基于大模型的RAG算法的快速评估系统：RGE-Pipeline

路思远[1,2]，叶尔潘·托合提亚尔[1,2]，朱苑晔[2,3]，施禹伯[1,2]，王美琪[4]，王中风[1,4*]

（1. 南京大学电子科学与工程学院，江苏南京 210023；2. 南京风语智能信息技术有限公司，江苏南京 210000；
3. 南京大学历史学院，江苏南京 210023；4. 中山大学集成电路学院，广东深圳 518107）

摘 要： 检索增强生成（Retrieval-Augmented Generation，RAG）技术将大语言模型（Large Language Model，LLM）与检索系统相结合，凭借可溯源、可解释、知识更新成本低等优势，已成为LLM落地的主流方案。然而，RAG系统上线前需要经过严苛评估，开发者要构建大规模知识库，并以数千条查询进行全面测试，其间涉及密集检索与反复LLM调用，导致评估耗时极长，严重制约企业级AI的研发迭代效率。为破解这一瓶颈，本文提出RGE-Pipeline（检索器-生成器-评估器流水线）系统，面向基于LLM的RAG算法提供高吞吐、可扩展的快速评估方案。首先，通过预实验对典型RAG系统进行量化分析，定位检索器初始性能瓶颈，引入BM25S替代传统BM25算法，使检索耗时占比降到4%以下，将瓶颈转移至生成与评估阶段。在此基础上，RGE-Pipeline从3个层面进行系统级优化：其一，将评估流程解耦为检索、生成、评估3个模块，引入流水线并行架构，消除串行等待与模型反复加载带来的开销；其二，基于vLLM推理框架设计精细化硬件资源管理方案，支持在同一组GPU上并发部署多个LLM实例；其三，构建数学模型，揭示生成器与评估器之间显存分配比例与系统整体吞吐率的定量关系，并提出3种GPU资源分配策略——显存共享、整卡分配与混合分割，通过平衡两阶段计算负载实现吞吐率最大化。基于CRUD-RAG数据集（涵盖文本续写、摘要生成、多文档问答与幻觉修改等任务，共计6 400条查询）的实验证明，在固定使用BM25S、生成器与评估器均采用Qwen2.5-7B的条件下，RGE-Pipeline展现出显著的加速效果。相比原始串行工作流（耗时约95 h），混合分割方案将总评估时间压缩至1.3 h，加速比达71.7倍；相比模型预加载工作流（耗时约10.8 h），加速比达8.2倍。此外，扩展性实验也表明，RGE-Pipeline在不同知识库规模（18 KB ~ 60 MB）及小规模查询集上均具备良好的适应性。总之，RGE-Pipeline不仅大幅降低了RAG算法的验证成本，还为多LLM并行推理场景的系统优化提供了可借鉴的设计思路。

关键词： 大语言模型（LLM）；检索增强生成（RAG）；BM25（Best Matching 25）算法；基于知识库的问答；vLLM；CRUD-RAG

基金项目： 江苏省科技重大专项（BG2024032）

中图分类号： TP391.1；TP183 **文献标识码：** A **文章编号：** 0372-2112(2026)02-0750-15

电子学报URL:http://www.ejournal.org.cn **DOI:**10.12263/DZXB.20250372

RGE-Pipeline: Fast Evaluation for LLM-Based Retrieval-Augmented Generation Systems

LU Siyuan[1,2], YEERPAN Tuohetiyaer[1,2], ZHU Yuye[2,3], SHI Yubo[1,2], WANG Meiqi[4], WANG Zhongfeng[1,4*]

(1. *School of Electronic Science and Engineering*, *Nanjing University*, *Nanjing*, *Jiangsu* 210023, *China*;
2. *Nanjing Windyword Intellgence Information Technology Co.*, *Ltd.*, *Nanjing*, *Jiangsu* 210000, *China*;
3. *School of History*, *Nanjing University*, *Nanjing*, *Jiangsu* 210023, *China*;
4. *School of Integrated Circuits*, *Sun Yet-Sen University*, *Shenzhen*, *Guangdong* 518107, *China*)

Abstract: Retrieval-augmented generation (RAG), which integrates large language model (LLM) with retrieval systems, has become a mainstream solution for LLM deployment due to its advantages in traceability, interpretability, and low cost for knowledge updates. However, before deployment, RAG systems require rigorous evaluation: developers must construct large-scale knowledge bases and conduct comprehensive tests with thousands of queries, involving intensive retrieval computations and repeated LLM calls. This results in extremely time-consuming evaluation processes, severely hindering the development and iteration of enterprise-level AI systems. To address this bottleneck, we propose RGE-Pipeline (Retriever-Generator-Evaluator Pipeline), a high-throughput, scalable, and fast evaluation framework for LLM-based RAG algorithms. We first conduct preliminary experiments to quantitatively analyze the performance bottlenecks in typical RAG sys-

收稿日期：2025-05-12；录用日期：2026-02-24；责任编辑：覃怀银

*通信作者：王中风

tems, identifying the retriever as the initial bottleneck. By replacing the traditional BM25 algorithm with BM25S, we reduce the retrieval time proportion to below 4%, shifting the bottleneck to the generation and evaluation stages. Building on this, RGE-Pipeline performs system-level optimization from three aspects: (1) decoupling the evaluation workflow into three modules—retriever, generator, and evaluator—and introducing a pipeline parallel architecture to eliminate serial waiting and repeated model loading overhead; (2) designing a fine-grained hardware resource management scheme based on the vLLM inference framework to support concurrent deployment of multiple LLM instances on the same set of GPUs; (3) constructing a mathematical model that reveals the quantitative relationship between the memory allocation ratio of the generator and evaluator and the overall system throughput, proposing three GPU resource allocation strategies—shared VRAM, full-card allocation, and hybrid partitioning—to maximize throughput by balancing the computational load between the two stages. Experiments conducted on the CRUD-RAG dataset, which covers tasks such as text continuation, summarization, multi-document question answering, and hallucination modification with a total of 6 400 queries, demonstrate the significant acceleration achieved by RGE-Pipeline. Under the fixed configuration using BM25S and Qwen2.5-7B for both the generator and evaluator, the hybrid partitioning scheme reduces the total evaluation time from approximately 95 hours (original serial workflow) to 1.3 hours, achieving a speedup of 71.7×, and from approximately 10.8 hours (model-preloading workflow) to 1.3 hours, achieving a speedup of 8.2×. Furthermore, extensibility experiments confirm that RGE-Pipeline maintains strong adaptability across different knowledge base sizes (ranging from 18 KB to 60 MB) and on small-scale query sets. In summary, RGE-Pipeline not only significantly reduces the validation cost of RAG algorithms but also provides a reference design for system optimization in multi-LLM parallel inference scenarios.

Keywords: large language model (LLM); retrieval-augmented generation (RAG); best matching 25 (BM25); knowledge-based QA; vLLM; CRUD-RAG

Foundation Item(s): Jiangsu Province Major Scientific Project (No.BG2024032)

0 引言

自2022年11月ChatGPT发布以来,大语言模型(Large Language Model, LLM)技术得到了广泛关注[1]。随着LLM在各种场景的落地应用,为了增强LLM而提出的众多技术也在快速发展。其中,检索增强生成(Retrieval-Augmented Generation, RAG)在处理大型数据集方面展现出高度的灵活性和性价比,在容错率较低的场景中具备很强的可解释性和溯源性,具有不可替代的优势。一方面,RAG利用外部知识库对LLM进行补充,无需对模型做重新训练即可提供最新信息;另一方面,RAG在特定领域内具有更高的相关性和准确性,这些特性使RAG成为LLM实现垂类应用的理想选择。

尽管大模型与RAG的组合在部署后能带来诸多便利,但其上线前仍需完成多项准备工作,其中算法精度评估尤为关键。为衡量系统在实际环境中的准确性,知识库与测试查询(query)均需具备足够的数据量以支撑量化打分。目前,学术界已涌现出若干高质量评估基准,例如CRUD-RAG[2]和RagChecker[3]。然而,在整个RAG全生命周期中,相较于数据收集、算法选型、模型部署等环节,评估环节的耗时依然最为可观[4]。其复杂性主要体现在5个方面。

(1)为保证系统可靠性,即便部署时仅需接入少量文档,验证阶段所需的知识库规模通常也在10倍以上,以应对客户实际环境中知识库的动态扩展需求。

(2)为全面测试系统效果,企业往往需要使用数千条甚至数万条查询数据进行大规模测试。

(3)评估过程涉及LLM的反复调用与大量推理运算,且每条查询通常至少需要两次LLM调用(一次生成、一次评估),而LLM评估相比BLEU、ROUGE-L等传统指标更贴合用户真实需求,但同时也显著增加了计算负担。

(4)检索器自身的计算开销亦不可忽略[5]。

(5)若项目还需经历多轮算法迭代与重新评估,则整个评估过程的耗时将成为制约产品迭代周期的重要因素。

本文提出的检索器-生成器-评估器流水线(RGE-Pipeline)是一种旨在高效评估基于LLM的RAG算法的创新解决方案。该系统具备高吞吐量、支持大规模知识库接入、利用LLM自动完成评估等优势,能够有效避免繁琐的人工验证流程。在系统设计与优化上,首先引入流水线和并行计算理念对系统计算流程进行深度优化,将RAG算法的评估过程细分为检索器、生成器和评估器3个关键环节,并使其在流水线架构下并行处理,显著提升了整体运算效率。其次,基于高效的向量级大语言模型推理系统(vLLM)[6]构建了一套精细化的硬件资源管理方案,通过合理分配计算资源、优化内存管理等方式,充分挖掘硬件潜能,为系统的高效运行提供坚实支撑。同时,构建了一套数

