Faculdade de Design, Tecnologia e Comunicação
Universidade Europeia

**2025**

**José Pedro Farinha Ribeiro**

# An Adaptive Query-Routing Framework for Optimizing Small Language Models in Resource-Constrained Environments

Faculdade de Design, Tecnologia e Comunicação
Universidade Europeia

**2025**

**José Pedro Farinha Ribeiro**

# An Adaptive Query-Routing Framework for Optimizing Small Language Models in Resource-Constrained Enviornments

Dissertação apresentada ao IADE - Faculdade de Design, Tecnologia e Comunicação da Universidade Europeia, para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Creative Computing and Artificial Intelligence realizada sob a orientação científica da Doutora Cláudia Sofia Sevivas Ribeiro, professora auxiliar da Universidade Europeia.

**Acknowledgements**

I want to express my deepest gratitude to my grandfather, Leonel Pires Ribeiro, for not only giving me a place to stay but also for helping with all the expenses. I cherished every conversation we had throughout most of college and deeply appreciate all the support he provided.

A sincere thanks to my advisor, Prof. Cláudia Sofia Sevivas Ribeiro, for the valuable insights shared throughout the thesis and for the many constructive discussions along the way.

A heartfelt thank you to my wonderful girlfriend, Bárbara Höller, for taking care of most of the household tasks while I focused on completing this thesis.

None of this work would have been possible without the support of my parents, Lídia Ribeiro and Nuno Ribeiro, who provided the resources needed to run all the required models and processes including the high electricity costs, kindly and unquestioningly covered by my grandmother, Maria Ribeiro.

And finally, the biggest thanks go to my other grandmother, Maria Lopes Marçal, for all the support she gave me throughout my academic journey and for all the boxes of vegetables which, according to her, gave me my beautiful eyes.

**palavras-chave**

AI; RAG; Quantization; HyDE; Large Language Models; Instruction Optimization.

**resumo**

À medida que os custos computacionais e financeiros dos modelos de linguagem de grande escala (LLMs) de última geração continuam a aumentar, a sua implementação torna-se mais difícil para organizações com recursos limitados. Uma vez que métodos de melhoria como a Retrieval Augmented Generation (RAG), Chain-of-Thought (CoT), HyDE e técnicas relacionadas melhoram a qualidade, mas implicam custos variáveis. Este trabalho apresenta uma estrutura dinâmica de encaminhamento de consultas, na qual um LLM compacto (8 mil milhões de parâmetros) é emparelhado com um controlador adaptativo que seleciona uma de três vias por consulta: resposta direta, CoT ou RAG. Para tal, o controlador baseia-se no refinamento iterativo de prompts, progredindo através de seis designs de instrução que evoluem de heurísticas baseadas em formato para uma classificação baseada em perfil, e emprega um pós-processador do tipo votação para garantir uma extração de decisão robusta. A estrutura proposta é avaliada em termos de precisão de encaminhamento, correção da resposta de ponta a ponta e perfil energético detalhado (CPU e GPU), utilizando um conjunto de dados compósito que combina conhecimento geral com forte dependência de recuperação de informação e questões de ciência focadas em raciocínio (itens ao estilo ARC-Easy e HotPotQA), num computador com uma única GPU. Os resultados mostram que os prompts baseados em perfil podem melhorar o equilíbrio do encaminhamento: as versões mais desenvolvidas atingem uma precisão de resposta superior a 85% em consultas do tipo ARC, mantendo-se muito mais eficientes em termos energéticos do que modelos maiores. Além disso, as análises demonstram que as respostas incorretas consomem mais energia e que o design da instrução transfere o consumo de energia entre a recuperação de informação (RAG), intensiva em CPU, e o raciocínio, intensivo em GPU. Consequentemente, os nossos resultados indicam que o controlo arquitetónico e a engenharia de prompts podem diminuir a diferença de desempenho entre modelos de pequena e média dimensão, enquanto alcançam ganhos de eficiência significativos e fornecem um caminho prático para sistemas de Recuperação de Informação (IR) e de Pergunta-Resposta

(QA) de alta qualidade, sob fortes restrições de recursos e requisitos de segurança de dados.

**Keywords** AI; RAG; Quantization; HyDE; Large Language Models; Instruction Optimization.

**abstract** As the computational and financial costs of state-of-the-art large language models (LLMs) continue to grow, deploying them becomes harder for resource-constrained organizations as improvement methods such as Retrieval-Augmented Generation (RAG), Chain-of-Thought (CoT), HyDE, and related techniques enhance quality but incur variable overheads. This work presents a dynamic query-routing framework, in which a compact LLM (8B parameters) is paired with an adaptive controller that selects from three routes per query: direct answer, CoT or, RAG. Therefore the controller builds on iterative prompt refinement, proceeding through six instruction designs that evolve from format-driven heuristics to profile-based classification, and employs a voting-style post-processor to ensure robust decision extraction. The proposed framework is evaluated on routing accuracy, end-to-end answer correctness, and detailed energy profiling (CPU and GPU) using a composite dataset that combines retrieval-heavy general-knowledge and reasoning-focused science questions (ARC-Easy and HotPotQA-style items), on a single-GPU workstation. Results show that profile-based prompts can improve routing balance: mature versions reach 85%+ answer accuracy on ARC-style queries while remaining much more energy efficient than larger models. Moreover, analyses show that incorrect answers consume more energy, and that instruction design shifts the energy burden between CPU-heavy retrieval and GPU-heavy reasoning. Consequently our results indicate that architectural control and prompt engineering can close the performance gap between small and mid-sized models while achieving significant efficiency gains and providing a practical path to high-quality IR and QA systems under tight resource constraints and data security requirements.

