werden.

II. Zusammenfassung der Feedbacks

Das Feedback war positiv und der Plan zur Umsetzung stimmig. Es wurde eingeworfen, den Fokus mehr auf das Framework zu richten. Es kam die Befürchtung auf, dass mit zunehmender Arbeit für die Benchmarks, die eigentliche Umsetzung des Frameworks zu kurz kommt. Besonderer Fokus sollte auf die Rekonfigurierbarkeit gelegt werden. Neben der potenziellen Evaluation durch typische Performance-Metriken wie Precision, Recall und F1-Score, sollten Hardware Metriken wie RAM, Inferenzzeit und CPU-Zeit verwendet werden, da diese in der Praxis eine wichtige Rolle spielen. Zudem wurde empfohlen, das Paper der SWS-Abteilung "A Methodology for Evaluating RAG Systems: A Case Study On Configuration Dependency Validation" zu lesen und zu analysieren. Der entwickelte Blueprint soll dabei auf seine Anwendbarkeit in der praktischen Umsetzung überprüft werden. Außerdem wurde darauf hingewiesen, dass SLURM für die Experimente eingesetzt werden kann.

III. Angepasste Umsetzung

Für die Anpassung der Umsetzung muss die Implementierung des Frameworks und das Evaluieren durch einen Software Engineering Task angepasst werden.

A. Angepasste Umsetzung des Frameworks

Das Framework, basierend auf dem von der SWS-Abteilung veröffentlichten Blueprint, sollte folgende Voraussetzungen erfüllen:

- Einfache Integration der benötigten Kontextressourcen
- Hohe Abdeckung möglicher RAG-Varianten
- Hohe Abdeckung möglicher Evaluatonsmetriken
- Eine LLM- oder Naive-RAG-Baseline
- Aufteilung der Daten in Validation & Testdatensatz (Fehleranalyse/Hyperparameter-Tuning & Bestimmung des Generalisierungsfehler)

Weiterhin wurden die Anforderungen gestellt, dass das Framework einfach zu rekonfigurieren sein soll, sodass die Evaluationen nach Fehleranalyse einfach zu replizieren sind. Replizierbarkeit und Interne Gültigkeit der Experimente sind Voraussetzungen für das Framework.

Das Framework wird neben Task-spezifischen Metriken, Hardware-Metriken nutzen. Darunter zählen zum Beispiel Inferenzzeit, RAM und CPU-Zeit. Daher ist es wichtig, dass die Evaluierungen nicht parallel gestartet werden, sondern isoliert von anderen Prozessen und sequenziell auf einer dedizierten Hardware.

1) Analyse bestehender Frameworks und Bibliotheken: Für die Neuausrichtung des Fokus, werde ich noch sicherstellen, dass ein solches Framework noch nicht existiert.

a) AutoRAG[7]: AutoRAG ist ein Tool für Hyperparameter-Tuning für RAG-Systeme. Der Nutzer erstellt eine Konfigurationsdatei, welche die sogenannten Nodes (Retriever, Generator, Reranker) definiert und die Hyperparameter (Top-K, Models, Prompts) angibt. Danach läuft die Analyse und es wird die Konfiguration des besten Systems angegeben.

Das Tool nutzt nur End-to-End-Evaluation und ist schwierig anpassbar. Laut den Entwicklern im Discord sind fortgeschrittene Methoden (IRA) derzeit nicht möglich. Es bietet keine Möglichkeit ein Baseline-LLM zu evaluieren.

b) Llama-Index[8] & Langchain[2]: Llama-Index und Langchain sind umfangreiche Bibliotheken für die Entwicklung von RAG-Pipelines. Sie bieten jedoch keine zeiteffiziente Möglichkeit zum Benchmarken vieler RAG-Systeme.

c) RaLLe[3]: RaLLe ist eine Bibliothek zur zeiteffizienten Entwicklung von RAG-Systemen. Der letzte Commit ist über ein Jahr alt und die Bibliothek scheint sehr restriktv. Unter anderem ist es nur möglich .tsv-Dateien mit einer vorgegebenen Struktur zu nutzen. Fortgeschrittene Methoden sind nicht möglich.