学模型对系统吞吐率进行分析，精准定位限制因素，并针对性地调整资源分配策略与计算流程，成功探索出一条行之有效的系统吞吐率最大化路径。实验结果有力地验证了RGE-Pipeline系统的卓越性能，在相同的硬件配置下，相较于传统的评估方法，该系统将评估时间从数十小时大幅压缩至1个多小时，评估效率提升显著。这一突破不仅极大程度地减轻了RAG算法评估环节的工作量，降低开发难度，还为大语言模型在知识密集型领域的广泛应用铺平了道路，推动了相关技术的快速发展与落地实施。

本文的主要贡献包括以下4点。

(1)通过预实验分析，定位了未优化RAG系统的性能瓶颈在于检索环节，这为后续专注于生成与评估阶段优化的RGE-Pipeline设计提供了依据。

(2)在解决了检索器的瓶颈问题后，通过巧妙的计算流程设计，结合对vLLM框架的巧妙运用和硬件资源的合理分配，本文提出了RGE-Pipeline这一快速的RAG评估方案，包括3个主要部件，即检索器、生成器和评估器。

(3)围绕检索器、生成器、评估器这3个部件，本文提出了一套数学建模分析方案以最大化地提升系统性能，并针对性地为RGE-Pipeline提出了一套创新的GPU硬件资源分配策略。

(4)在CRUD-RAG数据集上的实验结果表明，RGE-Pipeline相比原始工作流达到70倍以上加速比，相比模型预加载工作流达到8倍以上加速比。

图1是对本文思路的一个直观呈现，包含RAG评估流程的瓶颈分析和本文工作定位。

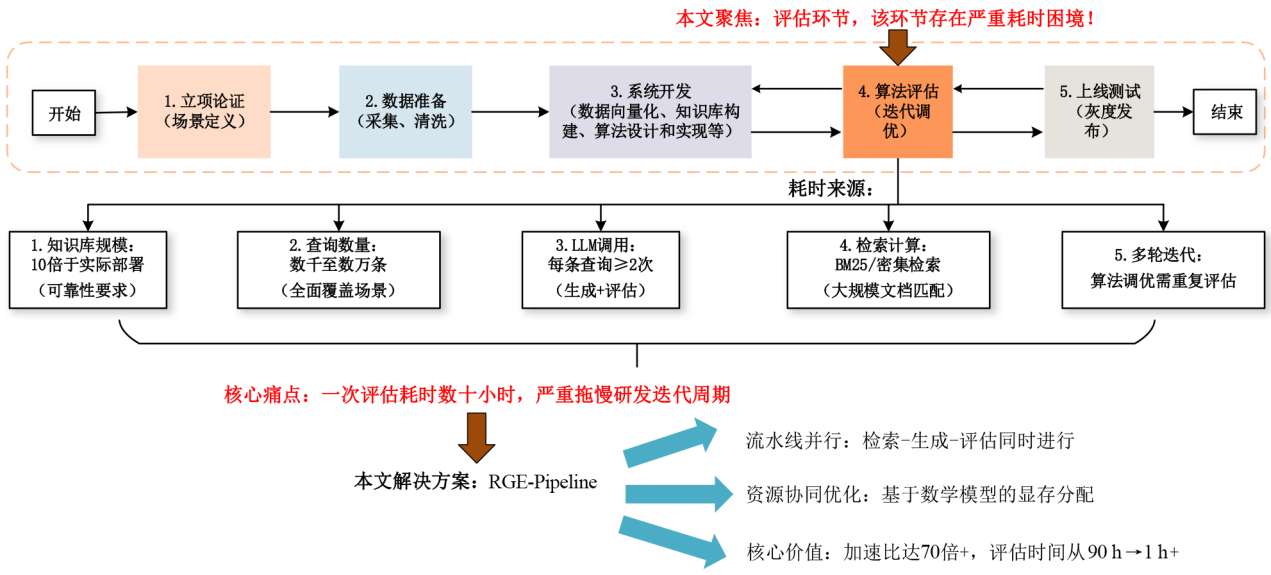


图1 RAG评估流程的瓶颈与本文工作定位

Figure 1 Bottlenecks in the RAG evaluation process and the scope of this work

1 背景

1.1 LLM与RAG

LLM的出现是人工智能领域的一次重大飞跃，其在各种自然语言处理(Natural Language Processing，NLP)任务中展现出了卓越性能[7-9]。从GPT-1到GPT-4的演进，清晰地展示了参数量增长与模型能力提升之间的强正相关关系[10-13]。相比于“AI 1.0”时代专注于图片分类、目标检测、语音识别等感知智能任务的技术，LLM在认知智能层面展现出了强大的逻辑推理与知识整合能力，如DeepSeek-R1[14]等思维链大模型，更是在决策智能层面为复杂问题解决提供了全新的思路与方法。

然而，LLM在应用中面临一些问题，如信息过时、生成内容不准确以及由于数据爆炸性增长带来的检索难度剧增[15]等，其中幻觉问题尤为严重，即模型生成不准确、误导性或无意义的信息。为了解决这些问题，RAG技术应运而生[16-17]。RAG系统通常包括检索、生成和增强3个主要流程。

1.1.1 检索流程

检索流程负责从外部数据源中提取与输入查询相关的信息，两类常用的检索器是稀疏检索器和密集检索器。

(1)稀疏检索器。它是基于单词的方法，如词频-逆文档频率(TF-IDF)[18]的倒排索引匹配(如式(1))和原始数据输入的方法[19]。TF-IDF通过将词频和逆文档频率相乘，得到一个文档-词项对的权重值。这个权重值可以用来评估一个词项对一个文档集中的

某个文档的重要程度,较高的TF-IDF值意味着该词项在文档d中有较高的词频,并且在文档集合D中不常出现。计算方式为

$$\mathrm{TFIDF}(Q,d)=\sum_{t\in Q}\frac{\mathrm{tf}_{t,d}}{l_d}\cdot\mathrm{idf}_t \tag{1}$$

其中,$\mathrm{tf}_{t,d}$指词项t在文档d中出现的次数;l_d指文档d的长度,通常以词数或字数为单位;idf_t指逆文档频率,即包含词项的文档数与总文档数的对数关系的倒数。

TF-IDF的一种常用变体是BM25,具体如下:

$$\mathrm{BM25}(d,Q)=\sum_{t\in Q}\frac{\mathrm{tf}_{t,d}\cdot(k+1)}{\mathrm{tf}_{t,d}+k\cdot\left(1-b+b\cdot\dfrac{l_d}{\mathrm{mean}(l_d)}\right)}\times\mathrm{In}\left(1+\frac{N-\mathrm{df}_t+0.5}{\mathrm{df}_t+0.5}\right) \tag{2}$$

其中:k和b是调节参数,通常设置$k\in[1.2,2]$和$b=0.75$;$\mathrm{mean}(l_d)$指文档集合中所有文档的平均长度;N指文档集合中文档的总数。

BM25是一种常用且有效的稀疏检索方案,但在处理大型数据集和大量并发查询时速度受限。针对这一问题,BM25S作为一种优化解决方案被提出[5]。

(2)密集检索器。它将查询和文档嵌入连续的向量空间,通常基于语义相似性进行检索,具有更高的灵活性和适应性。值得一提的是,在RAG系统中,稀疏和密集检索的混合使用被认为是一种更优越的方法,它充分利用了两种检索方法的优势[20]。

1.1.2　增强流程

增强流程的作用是将检索到的信息与RAG系统中生成器的输出相结合。它通过使用检索到的文档片段作为上下文输入,增强模型对特定问题的理解和回答能力。增强步骤的目标是将外部知识整合到生成过程中,使生成的文本内容更丰富、更准确、更符合用户需求。

1.1.3　生成流程

生成流程负责生成最终的自然语言输出,一般来说就是运用LLM来完成相关任务。它通常根据检索流程提供的相关文档片段,结合输入查询,生成准确、相关度高和自然的文本回复。生成流程中使用的生成器是预先训练好的模型,如GPT系列[9-12]、BART[21]或T5[22],它们利用自身的语言能力和检索到的外部知识来提高生成文本的准确性和丰富性。生成步骤的作用不仅是生成流畅的语言输出,还要确保输出内容既包括模型的固有知识,也包括最新的外部信息。

1.2　RAG评估基准与效率挑战

构建全面、可靠的基准是评估RAG系统性能的基础。近年来,学界相继提出了CRUD-RAG[2]、RGB[23]、NQ[24]等评估基准,它们通常提供大规模知识库与配套查询集,从准确性、忠实度等多个维度对RAG进行量化评估。然而,这些基准在推动技术进步的同时,也揭示了评估流程面临的效率挑战:为获得统计意义显著的结果,需要在数万至数十万个文档的知识库上执行数千乃至数万次查询[3-4]。在此规模下,传统的串行评估流程(检索-生成-评估)暴露出巨大的计算开销,检索器需快速匹配海量文档,而每条查询的生成结果还需调用LLM进行自动化评估,构成反复、大批量的推理负载[2,25]。为确保系统可靠性,开发者要在大规模知识库与海量查询上反复验证,密集的检索计算与LLM调用导致评估耗时极长,严重延长企业级AI应用的开发周期[4]。因此,如何系统性优化RAG评估流程、显著降低其时间成本,已成为兼具工程价值与学术意义的重要课题。