# Contents

# List of Figures

# List of Tables

# 3 Acknowledgments

I want to express my deepest gratitude to my grandfather, Leonel Pires Ribeiro, for not only giving me a place to stay but also helping with all the expenses. I appreciated every conversation throughout most of college and all the help he provided.

A great thanks to my advisor, Prof. Cláudia Sofia Sevivas Ribeiro, for all the great insights provided throughout the thesis and all the constructive discussions held along the way.

A heartfelt thanks to my beautiful girlfriend, Bárbara Höller, for taking care of most of the household tasks while I was focused on bringing this thesis to its end.

None of this work could have been done without the help of my parents, Lídia Ribeiro and Nuno Ribeiro, for all the resources provided to run the required models and processe including the expensive electricity bill, which was paid without hesitation by my grandmother, Maria Ribeiro.

The biggest thanks go to my other grandmother, Maria Lopes Marçal, for all the support she gave me throughout my entire studies, and for all the boxes of vegetables that give me my beautiful eyes according to her.

# List of Abbreviations

| | |
|---|---|
| **AF** | Adversarial Filtering |
| **AI** | Artificial Intelligence |
| **ARC** | AI2 Reasoning Challenge |
| **BART** | Bidirectional Auto-Regressive Transformer |
| **BBH** | Big-Bench-Hard |
| **BERT** | Bidirectional Encoder Representations from Transformers |
| **BM25** | Best Match 25 |
| **BPTT** | Backpropagation Through Time |
| **CAG** | Cache Augmented Generation |
| **CCPA** | California Consumer Privacy Act |
| **CoT** | Chain of Thought |
| **CoT+** | Chain of Thought plus |
| **DNNs** | Deep Neural Networks |
| **DPR** | Dense Passage Retriever |
| **EM** | Exact Match |
| **FFN** | Feed-Forward Network |
| **FLOPs** | Floating Point Operations per Second |
| **FP16** | 16-bit Floating Point |
| **FP32** | 32-bit Floating Point |
| **GDPR** | General Data Protection Regulation |
| **GPQA** | Graduate-Level Google-Proof Q&A |
| **GPT** | Generative Pre-trained Transformer |
| **GRU** | Gated Recurrent Unit |
| **GSM8K** | Grade School Math 8K |

| | |
|---|---|
| **HBM** | High Bandwidth Memory |
| **HotPotQA** | A dataset for question answering |
| **HyDE** | Hypothetical Document Embedding |
| **ICT** | Inverse Cloze Task |
| **IFEval** | Instruction-Following Evaluation |
| **INST** | Instruction |
| **INT4** | 4-bit Integer |
| **INT8** | 8-bit Integer |
| **IR** | Information Retrieval |
| **kNN** | k-Nearest-Neighbor |
| **KV** | Key-Value |
| **LC** | Long-Context |
| **LLM.int8** | A specific quantization method |
| **LLM-QAT** | Language Model - Quantization-Aware Training |
| **LLMs** | Large Language Models |
| **LoRC** | Low-Rank Compensation |
| **LSTM** | Long Short-Term Memory |
| **LUT-GEMM** | Look-Up Table based General Matrix-matrix Multiplication |
| **MIPS** | Maximum Inner Product Search |
| **MLM** | Masked Language Modeling |
| **MMLU** | Massive Multitask Language Understanding |
| **MoCo** | Momentum Contrast |
| **MUSR** | Multi-Step Reasoning |
| **NLI** | Natural Language Inference |
| **NLP** | Natural Language Processing |
| **NSP** | Next Sentence Prediction |

| | |
|---|---|
| **OBQ** | Optimal Brain Quantization |
| **OPTQ** | Optimal Quantization |
| **PCIe** | Peripheral Component Interconnect Express |
| **PGDSF** | Prefix-aware Greedy-Dual-Size-Frequency |
| **PIQA** | Physical Interaction: Question Answering |
| **PMI** | Pointwise Mutual Information |
| **PPL** | Perplexity |
| **PTQ** | Post-Training Quantization |
| **QAT** | Quantization-Aware Training |
| **QRA** | Question-Reference-Answer |
| **RAG** | Retrieval-Augmented Generation |
| **RNN** | Recurrent Neural Network |
| **RPTQ** | Reorder-based Post-training Quantization |
| **RTN** | Rounding to Nearest-Number |
| **RTRL** | Real-Time Recurrent Learning |
| **SKR** | Self-Knowledge Guided Retrieval |
| **SLMs** | Small Language Models |
| **SMEs** | Small to Medium Enterprises |
| **SMLs** | Small Language Models |
| **SpQR** | Sparse-Quantized Representation |
| **T-PTLMs** | Transformer-based Pre-trained Language Models |
| **ThrutfulQA** | TruthfulQA |
| **vLLM** | A high-throughput LLM serving engine |
| **WinoGrande** | Winograd Schema Challenge |

# 4 Introduction

Artificial Intelligence (AI) has revolutionized natural language processing (NLP) through the advent of Large Language Models (LLMs), which demonstrate exceptional capabilities in understanding and generating human-like language, with widespread applications across diverse industries. However, deploying these models in real-world, regulated environments presents substantial challenges.