d) FlashRAG[6]: FlashRAG ist ein Python Toolkit, dass eine ähnliche Motivation hat, wie ich sie hier versuche umzusetzen. Es bietet eine Konfigurationsdatei als Start, es hat eine Auswahl von Task-spezifischen Metriken und eine

Vielzahl an RAG-Komponenten. Es lassen sich viele RAG-Varianten und einige Spezialfälle damit umsetzen (IRA, Self-RAG[1], FLARE[5], IRCoT[13]).

e) RAGLAB[14]: RAGLAB ist das neueste Framework und bietet ähnliche Funktionalitäten wie dessen Vorgänger. Es führt "Fair Comparison" ein, einen faireren Vergleich für LLM's und Retriever zum Beispiel das Entfernen von Special Tokens im Training oder dem Seedsetzen in Retrieval und Generator Pipelines.

f) Haystack[9] & FastRAG[4]: Haystack ist ähnlich wie Llama-Index und LangChain eine umfangreiche Bibliothek für die Entwicklung von RAG-Pipelines. Sie bietet Pipeline Definition mittels YAML oder JSON Dateien und ist stark modular aufgebaut. Es lassen sich IRA Methoden damit entwickeln. FastRAG ist kompatibel mit Haystack und hat eine Vielzahl an RAG-Komponenten entwickelt, mit denen es möglich ist Spezialfälle aus Papers zu entwickeln. Darunter zählen zum Beispiel REPLUG[12], PLAID[10], ColBERT[11] oder quantized LLM's.

2) Einordnung bestehender Frameworks und Bibliotheken: Bei jeder Analyse wurden zwei Punkte betrachtet:

- Bietet das untersuchte Frameworks bereits die Lösung für das in der Masterarbeit untersuchte Thema?
- Kann ich das Framework als Fundament für mein Framework nutzen?

Keines der Frameworks bietet einen vollständige Pipeline für das Entwickeln von RAG-Varianten durch iteratives Rekonfigurieren und systematischer replizierbarer Evaluierung mit Holdout-Testset. Eine LLM-Baseline zur Überprüfung, ob ein RAG-System eine Verbesserung darstellt, ist in keinem der Frameworks vorhanden. Weiterhin misst keines der Frameworks den Einfluss der Komplexität auf Metriken wie Inferenzzeit, RAM-Usage oder CPU-Time. Damit ist der erste Punkt beantwortet. Für den zweiten Punkt können einige Frameworks herausgefiltert werden. RaLLe und AutoRAG sind sehr restriktiv und es lassen sich nur eine begrenzte Anzahl von RAG-Varianten umsetzen. Llama-Index und LangChain sind ebenfalls restriktiv und nicht zeiteffizient genug, um viele verschiedene RAG-Varianten zu testen. RAGLAB bietet keine Pipeline-Definition durch Konfigurationsdateien wie YAML oder JSON an. Dies könnte die Entwicklungsgeschwindigkeit stark reduzieren.

FlashRAG und Haystack+FastRAG wären mögliche Kandidaten als Basis für das geplante Framework. Ich werde Haystack und FastRAG nutzen, da Haystack stark modular aufgebaut ist und in zukunft eine UI bieten wird um RAG-Systeme per Drag&Drop zu erstellen. Die daraus resultierenden RAG-Varianten können in YAML's exportiert werden und für mein Framework benutzt werden.

Zusammenfassend wird sich die Umsetzung des Frameworks auf die Erweiterung von Haystack und FastRAG beschränken, sodass ein Benchmarking für RAG-Systeme nach dem Blueprint des durch die SWS-Abteilung veröffentlichten Paper durchgeführt werden kann.

Mit Haystack als Fundament werden einige Funktionalitäten bereits erfüllt sein. Die Vielfalt an Rag-Varianten durch Angabe einer Konfigurationsdatei ist bereits möglich. Weiterhin bietet Haystack sowohl eigene Metriken für Komponenten- und End-to-End-Evaluation an. Das Framework mit DeepEval, UpTrain und RAGAS integrierbar. Somit lassen sich die meisten Performance-Metriken nur mit zusätzlichen Aufwand berechnen, da sie nicht durch die Konfigurationsdatei definiert werden können.