1.3　面向LLM推理的加速技术

RAG评估的核心耗时环节在于对LLM的反复调用。因此,针对LLM推理的加速技术是构建高效评估系统的重要基础。近年来,相关研究在推理框架与系统优化层面取得了显著进展。

vLLM[6]是目前较为流行的LLM推理加速框架之一,其核心创新Paged Attention通过高效管理KV Cache,显著提升系统吞吐量,降低推理延迟。在性能方面,与HuggingFace Transformers相比,vLLM的吞吐量可提升24倍;与HuggingFace TGI相比,可提升3.5倍。此外,vLLM兼容HuggingFace Transformers中的绝大多数生成式模型(如GPT、ChatGLM、Qwen等),为LLM的高效服务提供了通用、灵活的解决方案。

此外,诸如TensorRT、DeepSpeed等推理框架也通过算子融合、内核优化、量化等技术路径来提升LLM的服务效率。这些推理加速技术为处理RAG评估中的大批量、流式LLM调用提供了强有力的底层工具。然而,现有工作主要聚焦于单个LLM服务实例的优化,或简单地将检索、生成、评估视为独立的服务进行部署。如何将这些技术有机地整合到一个统一的、端到端的流水线系统中,并从全局视角对检索、生成、评估3个阶段进行协同优化与资源调度,以最大化整个评估工作流的吞吐率,仍需探索,而本文工作的核心正是为了填补这一空白。

2　预实验和检索器改进

本章旨在通过控制变量实验,量化分析随着知识库规模增长,RAG系统各环节的时间开销变化趋势,从而定位主要性能瓶颈。

2.1　针对典型RAG系统的分解

如图2所示,在典型的RAG系统中,文档预处理是处理用户查询前的关键步骤,是知识库构建的基

础。这一过程通常包括知识库数据采集、文档切分、文档标签/索引生成、嵌入生成和向量存储。这些预处理步骤对于RAG系统后续的检索和生成阶段至关重要，可确保系统能准确、高效地处理用户查询的相关文档信息。

在RAG系统中，用户查询后的处理过程包括检索器和生成器两个主要部分，它们共同承担了系统的大部分计算负荷。典型RAG系统中的检索器采用混合检索，对稀疏检索和密集检索进行了结合。这种双重策略使系统能够实现与文档内容细致入微的语义匹配，同时还能对用户查询中的特定内容进行精确查找。在本文讨论的范围内，生成器主要基于LLM，这在当前的RAG实现中得到了广泛应用。通过利用这些大型语言模型，生成器可以合成与上下文相关并由检索数据提供信息的响应，从而提高输出的整体质量和实用性①。

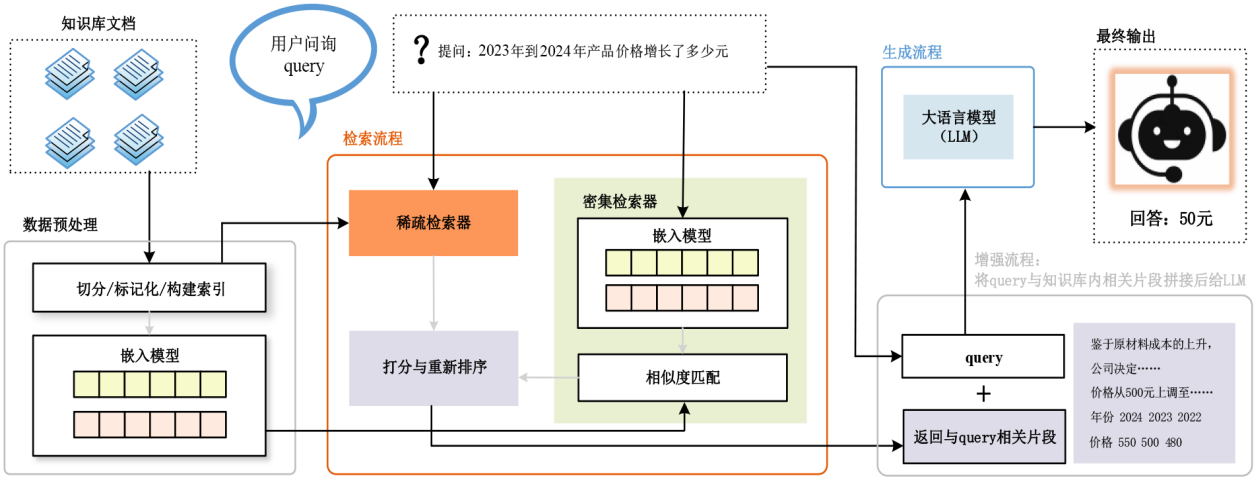


图2 典型RAG系统的顶层架构

Figure 2 Top-level architecture of a typical RAG system

2.2 预实验数据集选择

CRUD-RAG[2]是一套面向RAG系统的综合性评估基准，提供了大规模、高质量的中文数据集，旨在支持多样化的RAG性能测试。相较于RGB[23]、NQ[24]等其他基准，CRUD-RAG的测试平台更为广泛和真实，数据集规模庞大，包含多类任务，能够有效评估RAG系统在知识密集型场景下的交互能力。目前，已有研究将CRUD-RAG作为基准来验证新型RAG方法（如QAEncode[25]、Retriever-and-Memory[26]、Meta-chunking[27]）的有效性，进一步印证了其在推动RAG技术评估方面的实用价值。

2.3 预实验初步设定和结果

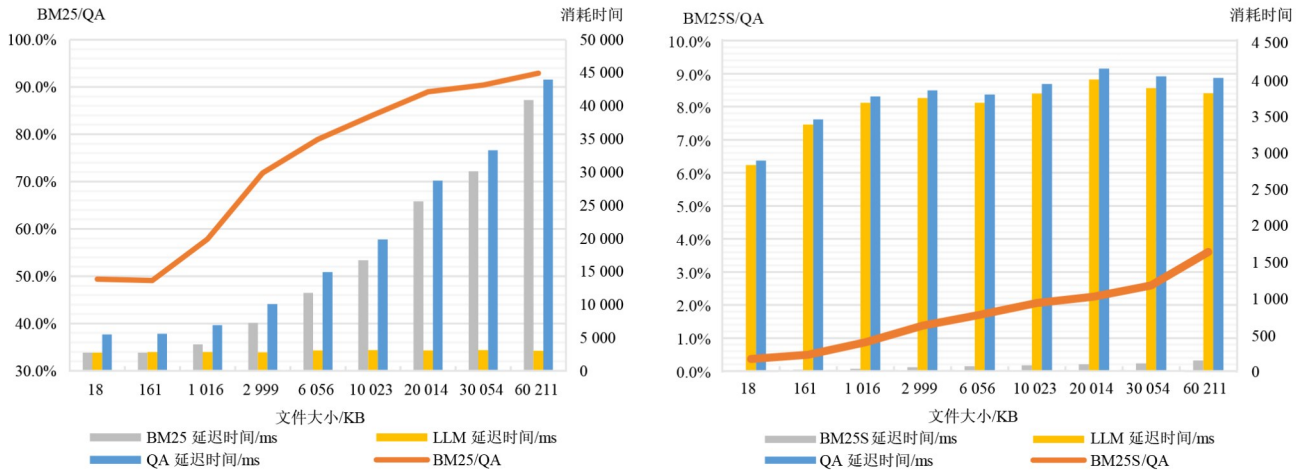
预实验所采用的服务器硬件配置如下：CPU是2.9 GHz的Intel(R) Xeon(R) Platinum 8375C，GPU则使用了一张Nvidia RTX4090。在预实验所搭建的RAG系统中，本文的LLM部分使用了ChatGLM3-6b-INT4，稀疏检索器则使用BM25算法和BM25S算法进行对比。为定位主要延时瓶颈，本文分9次从CRUD-RAG所提供的文档中抽取不同数据量的文件，每次抽取的若干文件的大小展示在图3的横轴上。基于这9次抽取的不同大小的文件，本文构建了9个尺寸各异的文档数据库。之后，本文又从CRUD-RAG数据集中的约7 000条query里随机抽取了120条query。这120条query的每一条都被当作RAG系统的输入提供进来，在每个文档数据库中都分别完成了一次完整的问答(QA)流程。最终，本文统计每个文档数据库所被提问的所有问题的平均值记作QA延迟时间，并且将QA延迟时间中两组检索器和LLM的计算延时都统计了出来，如图3所示。

如图3(a)所示，随着RAG接入的知识库内的文件尺寸不断增大，BM25算法的执行时间迅速增加，而LLM的时间消耗增长较为缓慢。因此，最后系统的主要瓶颈就变成了RAG的检索器而非生成器。这也是本文在后续的优化中会选择BM25S作为基准线(baseline)的原因：相比于BM25算法，BM25S算法的速度最多可提高500倍[5]。BM25S在索引阶段预先计算BM25分数，并将其存储在稀疏矩阵中，以便在查询时快速切分和求和，从而大大提高了检索速度。这一改进不仅提高了检索效率，还保持了BM25算法的准确性，使BM25S成为资源有限或对速度要求较高的应用的理想选择。