Organizations like banks, hospitals, and government offices are likely to handle sensitive information that should not exit their premise under strict data privacy laws like GDPR and CCPA. Legal restrictions like these inhibit the use of AI models that are often hosted on external servers, where there is limited transparency in data processing operations. Furthermore, the computational demands of LLMs make them prohibitively expensive for small and medium-sized enterprises (SMEs), which often lack access to high-performance hardware infrastructure.

Despite advancements such as model quantization and the development of lightweight LLMs, a major gap still remains in effectively adapting these models to specialized, domain-specific tasks under limited computational resources. Methods like Retrieval-Augmented Generation (RAG) and Hypothetical Document Embedding (HyDE) have emerged as strong contenders in the sense that they can integrate external knowledge into the models with ease. However, success in such applications largely relies on the quality of query rewriting and retrieval.

This thesis focuses on retrieval-based question answering in such contrained domains, leveraging query rewriting to improve relevance and reduce computational overhead. This is achieved by at first optimizing Small Language Models (SLMs) through advanced quantization techniques, which significantly reduce their computational and memory footprint. However, this approach introduces a critical challenge, that is the degradation in model performance and knowledge retention. To counter this effect, the system integrates a powerful Retrieval-Augmented-Generation (RAG) framework. This framework is not merely just add-on for external knowledge but it serves as a targeted mechanism to recover most of the performance lost during the quantization stage. By leveraging instructions optimization, and Chain-of-Thought reasoning, the RAG component ensures that the quantized SLM can access and effectively utilize precise, relevant information from external document.

## 4.1 Motivation

Artificial Intelligence (AI) has made remarkable advancements in natural language processing (NLP) through the development of large language models (LLMs). Despite their capabilities, significant challenges remain in deploying these models in highly regulated and resource-constrained environments. Organizations such as banks and public institutions face significant barriers in adopting AI models due to strict data protection laws (eg., GDPR, CCPA) and high computational requirements. These constraint limit their ability to utilize externally hosted LLMs or fine-tune large models for domain-specific tasks. Additionally, the computational

demands of LLMs make them expensive to deploy, requiring powerful hardware infrastructure that is often unaffordable for small to medium enterprises (SMEs). While advancements in quantization techniques and lightweight models have made LLMs more accessible, there is still a gap in optimizing these models for domain-specific tasks without extensive computational resources. Retrieval-Augmented Generation (RAG) and HyDE methods have emerged as promising solutions, enabling models to integrate external knowledge efficiently. However, their effectiveness depends on the quality of query rewriting and retrieval mechanisms. This thesis seeks to address these challenges by evaluating lightweight, domain-aware strategies for query rewriting within a RAG framework. By leveraging techniques such as query augmentation, voting system, Retrieval Augmented Generation and Chain-of-Thought reasoning, the goal is to improve retrieval relevance and performance in regulated and resource-constrained settings. This work will also explore the trade-offs between large quantized models and small non-quantized models, providing practical insights for deploying AI systems in real-world applications.

## 4.2 Aims and Research Questions

This thesis aims to optimize small language models (SLMs), explore the trade-offs between quantized and non-quantized models, investigate retrieval methods to enhance SLM performance, and outline directions for future research. SLM optimization will be approached by exploring recent techniques identified throughout the course of this work, with the goal of making these models more practical and resource-efficient. Analyzing the trade-offs between quantized and non-quantized models is an important step, as it will help determine whether future research should focus on quantizing larger models or on further refining smaller ones.

Retrieval methods are a good way to achieve better performance on SMLs on tasks that normally would require training with more specific data-sets. Selecting an effective retrieval strategy is key to developing a lightweight, high-performing system.

Model optimization may also include simple strategies such as query injection. These approaches will be evaluated to determine their usefulness and relevance to the overall objectives of the thesis.

This research aims to answer the following key questions:

RQ1: How can a system using smaller models still compete with larger ones in terms of performance and efficiency?

RQ2:Can HyDE be applied to a system designed for efficiency?

RQ3: What are the trade-offs among answer quality, inference latency, and energy consumption for each rewriting strategy?

RQ4: Can an efficient approach still achieve high accuracy while remaining useful?

## 4.3 Document Outline

1. Introduction
    - Sets the stage by explaining the motivation, challenges, and research questions.
2. Background and State-of-the-Art
    - Reviews existing techniques and it's limitations.
3. Methodology
    - Details the approaches used to optimize SLMs and enhance retrieval.
4. Evaluation Framework
    - Explains how the methods are assessed using different metrics.
5. Results and Discussion
    - Analyzes the outcomes and trade-offs of the proposed methods.
6. Conclusion and Future Work
    - Summarizes the contributions and suggests future work.

# 5 State-of-the-Art

## 5.1 Transformers and Language Models

Natural language processing (NLP) has been improving significantly over the last decades due in part to the resurgence of deep neural networks (DNNs) [1]. The first sequential architecture, RNN [2], had limitations regarding it's ability to capture temporal dependencies. This limitation is related to the vanishing or exploding gradient, which results in the impossibility for RNN to retain information over longer sequences. The longer the sequence, the more the gradients would diminish to near zero or infinity, resulting in less relevant weight updates. LSTM [3] and GRU [4] are two sequential models developed to overcome this limitation. LSTM was the first to use an algorithm to consider gradient-based learning, which could bridge time intervals in excess of 1000 steps. This was achieved using memory cells and gating mechanisms (input, forget and output). This allowed the networks to retain, update, or forget information in the memory cell, avoiding the vanishing/exploding gradient. On the other hand GRU, doesn't have a separate memory cell; instead, it directly updates the hidden state using two gates: an update gate that combines the forget and input gates of the LSTM model and a reset gate that gives the network the ability to control how much information it forgets.