Abschließend lässt sich zusammenfassen, dass Kontextressourcen und RAG-Architekturen vollständig durch Haystack abgedeckt sind. Die Evaluierung wird in der Umsetzung dieses Frameworks a) nutzerfreundlicher gemacht, in dem sie in der Konfigurationsdatei angegeben werden kann und b) durch weitere relevante Metriken wie Hardware-Metriken erweitert. Das Framework wird per default für ein LLM und ein Naive-RAG als Baseline ausführen. Nachfolgend wird die Konfiguration der RAG-Architekur ausgelesen und die Evaluation durchgeführt. Die Konfigurationsdatei sollte Matrix-Schreibweisen zulassen, sodass verschiedene Hyperparameter getestet werden können. Die Testdaten sollten automatisch in Validation & Holdout-Testset aufgeteilt werden.

Offen bleibt die Frage, wie die Fehleranalyse einfacher gestaltet werden kann.

B. Angepasste Umsetzung der Benchmarks

Der Fokus auf die Umsetzung des Frameworks bedeutet, dass der Umfang der Benchmark-Experiment reduziert werden muss. Anstelle eigener Experimente werde ich mich auf bereits existierende Experimente konzentrieren. Mögliche Experimente sind Configuration Validation mit dem CTest Datensatz und Vulnerability Detection mit den Datensätzen SySeVR, diversful oder bifi.

REFERENCES

[1] Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. Self-rag: Learning to retrieve, generate, and critique through self-reflection. http://arxiv.org/pdf/2310.11511v1.

[2] Harrison Chase. LangChain. https://github.com/langchain-ai/langchain, October 2022.

[3] Yasuto Hoshi, Daisuke Miyashita, Youyang Ng, Kento Tatsuno, Yasuhiro Morioka, Osamu Torii, and Jun Deguchi. Ralle: A framework for developing and evaluating retrieval-augmented large language models. https://arxiv.org/abs/2308.10633, 2023.

[4] Peter Izsak, Moshe Berchansky, Daniel Fleischer, and Ronen Laperdon. fastRAG: Efficient Retrieval Augmentation and Generation Framework. https://github.com/IntelLabs/fastrag, February 2023.

[5] Zhengbao Jiang, Frank F. Xu, Luyu Gao, Zhiqing Sun, Qian Liu, Jane Dwivedi-Yu, Yiming Yang, Jamie Callan, and Graham Neubig. Active retrieval augmented generation. http://arxiv.org/pdf/2305.06983v2.

[6] Jiajie Jin, Yutao Zhu, Xinyu Yang, Chenghao Zhang, and Zhicheng Dou. Flashrag: A modular toolkit for efficient retrieval-augmented generation research. *CoRR*, abs/2405.13576, 2024.

[7] Jeffrey Kim and Bobb Kim. AutoRAG. https://github.com/Marker-Inc-Korea/AutoRAG, February 2024.

[8] Jerry Liu. LlamaIndex. https://github.com/jerryjliu/llama_index, 11 2022.

[9] Malte Pietsch, Timo Möller, Bogdan Kostic, Julian Risch, Massimiliano Pippi, Mayank Jobanputra, Sara Zanzottera, Silvano Cerza, Vladimir Blagojevic, Thomas Stadelmann, Tanay Soni, and Sebastian Lee. Haystack: the end-to-end NLP framework for pragmatic builders. https://github.com/deepset-ai/haystack, November 2019.

[10] Keshav Santhanam, Omar Khattab, Christopher Potts, and Matei Zaharia. Plaid: An efficient engine for late interaction retrieval, 2022.

[11] Keshav Santhanam, Omar Khattab, Jon Saad-Falcon, Christopher Potts, and Matei Zaharia. Colbertv2: Effective and efficient retrieval via lightweight late interaction, 2022.

[12] Weijia Shi, Sewon Min, Michihiro Yasunaga, Minjoon Seo, Rich James, Mike Lewis, Luke Zettlemoyer, and Wen tau Yih. Replug: Retrieval-augmented black-box language models, 2023.

[13] Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions. http://arxiv.org/pdf/2212.10509v2.

[14] Xuanwang Zhang, Yunze Song, Yidong Wang, Shuyun Tang, et al. RAGLAB: A modular and research-oriented unified framework for retrieval-augmented generation. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Association for Computational Linguistics, December 2024.