① https://www.ibm.com/think/topics/retrieval-augmented-generation

作为对比,本文保持其他设置不变,仅仅将BM25替换为BM25S。从图3(b)中可以看出,BM25S算法的耗时确实大大降低了,只有原先BM25的1%不到。因此,它占整个问答过程的时间比例也非常小,不到4%。得益于这样的优化,从曲线中也可以看出,整体的延时不仅小了一个数量级,而且随着文件大小的增加,延时的增加变得非常缓慢。因此,本文在后续的技术方案设计以及后续实验中,都默认将BM25S作为稀疏检索器所采用的算法而非BM25。

预实验结果表明,在未优化情况下,检索器是主要瓶颈,而使用BM25S这一现有高效检索器后,瓶颈转移到生成阶段。这一发现为本文后续设计专注于生成与评估阶段优化的RGE-Pipeline提供了直接依据。



(a) 稀疏检索器为BM25

(a) Using BM25 as the Sparse Retriever

(b) 稀疏检索器通过BM25S得到了改进

(b) Improving the Sparse Retriever with BM25S

注:"BM25/QA"和"BM25S/QA"两条曲线分别代表BM25和BM25S在整个QA过程中的延时占比。

图3　初步实验结果

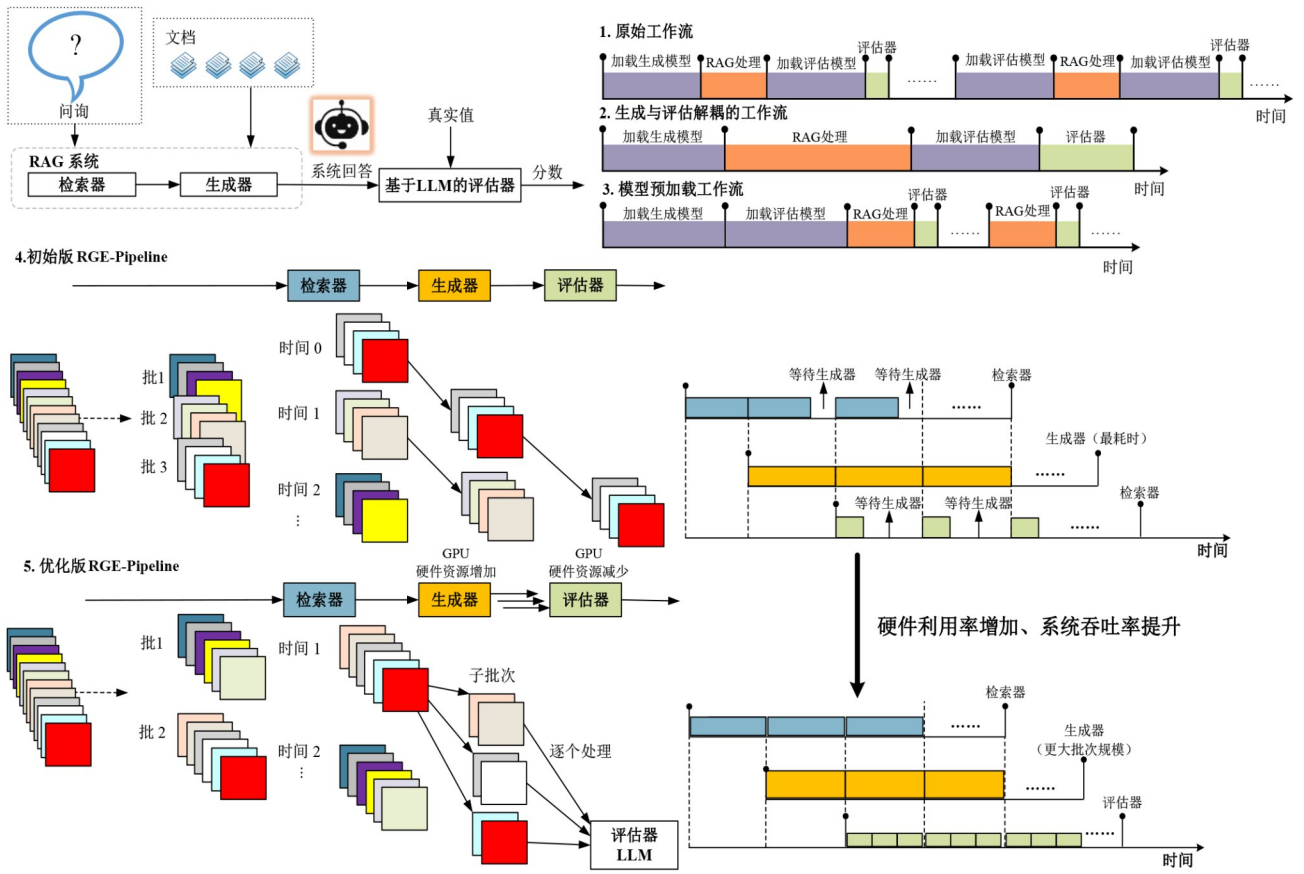
Figure 3　Preliminary experimental results

3　RGE-Pipeline

3.1　RGE-Pipeline的初步构建

RGE-Pipeline系统构建的思路如图4所示。在使用流水线的思想重构整个系统之前,最基本的评估工作思路是等待RAG的处理结果完成后,再加载用于评估RAG结果的LLM(图4的"1.原始工作流")。基于这种直接的思路,如果每次处理完单条query就加载评估模型直接评估,整个系统将面临生成器模型和评估器模型的反复加载,这将带来较大的延时。为了避免这种情况的发生,本文可简单地选择另一种方案:完成全部RAG部分的计算后,再将所有结果全部提供给评估器(对应图4中"2.生成与评估解耦的工作流")。但是这种方法仍然面临一个问题,即对于算法研发人员来说,这种方案单条query的测试结果实时性差。换言之,在批量测试时,每条query从输入到最终评估分数给出,中间经历的时间太长,会影响算法开发的效率,很不实用。一个比较直观的解决思路就是,预先将生成模型和评估模型加载到计算设备上,让每条或者每一小批查询请求都能实时得到计算(对应图4中的"3.模型预加载工作流")。此时,除非生成器中使用的LLM和评估器中使用的LLM完全相同,否则这两者的计算就会处于"交替进行"的状态,其中一个在执行推理计算时,另一个处于被加载到设备上但没有进行计算的状态,最终导致硬件资源的浪费。

但从另一方面来说,"模型预加载工作流"在本文整个优化思路中起到了关键作用,相比于"生成与评估解耦工作流"的做法,它的改进方向也可以被看作是另一种解耦,即加载模型与检索+模型推理计算的解耦。这也使得RGE-Pipeline的流水线策略能够在此基础上展开进一步的设计和实现。首先将整个系统分为检索器、生成器、评估器3个部分,并将其构建为一个完整的流水线。在系统的第一个时刻,检索器负责计算批次1,此时生成器和评估器还未开始计算。在系统的第二个时刻,生成器接收检索器的计算结果并开始计算,而检索器则开始计算批次2。在系统的第三个时刻,检索器、生成器、增强器将全部启动,并分别开始计算批次3、批次2和批次1。之后的



注：左侧是系统几个模块的排布情况，右侧反映了各模块在实际运行中的计算顺序。

图4 RGE-Pipeline构建思路的简单示意图

Figure 4 A simple diagram illustrating the RGE-Pipeline architecture

时刻将以此类推。相比于原始工作流来说，流水线的做法大大提高了系统的效率，它成功地让每条query输入系统后都能快速生成回复并评估RAG的效果，同时也避免了大量冗余的中间结果存储。

初始版的RGE-Pipeline主要是通过对RAG推理和评估计算进行模块化划分以及采用流水线并行处理的方式来实现提速，但正如图4的“4. 初始版RGE-Pipeline”所示，在这种划分方式下，硬件资源仍然无法非常高效地被利用起来，这一问题将在本章的后续内容中得到进一步的详细讨论。

3.2 系统进一步优化的方向和思路

尽管运用流水线和并行计算的思想将整体系统进行重构已经可以提高计算效率，但由于流水线中每个环节的计算量不确定、不平衡，系统仍然存在着较大的提速空间。基于上一章节的分析，在使用BM25S来替换BM25的前提下，检索器不再是系统的瓶颈，而生成器会比评估器更有可能花费更多的时间。生成器往往需要处理更多来自外部知识库的数据（这意味着更长的上下文），所以在评估器和生成器使用的模型大小相当时，生成器往往需要更长的时间来进行计算。

因此，图4就展示了基于这一事实所构建的一种比较直接的优化思路，即将评估器的单次批处理规模降低，将单个批次分解为若干个更小的子批次，这样它就可以节省出更多的硬件资源（显存占用），让生成器可以以更大的批次规模来进行计算。而为了维持流水线的正常运转，评估器需要在一个系统周期内计算多个子批次，从而匹配检索器和生成器的计算速度。这里的初步分析虽然较为粗略且更多的是从工程实践的角度所得到的优化思路，但也揭示了更本质的问题：为了让流水线系统的吞吐率最大化，本文需要重新平衡生成器和评估器模型所占用的硬件资源。

3.3 建模分析

本节构建数学模型的核心工程目标是最大化RGE-Pipeline系统的整体吞吐率T_{rge}。需要指出，影响这些吞吐率的因素有很多，包括硬件资源、大模型的参数量大小、模型的输入输出长度等。但是，如果把上述各种因素全部考虑在内，不仅建模和优化工作无从下手，而且对工程优化没有任何参考价值。本模型将显存分配作为主导优化变量，基于以下观察与实践经验。在现代LLM推理框架（如vLLM）中，显存容量