| Layer Type | Complexity per Layer | Sequential Operations | Maximum Path Length |
|---|---|---|---|
| Self-Attention | $O(n^2 \cdot d)$ | $O(1)$ | $O(1)$ |
| Recurrent | $O(n \cdot d^2)$ | $O(n)$ | $O(n)$ |
| Convolutional | $O(k \cdot n \cdot d^2)$ | $O(1)$ | $O(log_k(n))$ |
| Self-Attention (restricted) | $O(r \cdot n \cdot d)$ | $O(1)$ | $O(n/r)$ |

Table 2: Maximum path lengths, per-layer complexity and minimum number of sequential operations for different layer types. [5]

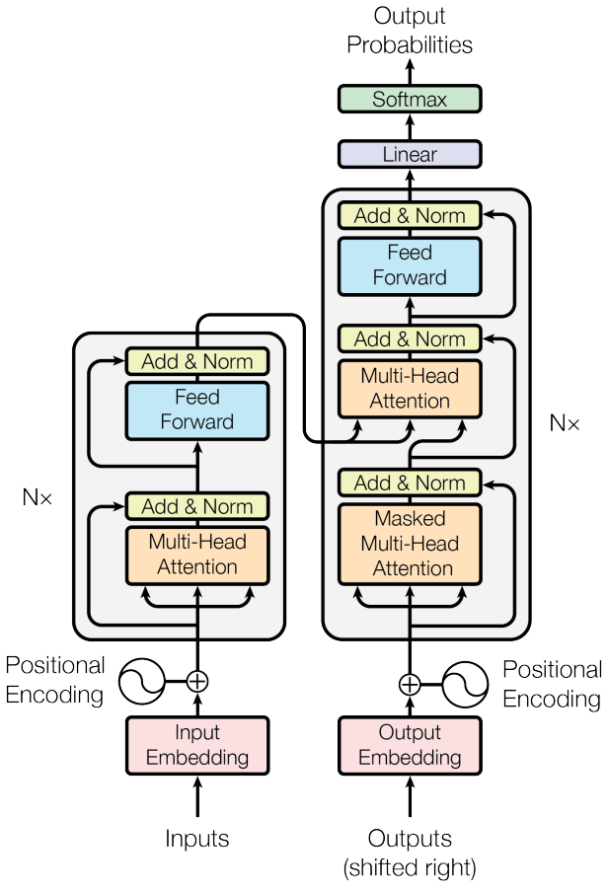



Figure 1: Transformer model architecture [5]

Transformer models introduced many changes to try to solve all of the problems in the previous models. Starting with the Self-attention Mechanism which lowers the complexity for short sequences. This is due to the fact that the computational complexity is $O(n^2 * d)$ which is more efficient than Recurrent layers $O(n * d^2)$ when the sequence $n$ is smaller than the dimensionality $d$. When parallelization is used it requires only $O(1)$ sequential operations but on the recurrent layers they need $O(n)$ sequential operations due to their sequential nature. Due to the short path length between any two positions in the input and output sequences the signals can travel between all pairs of positions, this proved to be a big improvement for leaning long-range dependencies. Moreover which can be useful to improve the global context since it can attend to all positions in the sequence at once. Transformers self-attention can dynamically

adjust based on the input sequence, this can also be modified to make the model focus on just the local context.

The ability to interpret attention mechanisms is a significant advantage for understanding how the model works. Papers such as [6] investigate what the model focuses on, which in turn provides insights into its decision-making process.

Multi-Head Attention this mechanism instead of performing a single attention function with $d_{model}$-dimensional keys, values and queries they linearly project the queries, keys and values $h$ times with different learned linear projections to $d_k(Queries), d_k(Keys)$ and $d_v(Values)$ dimensions. The queries represent the search The way Multi-Head Attention computes attention independently for each head allows the model to focus on different types of relationships and features in parallel capturing more information, this also makes it possible to capture information that with a single head would normally be lost like complex relationships [7].

Multi-Head Attention extends the standard attention mechanism by applying multiple attention functions in parallel. Instead of performing a single attention operation with $d_{model}$ dimensional keys, values, and queries, the mechanism linearly projects the queries, keys, and values h times using different learned linear transformations into dimensions $d_k(Queries), d_k(Keys)$ and $d_v(Values)$. These projections allow each head to compute attention independently, enabling the model to focus on different types of relationships and features simultaneously. This parallelization captures a richer set of dependencies and patterns, including complex relationships that might otherwise be lost with a single attention head [7].

Because each head has smaller subspaces instead of one high-dimensional space this reduces the computational complexity and improves the optimization process by scaling the dot products to prevent their values from becoming too large.

In natural languages, word order plays a crucial role in conveying meaning and context. Therefore, it is essential for models to incorporate positional information. Unlike recurrent or convolutional models, the Transformer does not have an inherent mechanism to capture token order. To address this, positional encoding is introduced to inject information about the relative or absolute position of tokens within the input sequence. To make sure the the weight adapts to the size of the query it uses the wave length of $\sin$ and $\cos$ that increases exponentially with the dimension of the index $i$, this grants that the positional encoding adjusts to compensate for any dimension length:

$$PE(\text{pos}, 2i) = \sin\left(\frac{\text{pos}}{10000^{\frac{2i}{d_{\text{model}}}}}\right) \quad [5]$$

$$PE(\text{pos}, 2i+1) = \cos\left(\frac{\text{pos}}{10000^{\frac{2i}{d_{\text{model}}}}}\right) \quad [5]$$

As a benefit to how the model treats the positional encoding, it is able to adapt to sequences longer than the ones found in training.