直接决定了模型实例的最大可持续批处理大小,而批处理大小是影响推理吞吐率的关键因素之一。尽管计算、通信等其他资源也会产生影响,但在给定的硬件环境和模型架构下,通过显存分配来调整批处理规模,是控制各阶段相对速度最直接、最有效的手段。

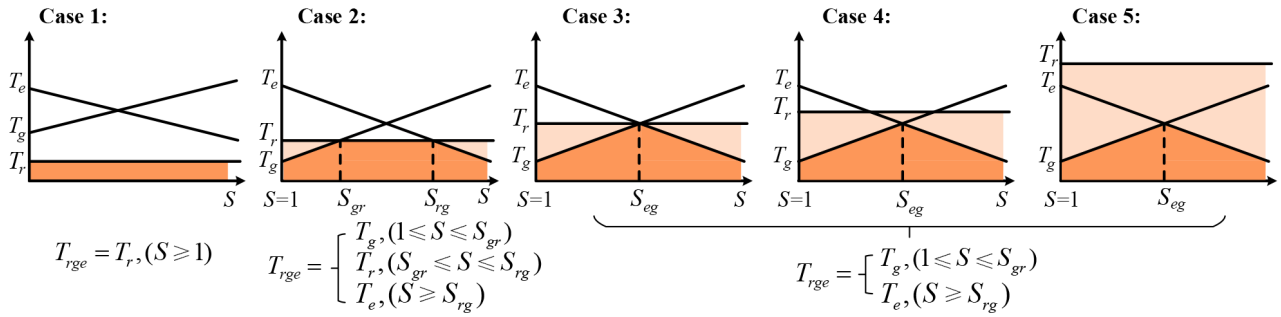
考虑一种最普遍的情况,即检索器主要使用CPU的算力,而生成器和评估器则使用GPU的算力。因此,为建立可指导工程调优的分析模型,本文明确以下关键参数:设M_g与M_e分别为分配给生成器和评估器模型的GPU显存。相应地,生成器和评估器的吞吐率可表示为所分配显存的函数,即$T_g(M_g)$与$T_e(M_e)$,系统整体吞吐率由最慢的阶段决定:

$$T_{rge}=\min\left(T_r,T_g(M_g),T_e(M_e)\right) \tag{3}$$

其中,T_r是检索器的吞吐率。当检索器经过优化(T_r足够高)后,系统瓶颈转移至生成与评估阶段。此时的优化问题可表述为:在满足总显存约束($M_g+M_e\leqslant M_{\text{total}}$)和最低显存需求($M_e>M_e^{\min}$且$M_g>M_g^{\min}$)下,寻找最优显存分配($M_g^*$, M_e^*)以最大化整体吞吐率:

$$\max_{M_g,M_e} T_{rge}=\max_{M_g,M_e}\min\left(T_g(M_g),T_e(M_e)\right) \tag{4}$$

假设生成器和评估器使用大小相似的LLM,由于生成器需要读取更长的文档数据,所以在给两者分配的硬件资源相似的情况下,T_g大于T_e是一个比较普遍的情况。在这种情形下,最直接的方式就是调整GPU的资源分配,让生成器使用更多GPU硬件资源,让评估器使用更少的GPU硬件资源。图5展示的就是上述策略的应用效果,其中显存分配比例$S=M_g/M_e$为本模型的核心控制变量。可以看到,根据T_r、T_g和T_e的大小关系,一共有5种情况。情况1事实上已经在第三章被讨论过了,即在这种情形下检索器作为系统的瓶颈而存在,这时应该先去优化检索器。作为一个RAG的开发者,假设系统的检索器已经经过了充分的优化,即T_r这条线的位置足够高,那么接下来只需要考虑情况3、情况4和情况5。此时,随着曲线向右延伸,T_{rge}将不断增加,直到达到最大值。换言之,T_{rge}的最大值就是$T_g(S)$和$T_e(S)$这两条线的交点。



注:当$S=S_{\max}$时,评估器模型使用的资源将维持在一个最低值,低于这个值时评估器将无法正常运行。

图5　RGE-Pipeline系统吞吐量关系的5种情况以及硬件资源分配策略对每种情况下可实现的最大吞吐量的影响

Figure 5　Five scenarios regarding throughput in the RGE-Pipeline system, and the impact of hardware resource allocation strategies on the maximum achievable throughput in each scenario

在情况3、情况4和情况5中,这种情形可以用数学方法表示为

$$\max\left(T_{rge}(S)\right)=T_{rge}(S_{eg}),\text{其中}\ T_e(S_{eg})=T_g(S_{eg}) \tag{5}$$

在T_{rge}达到这个最大值后,随着S继续增加,此时评估器由于资源过少、速度太慢,反而会成为系统新的瓶颈。但要注意,如图5所示,T_{rge}越过最大值后,S的值也不是可以无限大的。本文将$S_{\max}$定义为整个系统能够正常运行时生成器和评估器所使用的硬件资源的最大比例,超过这个比例时评估器将无法正常运作,即不满足上述约束条件中的$M_e>M_e^{\min}$(此时工程中一般遇到的报错是CUDA out of memory,即显存溢出)。

需要再次强调,$T_g(\cdot)$和$T_e(\cdot)$的具体函数形式复杂,受计算单元利用率、内核延迟、通信开销等多重动态因素的影响,难以预先精确建模。本文提出的吞吐率模型本质上是一个高度简化的工程指导模型,它并不追求精确预测绝对性能数值,而是旨在揭示性能与资源分配之间的关键趋势,为优化提供方向。因此,该模型的价值在于它清晰地揭示出以下几点:(1)系统存在一个使T_g与T_e平衡的最优显存分配点S_{opt};(2)可以通过基于监控的搜索策略(如图5所示,逐步调整S并观察T_g与T_e的变化)来逼近该最优配置,从而避免资源分配的盲目性;(3)在检索器的吞吐率经过充分优化(第三章已经实现)的基础上,一个可以极大地增加系统吞吐率的办法就是从给生成器和评估器分配相同比例的硬件计算资源开始,不断增加生成器模型的资源,减少评估器模型的资源,直

到两者的吞吐率相等，此时系统的吞吐率将达到一个极大值。

3.4 资源分配方案

基于上述建模分析过程，以及 vLLM 框架对多个 LLM 并行推理、单个 LLM 下的张量并行等能力，如图 6 所示，本文可以运用 3 种不同的方案将生成器和评估器所使用的两个 LLM 部署到若干个 GPU 设备上。方法 1 是将每张显卡的显存同时共享给生成器和评估器，简称为“显存共享”。这里利用了 vLLM 可以将模型推理的计算任务分配到不同 GPU 设备上的功能特性，但张量并行下需要对头数做均匀分配，也就是说，只有在注意力头数被显卡数整除的情况下，模型的推理计算才能正常执行（无法整除的情况下可以不用张量并行采用按层分割的方法，但是按层分割的方法下 GPU 的硬件利用率显然会大大降低）。方法 2 直接将若干张卡的其中几张分配给生成器使用，将剩余几张卡全部分配给评估器使用，此时不存在评估器和生成器共享显存的情形，简称为“整卡分配”。这里同样利用了 vLLM 张量并行的能力，但是没有方法 1 更精细利用显存的能力。

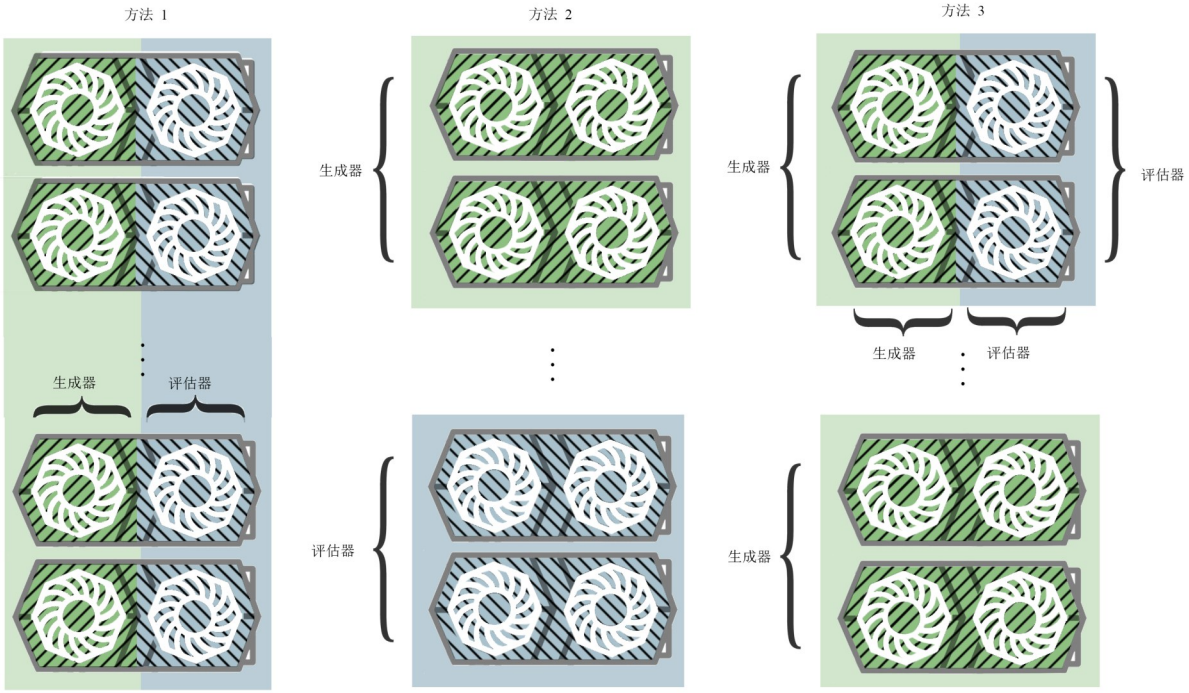


图 6 基于本文的建模分析提出 3 种分割 GPU 硬件资源的方法：显存共享、整卡分配和混合分割

Figure 6 Based on the modeling analysis presented in this paper, three methods for partitioning GPU hardware resources are proposed: shared VRAM, full-card allocation, and hybrid partitioning