Feed-Forward network consists of two linear transformations with a ReLU activation be-

tween them:

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2 \quad [5]$$

ReLU or rectified linear unit is used to add non-linearity to the network allowing it to learn more complex functions. The non linearity comes from the activation function $max(0, x)$, which when the input $x$ is positive it passes through without any change, on the other hand when the input is negative it outputs $0$. With linear transformations being the same across different positions, they use different parameters from layer to layer, for example if the input and output dimensionality is set to 512 and the inner-layer for 2048 this means that $w_1$ will transform from 512 to 2048 and then $w_2$ will transform the 2048 into 512. "Another way of describing this is as two convolutions with kernel size 1."[5].

The evolution of the Transformer models came with GPT and BERT [8]. GPT and BERT were the first T-PTLMs developed based on transformer decoder and encoder layers respectively, these models where the basis for the discovery that performance of T-PTLMs could be increased just by increasing the size of the model which triggered the development of models like GPT-3 (175B)[9], PANGU- (200B)[10] and even a model with 1.6 trillions of tokens named Switch-Transformers [11]. Although performance is not strictly linear and depends on many factors, the number of tokens plays a significant role. This was only made possible in part due to the parallelization ability of the Transformer model. Bert[12] differs from the original Transformer model due to its bidirectional architecture contrary to the original model which was unidirectional. Bert uses its bidirectional architecture (left-to-right, right-to-left) to access context from both directions simultaneously.

Another important feature introduced during pretraining is Masked Language Modeling (MLM), in which $15\%$ of the tokens in the input are randomly masked and the model learns to predict them using bidirectional context. Additionally, Next Sentence Prediction (NSP) is used to train the model to determine whether two sentences logically follow each other.

Another improvement to the model was the input representation, BERT adds special tokens [CLS] and [SEP], these are used by the model in conjunction with segment embeddings to handle sentence pairs, this allowed BERT to handle both single sentence and sentence pair tasks. GPT differs significantly in its training approach by combining both supervised and unsupervised learning. In the unsupervised phase, the model is trained on a large text corpus (e.g., BooksCorpus) using standard language modeling. This is followed by a fine-tuning phase using supervised learning on specific downstream tasks. This two-stage process allows the model to first learn general language patterns and then adapt to more specialized tasks. GPT also uses only a Transformer decoder architecture, consisting of 12 layers of masked self-attention. Because it relies solely on a decoder, the model can only attend to previous tokens in the sequence, making it well-suited for auto-regressive language modeling. This model construction makes it suitable to need minimal architecture changes when adapting to different tasks.

## 5.2 Quantization

Quantization is one of the most important techniques used to improve the performance and efficiency of large language models (LLMs), which are increasingly applied across a wide range of domains from customer service to scientific research. However, as models grow in size and computational demands increase, a major challenge arises: the hardware required to run these models becomes a limiting factor. High-performance hardware can be extremely expensive and energy-intensive. While recent advancements in hardware have enabled the development of more powerful AI models, the cost and accessibility of such infrastructure remain significant barriers to widespread adoption. To give an example, Microsoft's Phi-3-mini-4k-instruct model [13] requires 512 H100-80G GPUs to be run consecutively for 10 days with each costing around €30,000. Although such needs refer to training, which is done only once, running this kind of model still proves to be a computationally heavy task.

The most frequently used method to improve on this challenge is quantization, which reduces the computational and memory requirements of the machine learning model. It achieves this by converting the model weights and, in some cases, the activations from high-precision 32FP to a lower precision representation such as INT8. This reduces the memory consumption of the model on the GPU and can accelerate computation because integer operations consume fewer resources compared to floating-point operations.

The two major methods of quantization will be discussed in more detail in subsequent sections: Post-Training and Quantization-Aware Training.

### 5.2.1 Post-Training Quantization (PTQ)

TThis is the most commonly used quantization method because it does not require access to the model's training process. The quantization is applied after the model has already been trained, making it especially useful when training resources or data are unavailable. This type of quantization has been proven not to be as accurate at lower bit levels and there is a tendency of degradation if quantized to lower than 8-bits.[14] However since this method is less resource intensive, it has attracted more attention with a remarkable surge in post-training quantization methods in the recent years.

The simplest approach is also the least efficient, as it directly quantizes 16-bit values to 8-bit using row-wise symmetric quantization. While this method is straightforward, it typically results in only negligible degradation in perplexity. However, this breaks down with 4-bit quantization as it witnesses a significant drop in perplexity [15]. To improve the quantization performance for low-bit applications, ZeroQuant-V2 [15] proposed Low-Rank Compensation (LoRC) method. This method approximates the error $E$ between the original weight matrix $W$ and the quantized weight matrix $\hat{W}$ using storage-efficient low-rank matrix $\tilde{E}$ so that $\hat{W} + \tilde{E}$ would be a better approximation of the original weight $W$[16].