在实际测试中，在资源分配比例相同的情况下，方法 2 对显存通信带宽的占用更小，并且不存在方法 1 的两个模型争用计算资源的问题，所以显然方法 2 下系统的计算速度不会低于方法 1。但方法 2 由于本身是一种粗粒度的资源分割方案，很难精细地分配显存以让 $S = S_{eg}$（如图 5 中 Case3~Case5，方法 2 下 S 的值的分布较为离散，此时直接有某个 $S = S_{eg}$ 的概率极低），从而让整体吞吐率达到最大值，所以本文又在方法 1 和方法 2 的基础上提出了“混合分割”作为方法 3。在方法 3 中，生成器在方法 2 的基础上，通过二次分享评估器的部分资源来进一步提高其吞吐率，即可融合方法 1“显存共享”中带有的精细分割能力，从而让系统的分割比例无限趋近 S_{eg}。

4 实验结果

4.1 实验设定

实验的主要目的是验证 RGE-Pipeline 系统在大容量知识库下对大批量用户查询的速度提升效果如何。与预实验相同，本章实验所使用的数据集也是 CRUD-RAG，服务器所使用的 CPU 也不变，但由于部署完整的 RGE-Pipeline 需要更多的 GPU 资源，本文在后续实验中都使用了 4 张 RTX 4090（24 GB 显存）来部署生成器模型和评估器模型。实验所采用的 vLLM 版本为 0.6.2，CUDA 版本为 12.9。

本实验中，生成器与评估器均统一采用 Qwen2.5-7B-BF16 模型，主要基于以下 3 点考量。首先，受实验硬件资源约束，在 4 张 RTX 4090（共计 96 GB 显存）的

条件下,7B量级的模型为对比“显存共享”“整卡分配”“混合分割”等策略预留了充足的显存调整空间。其次,在同期同规模的开源模型中,Qwen2.5-7B在推理稳定性与多项下游任务精度上表现突出,其作为实验基线更具可信度。最后,出于评估任务适配性,评估器需输出稳定、可控的判别结果,带有思维链(Chain-of-Thought,CoT)的模型不利于高效、可复现的自动化评估。相比之下,Qwen2.5-7B结构简洁,输出一致性好,更契合本场景的需求。

CRUD-RAG数据集包括4类任务:文本续写、摘要、问题解答和幻觉修改。其中,问答任务又分为3个子类别:单文档、双文档和三文档。每个任务的用户查询提问数量详见表1。本文从每个任务中随机抽取了共计6 400个查询(如表1所示)。实验重点关注速度和准确性两个方面。在速度方面,4.2节中测量了不同工作流、初始版RGE-pipeline处理所有6 400个查询(包括提取、生成和验证)所需的总时间,以及系统各个子模块都在计算时的整体吞吐率。注意,系统总延时并不简单地等于总任务数量除以吞吐率,还包括模型加载、通信开销、资源分配开销、评估结果输出的耗时等。在准确性方面,评估指标与原始CRUD-RAG论文[2]中使用的指标一致。

表1　用户查询提问数量　　单位:个

Table 1　Number of user inquiries number　unit: number

任务	文本续写	摘要生成	QA-单文档	QA-双文档	QA-三文档	幻觉修改
查询次数	1 999	1 580	800	797	797	427

4.2　RGE-Pipeline的加速效果呈现

在加速效果实验中,本文固定采用BM25S作为检索器所使用的稀疏检索算法,生成器和评估器都使用了Qwen2.5-7B-BF16。表2展示了本文提出的方法在整体提速方面的表现,一共统计了6种情况下完成全部6 400次查询计算所需的时间:原始工作流、生成与评估解耦的工作流、模型预加载工作流和RGE-Pipeline(包含3种分割方案)。

表2　RGE-Pipeline的加速效应呈现

Table 2　The acceleration effect of the RGE-Pipeline

	生成器批处理尺寸	评估器批处理尺寸	延迟时间/s	加速比(相对原始工作流)	加速比(相对于模型预加载工作流)
原始工作流	1	1	342 117.38	1×	—
生成与评估解耦的工作流	400	400	37 892.44	9.0×	—
模型预加载工作流	400	400	38 996.24	8.8×	1×
RGE-Pipeline 显存共享	400	300	9 870.68	34.6×	4.0×
RGE-Pipeline 整卡分配	400	200	5 199.65	65.8×	7.5×
RGE-Pipeline 混合分割	600	100	4 772.13	71.7×	8.2×

在原始工作流程中,由于存在大量的模型重复导入操作,完成全部6 400次查询需要95 h左右,这种耗时是完全可以避免的。采用生成与评估解耦的工作流或者模型预加载工作流来进行优化,这一任务仍然需要10 h以上才能完成,因此这也进一步凸显了采用RGE-Pipeline进行优化的必要性。表2也对比了在RGE-Pipeline系统中应用3种不同的分割GPU硬件资源的方法所达到的最高处理速度时的总体延时情况。可以得出结论,无论是哪种方法,相比于RGE-Pipeline系统构建之前的方法而言,系统均实现了显著提速。其中,混合分割方案由于结合了显存共享的资源分配灵活性以及整卡分配方案中尽量减少不必要通信开销的特性,达到了最优性能,相比原始工作流达到70倍以上加速比,相比模型预加载工作流则达到8倍以上加速比。

由此,本文可以得出结论,RGE-Pipeline方案本身已经能够有效加快基于LLM的RAG系统的评估工作,且本文基于vLLM推理框架所提出的一种创新的GPU硬件资源分割方案也确实能将系统的性能进一步推向峰值。

4.3　GPU硬件资源分配策略

在表2的基础上,表3更详细地给出了3种GPU资源分配方法下各自 T_{rge} 的具体数值,这些数据可以从3个维度展开分析。

第一,按照前述数学模型的指导,在方法1和方法2中,生成器模型和评估器模型都是从平均分配资源开始,然后逐步增加生成器所占显存比例,同步减少评估器所占资源比例。无论是方法1还是方法2,从 T_{rge} 的数值结果上可以看出,系统整体吞吐率都实现了增加,且两者占用比例也符合方案设计部分的预期。方法1中的比例可以精细调整,而方法2中只有4张显卡供使用,因此只能有(75%,25%)和(50%,50%)两种比例。这也印证了前文中对方法1和方法2的判断:前者灵活性较强,但性能不足;后者灵活性

差,但性能较强。

第二,在方法2“整卡”分配中,本文也尝试了增加生成器模型的个数(模型型号不变),并把它们部署到2张或3张不同的显卡上。在生成器模型个数为2、使用显卡张数为2(显存占用比例 = 2/4 = 50%)的情况下,本文让每个生成器模型各占用1张显卡。此时,系统性能相比于初始情况(1个生成器模型使用2张显卡)有显著提高,这也进一步证明了在硬件资源相同的情况下,模型使用的显卡数量越少,通信开销越小,速度越快。于是,方法2中也顺理成章地出现了一种最优解,即生成器模型个数为3、占用显存比例为75%(使用3张显卡),T_{rge}达到了1.90 query/s,远超方法1。

第三,方法3在方法2的最优解的基础上,再次运用了显存共享的原理,从而达到了本文实验中所有T_{rge}数值的最高点。由于方法3本身就是在方法2基础上的进一步提升,实验中直接从使用3张显卡运行3个生成器模型作为起点进行测试,然后不断地提升显存占比,直到达到最高吞吐率后,T_{rge}又出现下降趋势。这一点和方法1的T_{rge}数据的趋势一致,也证实了图5中描述的数学模型的预言:随着S数值的增加,T_{rge}将先增加到极大值,再减少。

表3 在3种GPU分配方法下RGE-Pipeline的加速效应呈现

Table 3 The acceleration effect of the RGE-Pipeline under three GPU resource allocation methods

	生成器占用显存比例/%	生成器模型个数	评估器占用显存比例/%	评估器模型个数	系统整体吞吐率/(query·s^{-1})	备注
方法1 显存共享	50	1	50	1	0.75	S = 1
	53	1	47	1	0.78	—
	56	1	44	1	0.79	$S = S_{eg}$方法1的最优分配方式,但如3.3节所论述,S_{eg}也不能简单地等同于两者显存占比
	59	1	41	1	0.46	—
	61	1	39	1	0.48	—
方法2 整卡分配	50	1	50	1	0.87	S = 1
	50	2	50	1	0.98	—
	75	1	25	1	—	由于vLLM机制问题(注意力头数无法被3整除),这种情况下,系统无法正常运行
	75	2	25	1	1.32	—
	75	3	25	1	1.90	方法2的最优吞吐率
方法3 混合分割	79	3	21	1	1.91	—
	80	3	20	1	2.18	方法3的最高吞吐率
	81	3	19	1	2.12	—