Later research by LUT-GEMM[17] and SqueezeLLM[18] showed that non-uniform weight

distributions could achieve even lower bit-width quantization. This is due to the weight distribution after training being nonuniform so it makes sense for the weight distribution not being quantized uniformly. This is done by allowing the quantization process to allocate more precision to the ranges of weights that are more densely populated while leaving larger intervals for less frequent weight ranges. Building upon these methodologies OPTQ[19] emerged as an advancement in quantization for big LLM's. Making it possible to run OPT-175B on just a single Nvidia A100 GPU or only two of the more cost-effective A6000. OPTQ also provided greater results in the extreme quantization regime where models were quantized all the way down to 2 bits, or even ternary values. The OPTQ [20] algorithm improved on Arbitrary Order Insight, prior to this method the norm was to quantize the weights in a greedy order [21] this means that the weight picked for the next quantization was picked based on minimum quantization error, this performs well but compared to arbitrary order quantization only offered a negligible improvement in large, heavily-parameterized layers. The likely reasons for the lack of improvement were that large individual quantization errors were balanced out overall, and that these errors occurred later in the quantization process when fewer weights remained to be quantized, leaving less opportunity for adjustment. But with the OPTQ approach instead of quantizing the weights row-by-row, this method aimed to quantize the weights in all rows simultaneously and in the same order. This can be shown by how the unquantized weights $F$ and the inverse layer hessian $(H_F^{-1})$ depend only on the input activations $(X_F)$ and not on the weights themselves, this proves that the quantization of a column affected all rows uniformly. Columns within blocks are quantized recursively and at each step, unquantized weights are updated based on the quantized weights.

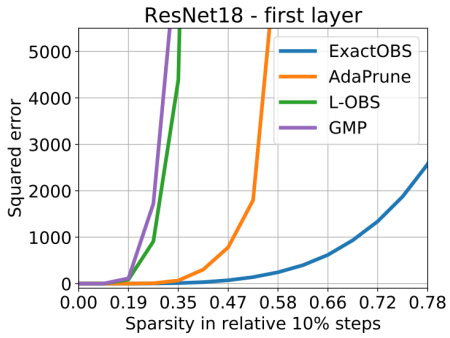



Figure 2: RN18 squared error. [21]

With that it also achieved efficiency gains compared to OBQ which achieved $O(d_{\text{row}} \cdot d_{\text{col}}^3)$ comparing it to OPTQ which achieves $O(\max\{d_{\text{row}} \cdot d_{\text{col}}^2, d_{\text{col}}^3\})$, reducing it by a factor of $\{d_{\text{row}}, d_{\text{col}}\}$. For larger models this can be proven to be more efficient into several orders of magnitude. See Table 3 for details on runtime analysis.

The second step involves the use of lazy batch updates, which were introduced to improve upon the original Optimal Brain Quantization (OBQ) method. In the original approach, weights were iteratively quantized, requiring updates to all elements of a potentially large matrix while

| Parameter | Value |
|---|---|
| $d_{\text{row}}$ | 10000 |
| $d_{\text{col}}$ | 100 |

| Runtime Type | Calculation |
|---|---|
| OBQ Runtime | $O(10000 \cdot 100^3) = O(10,000 \cdot 1,000,000) = O(10^{10})$ |
| OPTQ Runtime | Max Term: $O(\max\{10000 \cdot 100^2, 100^3\})$ |
| | Calculations: $10000 \cdot 100^2 = 10,000 \cdot 10,000 = 10^8$ |
| | $100^3 = 10^6$ |
| | Result: $O(\max\{10^8, 10^6\}) = O(10^8)$ |

Table 3: Runtime Analysis

using only a few FLOPs per entry[21]. This means that the GPU usage was limited by the speed of the memory bandwidth. Lazy Batch-Updates addressed this issue by grouping updates across $B \times B$ blocks of the inverse Hessian matrix $H^{-1}$, with $B$ being typically set to 128 columns at a time. These blocks are processed and after that, they are used to apply updates to the matrix. Thus avoiding the need for frequent recalculations throughout the matrix, cutting on memory-bound operations.

The final modification was the Cholesky Reformulation that was used to improve on two key issues, numerical inaccuracies and error accumulation, the numerical inaccuracies occur as model sizes increase beyond a few billion parameters, these can lead to instability. This happens because the matrix $H_F^{-1}$ becomes indefinite during iterative updates causing erratic weight updates, the error accumulation is caused by the compounding numerical errors of the matrix inversion. Though previously they used dampening techniques like adding a small constant $\lambda$ (which was normally always $1\%$ of the average diagonal value) to the diagonal elements of $H$. However, this proved to be inefficient on larger models, but by combining Cholesky decomposition, precomputation of necessary rows, and dampening prevents the $H_F^{-1}$ from becoming indefinite mitigating the accumulation of numerical errors, and makes the algorithm suitable for large models. A deeper dive into some of these methods are described in the subsequent sections.

### 5.2.2 Weight-Only Quantization

The Weight-Only Quantization focuses on quantizing only the weights of the LLM and not the activations, this reduces the model size and memory transfer time. However, this doesn't benefit from hardware-accerlerated low-bit operations.[16] The most used quantization method

is rounding to nearest-nember (**RTN**).[22], this works by quantizing a tensor $x$ into $k$-bits.

$$Q[x] = s \times \text{clamp}\left(\frac{x}{s}, l_{\text{min}}, l_{\text{max}}\right) \quad [22]$$

Here the $s$ is the quantization scale, $l_{\text{min}}$ and $l_{\text{max}}$ are the low and upper bound clipping, which is then rounded to the nearest number $\lfloor \cdot \rfloor$. Usually setting $l_{\text{min}} = -2^{k-1} + 1$ and $l_{\text{max}} = 2^k - 1$ and set $s$ to be the maximum absolute value in $x$. There are two main ways to find the best configuration in weight only quantization. The first one is by minimizing the reconstruction error of the weight parameter which is define as

$$r(W) := \|Q[W] - W\|^2 \quad [22]$$

On the function above only the weights are accessed therefore it's a data-free process. However recent studies ([19], [23]) propose the usage of output error as compensation.

$$e(W) = \sum_{X \in \mathcal{D}} \|Q[W]X - WX\|^2 \quad [22]$$

$D$ corresponds to the calibration set sampled form of the original training data, for optimization.