4.4 测试样例:不同LLM的精度结果

虽然精度结果的提升并不是本文所追求的研究目标,但基于本文的RGE-Pipeline提供了一些更换不同开源LLM生成器的测试数据,这得益于本文RGE-Pipeline的高效设计,本文的系统可快速更换模型进行测试。表4~表7呈现了这些算法的评估结果,除了BLUE-AVG和ROUGE-L以外,QA_avg_F1和QA_recall是用大模型作为评估器得到的精度测试结果,评估器仍然统一采用Qwen2.5-7B-BF16,这与前面的实验保持一致。注意,表格内所有指标都是数值越大表明算法效果越好。

从这些结果中可以看出,Qwen2.5-7B在参数量相近的开源大模型中表现相当突出,几乎在所有指标上都是最优解,这也是本章在前面关于速度和硬件资源分配的实验中使用这一模型的原因之一。同样,本文也推荐读者在构建自己的RAG系统时,可以优先考虑使用Qwen系列的大模型。表4~表7的结果表明,RGE-Pipeline框架具备兼容不同生成器模型的能力。然而,本文的核心工作与实验设计主要聚焦于生成器与评估器为同构模型的场景。在此设定下,本文得以清晰剥离并验证流水线架构与资源分配策略本身对系统吞吐率的优化效果。在生成器与评估器为异构模型(如不同尺寸或架构)时,对流水线平衡与资源调度产生的复杂影响,将是未来一个重要的研究方向。

4.5 补充实验:数据集大小与加速效应的关系

为全面验证RGE-Pipeline在不同评估场景下的鲁棒性与扩展性,本文补充了两组实验,分别探究知识库规模与查询集规模对系统加速效果的影响。

第一组实验旨在分析系统性能对知识库规模的

表4　生成器采用ChatGLM3-6B-32k的RAG算法评估结果

Table 4　Evaluation results of the RAG algorithm using the ChatGLM3-6B-32k model

	BLUE-AVG	BLUE-1	BLUE-2	BLUE-3	BLUE-4	ROUGE-L	QA_avg_F1	QA_recall	平均输出长度/tokens
文本续写	5.2	29.0	7.8	4.5	3.4	14.3	20.9	42.3	373.653
摘要生成	7.3	22.0	8.4	6.8	5.8	20.8	27.4	31.9	68.225
QA-单文档	3.5	10.3	4.1	3.1	2.5	10.2	21.2	30.2	376.392
QA-双文档	2.7	14.5	3.7	2.3	1.6	9.8	17.9	25.6	516.381
QA-三文档	3.6	17.5	5.0	3.0	2.0	10.6	19.1	31.9	332.985
幻觉修改	15.4	39.3	18.4	12.7	9.7	23.5	36.9	29.4	338.893

表5　生成器采用Qwen2.5-7B-BF16的RAG算法评估结果

Table 5　Evaluation results of the RAG algorithm using the Qwen2.5-7B-BF16 model

	BLUE-AVG	BLUE-1	BLUE-2	BLUE-3	BLUE-4	ROUGE-L	QA_avg_F1	QA_recall	平均输出长度/tokens
文本续写	5.2	31.7	7.9	4.1	2.9	18.9	22.5	52.4	326.356
摘要生成	41.5	64.1	47.1	38.7	32.4	60.9	58.1	80.3	42.487
QA-单文档	22.2	35.3	25.0	19.9	16.2	40.9	50.9	77.6	154.624
QA-双文档	17.3	35.8	20.2	14.2	10.8	35.3	46.6	83.7	277.567
QA-三文档	15.5	36.1	18.5	12.1	8.8	32.9	40.6	79.4	300.636
幻觉修改	21.7	47.3	24.1	16.8	12.7	34.7	53.5	68.1	209.570

表6　生成器采用DeepSeek-R1-Distill-Qwen-7B-BF16的RAG算法评估结果

Table 6　Evaluation results of the RAG algorithm using the DeepSeek-R1-Distill-Qwen-7B-BF16 model

	BLUE-AVG	BLUE-1	BLUE-2	BLUE-3	BLUE-4	ROUGE-L	QA_avg_F1	QA_recall	平均输出长度/tokens
文本续写	2.5	18.5	4.4	1.7	0.9	13.6	20.9	57.6	1 130.742
摘要生成	6.3	13.4	7.6	5.8	4.7	16.8	42.2	60.8	1 083.489
QA-单文档	3.3	9.7	4.4	2.8	2.0	11.5	30.8	46.0	709.366
QA-双文档	2.7	11.7	3.9	2.1	1.3	11.5	25.5	52.8	941.867
QA-三文档	2.7	12.8	4.0	2.0	1.2	12.1	25.6	55.0	970.162
幻觉修改	5.7	28.9	7.9	4.1	2.5	18.9	27.1	37.1	331.970

表7　生成器采用Qwen1.5-14B的RAG算法评估结果

Table 7　Evaluation results of the RAG algorithm using Qwen1.5-14B model

	BLUE-AVG	BLUE-1	BLUE-2	BLUE-3	BLUE-4	ROUGE-L	QA_avg_F1	QA_recall	平均输出长度/tokens
文本续写	1.4	20.5	3.4	1.0	0.4	14.7	18.7	52.1	534.228
摘要生成	6.9	21.7	10.0	6.0	3.9	26.2	45.7	73.6	123.020
QA-单文档	14.3	26.7	16.8	12.4	9.7	31.9	46.2	60.1	160.000
QA-双文档	11.6	32.1	14.6	8.9	6.1	29.0	38.9	75.9	256.984
QA-三文档	11.1	34.1	14.3	8.2	5.4	28.2	35.1	71.3	273.962
幻觉修改	11.1	31.0	13.1	8.0	5.5	26.1	48.7	66.8	368.080

敏感性。本文沿用第2.3节预实验的方法,从CRUD-RAG数据集中抽取了9个大小从18.2 KB~60.2 MB不等的知识库子集,并在RGE-Pipeline的3种资源配置方案下,测量处理全部6 400条查询的总耗时,结果如表8所示。分析表8数据可知,首先,RGE-Pipeline在所有知识库规模下均能实现显著加速,“混合分割”方案始终保持最优性能;其次,随着知识库规模增长,各方案的耗时增长曲线均较为平缓,尤其是“混合分割”方案的耗时最为稳定,这证明RGE-Pipeline的优化设计能有效应对知识库规模扩展带来的计算压力,具有良好的可扩展性。

第二组实验旨在验证RGE-Pipeline在较小规模

表8 文件大小与处理延迟的关系 单位:s

Table 8 Relationship between file size and latency unit: s

文件大小/KB	生成与评估解耦的工作流	显存共享	整卡分配	混合分割
18.2	23 123.33	7 565.14	4 641.20	4 312.72
160.6	31 572.43	9 109.78	4 922.14	4 421.98
1 016.4	33 767.29	9 338.28	5 002.85	4 592.81
2 998.8	34 012.66	9 472.15	5 092.78	4 643.90
6 055.5	34 282.45	9 445.32	5 034.73	4 662.52
10 022.9	36 441.67	9 717.72	5 087.31	4 679.32
20 013.7	37 892.32	9 816.68	5 092.25	4 672.83
30 053.8	37 429.17	9 827.71	5 102.84	4 724.02
60 211.3	38 092.31	9 856.41	5 172.76	4 775.39

查询集上的通用性。本文选取了RAGAS评估框架中使用的WikiEval数据集[28]进行测试。该数据集查询数量远少于CRUD-RAG,为本文观察系统在低负载、小批量场景下的行为提供了窗口。实验结果如表9所示。分析表9数据可知,即使在查询集规模较小的WikiEval数据集上,RGE-Pipeline依然能带来稳定的加速效果,“混合分割”方案同样表现最佳。然而,因为总查询量有限,流水线并行与批处理的优势未能得到充分发挥,所以绝对加速幅度不及在CRUD-RAG上的效果显著。这一结果与系统设计原理相符:充足的、持续的计算负载是维持流水线高吞吐率、最大化资源利用率的关键。

表9 RGE-Pipeline在WikiEval数据集上的测试结果

Table 9 Test results for RGE-Pipeline on the WikiEval dataset

	生成器批处理尺寸	评估器批处理尺寸	延迟时间/s	吞吐率/(query·s^{-1})
原始工作流	1	1	2 451.02	0.020 3
生成与评估解耦的工作流	20	20	373.13	0.134 0
模型预加载工作流	20	20	402.72	0.124 0
RGE-Pipeline显存共享	20	20	253.89	0.197 0
RGE-Pipeline整卡分配	20	20	168.22	0.297 0
RGE-Pipeline混合分割	20	20	143.87	0.348 0

综合两组实验的结论可知,RGE-Pipeline作为一个高效的评估系统,对于不同的知识库与查询集规模均具备良好的适应性与加速能力。特别是在企业级RAG系统评估的典型场景下,即知识库庞大、查询请求海量时,其流水线并行与精细化资源调度的优势能得到极致发挥,从而取得最为显著的加速效果。

5 结束语

为了解决基于大模型的RAG系统在实际应用场景中部署困难的问题,提高RAG的部署速度,本文提出了一种高效的RGE-Pipeline框架,用于加速基于LLM的RAG系统的验证过程。第4章的实验证明了该方法的有效性,可以将验证环节的速度提升数十倍(注意:第4章的实验中所有的加速实验对比都在使用BM25S作为稀疏检索器的前提下进行,因此这里的速度提升效果全部得益于RGE-Pipeline框架本身,并不涉及将BM25替换为BM25S这一操作所带来的增益)。这一框架的设计,包含了对原有RAG评估工作流的合理分解、流水线和并行化处理,也包括对整个系统的高层次抽象和数学建模,同时还提出了一种创新的GPU硬件资源混合分配策略。除了应用于RAG系统,该策略在其他涉及多个大模型并行推理计算的场景中也有应用潜力。文章作者将在未来工作中继续深入研究,以使其具备更好的泛化性,从而更灵活方便地服务于广大研究者和开发者。

为提升框架的完备性与实用性,本文计划未来从以下两个关键方向深化研究。一是异构模型场景下的动态调度。当前研究聚焦于同构模型的资源平衡,未来将探索生成器与评估器在模型架构、参数量级不同(即异构)时的动态资源分配策略,以应对更复杂的实际部署需求。二是面向流式查询的实时评估模式。当前实验基于批量查询,下一步将设计支持在线流式查询的低延迟实时评估机制,使RGE-Pipeline能够更好地适应需要即时反馈与持续监控的生产环境。未来,这些工作将推动RGE-Pipeline从一个高效的评估系统演进为一种更具通用性、适应性的多模型计算流水线框架。

参考文献

[1] Huang X J, Liu Z Y, Zhang M, et al. Towards a comprehensive understanding of the impact of large language models on natural language processing: Challenges, opportunities and future directions[J]. Scientia Sinica Informationis, 2023, 53(9): 1645.

[2] Lyu Y J, Li Z Y, Niu S M, et al. CRUD-RAG: A comprehensive Chinese benchmark for retrieval-augmented generation of large language models[J]. ACM Transactions on Information Systems, 2025, 43(2): 1-32.

[3] Chang S C, He T, Hu X K, et al. RAGChecker: A fine-grained framework for diagnosing retrieval-augmented generation[C]//Advances in Neural Information Processing Systems 37. Neural Information Processing Systems Foundation, Inc. (NeurIPS), 2024: 21999-22027.

[4] Şakar T, Emekci H. Maximizing RAG efficiency: A comparative analysis of RAG methods[J]. Natural Language Processing, 2025;31(1):1-25.

[5] Lu X H. BM25S: Orders of magnitude faster lexical search via eager sparse scoring[PP/OL]. V1.arXiv (2024-07-04)[2025-05-10]. https://doi.org/10.48550/arXiv.2407.03618.

[6] Kwon W, Li Z H, Zhuang S Y, et al. Efficient memory management for large language model serving with PagedAttention[C]//Proceedings of the 29th Symposium on Operating Systems Principles. New York: ACM, 2023: 611-626.

[7] Kaddour J, Harris J, Mozes M, et al. Challenges and applications of large language models[PP/OL]. V1.arXiv (2023-07-19)[2025-05-10]. https://doi.org/10.48550/arXiv.2307.10169.

[8] Minaee S, Mikolov T, Nikzad N, et al. Large language models: A survey[PP/OL]. V3. arXiv (2025-03-23) [2025-05-10]. https://doi.org/10.48550/arXiv.2402.06196.

[9] 许婷, 肖桐, 张圣林, 等. 基于LLM的日志故障诊断[J]. 电子学报, 2025, 53(4): 1123-1141.
Xu Ting, Xiao Tong, Zhang Shenlin, et al. Log fault diagnosis based on large language models[J]. Acta Electronica Sinica, 2025, 53(4): 1123-1141. (in Chinese)

[10] Radford A, Narasimhan K, Salimans T, et al. Improving language understanding by generative pre-training[R]. San Francisco: OpenAI, 2018.

[11] Radford A, Wu J, Child R, et al. Language models are unsupervised multitask learners[R]. San Francisco: OpenAI, 2019.

[12] Brown T B, Mann B, Ryder N, et al. Language models are few-shot learners[C]//Proceedings of the 34th International Conference on Neural Information Processing Systems. New York: ACM, 2020: 1877-1901.

[13] Achiam J, Adler S, Agarwal S, et al. GPT-4 Technical Report[R]. San Francisco: OpenAI, 2023.

[14] Guo D, Yang D, Zhang H, et al. DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning[J]. Nature, 2025, 645(8081): 633-638.

[15] 秦钰淑, 杨良怀, 朱艳超, 等. 融合图像与文本特征的组合检索方法[J]. 电子学报, 2025, 53(2): 558-567.
Qin Yushu, Yang Lianghuai, Zhu Yanchao, et al. A combined retrieval method by fusing image and text features[J]. Acta Electronica Sinica, 2025, 53(2): 558-567. (in Chinese)

[16] Gao Y F, Xiong Y, Gao X Y, et al. Retrieval-augmented generation for large language models: A survey[PP/OL]. V5.arXiv (2024-03-27)[2025-05-10]. https://doi.org/10.48550/arXiv.2312.10997.

[17] Hu Z B, Wang C, Shu Y F, et al. Prompt perturbation in retrieval-augmented generation based large language models[C]//Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining. New York: ACM, 2024: 1119-1130.

[18] Sparck Jones K. A statistical interpretation of term specificity and its application in retrieval[J]. Journal of Documentation, 1972, 28(1): 11-21.

[19] Robertson S, Zaragoza H. The probabilistic relevance framework: BM25 and beyond[J]. Foundations and Trends® in Information Retrieval, 2009, 3(4): 333-389.

[20] Mandikal P, Mooney R. Sparse meets dense: A hybrid approach to enhance scientific document retrieval[PP/OL]. V1. arXiv (2024-03-27)[2024-01-08].https://arXiv.org/abs/2401.04055.

[21] Lewis M, Liu Y H, Goyal N, et al. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension[C]//Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics. Stroudsburg: ACL, 2020: 7871-7880.

[22] Colin R, Noam S, Adam R, et al. Exploring the limits of transfer learning with a unified text-to-text transformer[J]. Journal of Machine Learning Research, 2020, 21(140): 1-67.

[23] Chen J W, Lin H Y, Han X P, et al. Benchmarking large language models in retrieval-augmented generation[J]. Proceedings of the AAAI Conference on Artificial Intelligence, 2024, 38(16): 17754-17762.

[24] Kwiatkowski T, Palomaki J, Redfield O, et al. Natural questions: A benchmark for question answering research[J]. Transactions of the Association for Computational Linguistics, 2019, 7: 453-466.

[25] Wang Z R, Yu Q H, Wei S D, et al. QAEncoder: Towards Aligned Representation Learning in Question Answering Systems[PP/OL]. V1.arXiv (2024-09-30) [2025-05-10]. https://arXiv.org/abs/2409.20434.

[26] Wang R B, Zhao Q F, Yan Y K, et al. DeepNote: Note-centric deep retrieval-augmented generation[PP/OL]. V2. arXiv (2025-04-07) [2025-05-10]. https://doi. org/10.48550/arXiv.2410.08821.

[27] Zhao J H, Ji Z Y, Feng Y C, et al. Meta-chunking: Learning text segmentation and semantic completion via logical perception[PP/OL]. V1. arXiv (2024-10-16) [2025-05-

10]. https://doi.org/10.48550/arXiv.2410.12788.
[28] Es S, James J, Anke L E, et al. Ragas: Automated evaluation of retrieval augmented generation[PP/OL]. V2. arXiv (2025-04-28)[2025-05-10]. https://arXiv.org/abs/2309.15217.

作者简介

路思远 男,1996年7月出生于江苏省无锡市。2018年本科毕业于南京大学通信工程专业,2025年博士毕业于南京大学电子科学与工程学院,现为南京风语智能信息技术有限公司总经理。主要研究方向为NLP大模型金融行业应用和高性能计算。
E-mail: sylu@smail.nju.edu.cn

叶尔潘·托合提亚尔 男,1998年4月出生于新疆维吾尔自治区伊犁哈萨克自治州伊宁市。现为南京大学电子科学与工程学院硕士研究生。主要研究方向为大语言模型的应用与优化。
E-mail: erpan.tohtiyar@smail.nju.edu.cn

朱菀晔 女,2001年6月出生于浙江省温州市。现为南京大学文物与博物馆专业硕士研究生。主要研究方向为博物馆领域的人机交互和人工智能技术应用。
E-mail: zhuyuyelh@163.com

施禹伯 男,1997年5月出生于辽宁省沈阳市。2019年毕业于南京大学微电子科学与工程专业。现为南京大学信息与通信工程专业博士研究生。主要研究方向为深度神经网络的高效鲁棒算法设计与实现、人工智能生成内容(AIGC)的算法与硬件高效协同设计。
E-mail: shiyubo@smail.nju.edu.cn

王美琪 女,1995年1月出生于山西省朔州市。现为中山大学集成电路学院副教授。主要研究方向为面向人工智能的高效算法、集成电路与智能系统设计、大模型/机器人应用优化加速等。
E-mail: wangmq53@mail.sysu.edu.cn

王中风 男,1963年10月出生于安徽省安庆市。国家高层次人才、IEEE Fellow和AAIA Fellow。现为南京大学电子科学与工程学院特聘教授、博士生导师,中山大学集成电路学院院长、教授、博士生导师。主要研究方向为现代纠错码设计与实现、高速有线和无线通信系统、深度学习算法及硬件加速等。
E-mail: zfwang@nju. edu. cn