By regularizing the model with its training data, more promising results are achieved compared to the reconstruction-based method. The data requirements of these 2 methods have a two big draw backs, the first one being that there's a requirement of having the original training data of the model since most quantizations are done by others than the creators off the model makes it hard to find exactly which data was used to train the model. The second problem is that using the same data again can jeopardize the ability of generalization of the model due to the model over-fitting to the calibration set. For this two reasons it is clearly very important to achieve a Data-free quantization.

### 5.2.3 Non Uniform Weight Quantization

Most of the typical quantization methods handle the weights differently from this approach. This method quantizes weights differently depending on their importance to the LLM. Not all weights are of equal importance to the performance of a model, and so they should not be treated similarly, which is the case with techniques such as EasyQuant.

EasyQuant[22] proposes the usage of the reconstruction error as the regulation metric since this can be used to optimize the quantized model indirectly improving the generalization ability of the model. According to [22] the performance gap of the quantized model (INT4) and the full precision model is due to two main factors. The first being that normally the quantization range is picked as the maximum absolute value of the weight thus inducing a large reconstruction error for low-bits quantization. The latter refers to the fact that the $0.1\%$ of weights corresponds to outliers, although representing a small percentage of the total, have a significant impact on the

model's performance.Keeping this in mind, if we try to define the outliers using the condition:

$$|W_{i,j} - \text{mean}(W)| \geq n \cdot \text{var}(W) \quad [22]$$

For any weight $W$, where $W_{ij}$ is the $(i, j)$-th weight and $(n)$ representing the threshold for identifying the outliers, we can classify certain weights as outliers [22]. However, the challenge is that simply detecting the outliers and avoiding their quantization is not sufficient to achieve good model performance. Furthermore, if the percentage of outliers becomes too large, the overhead introduced by the dequantization kernel increases, which can lead to a reduction in overall throughput.

| reconstruction error | int4 outlier | fp16 outlier |
|---|---|---|
| 4.8E4 | 12.65 | 12.50 |
| 3.5E4 | 14.73 | 11.61 |
| 2.7E4 | 19.71 | 11.25 |
| 2.3E4 | NA | 11.10 |
| 1.9E4 | NA | 11.02 |

Table 4: PPL results on Wikitext2 of BLOOM-7B with and without outlier isolation. [22]

EasyQuant also experimented with an ablation study focusing on three aspects, the outlier influence, outlier distribution, and the Quantization Range. The ablation study began by preserving $10\%$ of the weights in FP16. This resulted in an $8\%$ increase in perplexity, compared to only a $1\%$ increase achieved with EasyQuant. These findings suggest that simply isolating the outliers was not sufficient to maintain the expected perplexity levels. To check the outlier influence on EasyQUant, outlier isolation is key however this can only impose an indirect influence on the model accuracy. The phenomenon found is the outliers behave like a gating mechanism meaning that without the outlier isolation, the model performance deteriorates significantly with smaller reconstruction error, and with outliers in FP16 the model shows continuous improvement decreasing the perplexity with smaller reconstruction error Table 4. Another study was done to understand how much influence outliers with big weight magnitude and small weight magnitude have on the model performance, this was done by pruning $1\%$ of the values (according to their magnitude) in the weights into $0$ and see the perplexity results.

| pruned weights | PPL |
|---|---|
| smallest (top-0% 1%) | 11.66 |
| median (top-49% 50%) | 19.16 |
| largest (top-99% 100%) | 19.17 |

Table 5: PPL results after pruning $1\%$ weight with different magnitude [22]

Based on Table 5 [22] shows that the largest magnitude outliers imposed the same influence on the model performance as the normal values. This suggests that outliers exert a similar direct

influence on model accuracy as regular weights, thereby indicating that isolating outliers has an important indirect impact on the overall performance of the model.

| module name | outlier fraction (%) |
|---|---|
| Att.qkv | 0.2993 |
| Att.output | 0.5036 |
| FFN.1 | 0.288 |
| FFN.2 | 0.7560 |

Table 6: Outlier fraction distribution in different modules in BLOOM-7B under 3-sigma threshold [22]

| Layer index | outlier fraction (%) |
|---|---|
| 1 | 0.3187 |
| 5 | 0.8579 |
| 10 | 0.3953 |
| 15 | 0.3975 |
| 20 | 0.3962 |
| 25 | 0.4399 |
| 30 | 0.3954 |

Table 7: Outlier fraction distribution in different layer index in BLOOM-7B under 3-sigma threshold [22]

On the outlier distribution, it was explored the distribution along different modules and layers, and it showed that the fraction of the outliers share different patterns in different modules and layers, refer to Tables 6 and 7.

Another characteristic found was that the FFN.2 module showed a significantly higher fraction of outliers, however there was no pattern along the layer index.

On the quantization range, it was observed that the dynamic quantization range of different optimization steps and concluded that the range decreased fast in the early stages of training meaning a smaller quantization range facilitating more precise quantization of parameters. The study also revealed that after a certain number of steps, the quantization range became stable meaning that the optimal range had already been achieved. In deep neural networks, not all weights have the same influence on the model's performance some contribute more significantly than others [22]. This implies that relying solely on the magnitude of the weights is insufficient to fully capture the impact of each element on the model's behavior. A good benchmark to detect parameter sensitivity is the Hessian metric. This occurs due to the fact of the Hessian matrix being leveraged to assess the salience of parameters in each under-binarized layer. The optimized computation process to derive weight sensitivity is given by:

$$s_i = \frac{w_i^2}{[H^{-1}]_{ii}^2} \quad [24]$$

The $H$ represents the Hessian matrix of each layer and the $w_i$ which represents the original value of each element. The $s_i$ is then used as a criterion for assessing the weight significance of the element also used as a feature indicator for a structured selection.

Structural search selection can be implemented using unstructured selection, allowing the model to cover all salient weights. However, this approach requires an additional 1-bit bitmap index [25], which increases the average bit-width. This proves to be inefficient, especially for the Hessian outlier weights that are only less than $1\%$ of the total. According to [24] the majority of the weights that are sensitive Hessian values are predominantly concentrated in specific columns or rows.

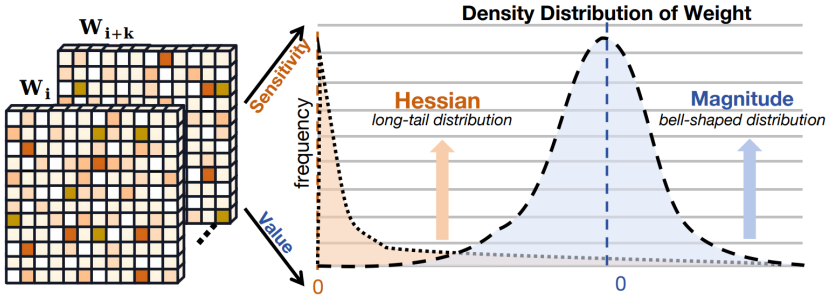



Figure 3: The Hessian metrics (sensitivity) and magnitude (value) of weights in LLMs. The weights of different layers in LLMs are characterized by bell-shaped distribution, accompanied by a few salient values.[24]

This pattern is due to the convergence effects inherent in multi-head self-attention mechanism of the models, thus needing a structured approach to select salient weights, reducing the additional bit-map. The approach described in [24] is to employ a per-channel or per row type of binarization, they determine salience through a per-column segmentation on the whole matrix.

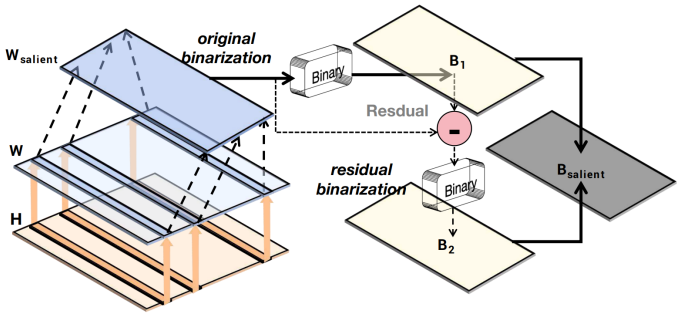



Figure 4: Illustration of salient weight binarization. The B1 binarized from salient weight is made into a residual with the original value and then binarized again to obtain B2.[24]

The main idea is to rank the columns by their salience in descending order and use an optimized search algorithm to minimize quantization error. This process determines the optimal number of columns to include in the salient group. Based on this formula:

$$W_b = \alpha \cdot \text{sign}(W_f), \quad [24]$$

where $W_b$ corresponds to the binarized output, $\alpha$ denotes the scaling factor and $W_f$ denotes the weights at full precision (FP16). This was then used to define the objective of the binarization quantization, used in this equation:

$$\arg\min_{\alpha,B} \|W - \alpha B\|^2, \quad [24] \tag{1}$$

where the $B$ is the number of selected columns, $\alpha$ and $B$ can simply be solved as $\alpha = \frac{\|W\|_{\ell_1}}{n \times k}$ and $B = \text{sign}(W)$. Then the optimization function to select salient columns is defined as:

$$\arg\min_{W_{\text{uns}}} \|W - (\alpha_{\text{sal}}\, \text{sign}(W_{\text{sal}}) \cup \alpha_{\text{uns}}\, \text{sign}(W_{\text{uns}}))\|^2, \quad [24]$$

where $W_{sal}$ denotes the column-wise combination of the original weight and $W_{uns}$ is the left non-salient part. $W$ can be determined by $W_{sal} \cup W_{uns}$ so the only variable parameter is the number of rows in $W_{sal}$.

Binary Residual approximation is a technique use to address the challenge of preserving salient weights which are limited in quantity, but exhibit significant variance when aggregated. If these weights are preserved at their original formats FP16 or INT8 it increased the average weight bit-width, reducing the compression benefits of binarization. However traditional methods of binarization result in a substantial quantization errors. Contrary to the comprehensive high-order quantization [26] which also applies quantization to the entire weight matrix, the approach described in [24] uses a residual approximation method. This approach focuses on binarizing only a subset of salient weights minimizing the error through a second-order aproximation. This method grants the precision of salient weights while simultaneously decreasing bit-width overhead. As shown in Figure 3 this approach incorporates a recursive computation strategy for weight binarization compensation, applying a subsequent binarization process to the residuals remaining after the initial binary process. Based on the equation 1 they redesigned the residual approximation optimization specifically for salient weights by implementing:

$$\begin{cases} \alpha_o^*, B_o^* = \arg\min_{\alpha_o, B_o} \|W - \alpha_o B_o\|^2, \\ \\ \alpha_r^*, B_r^* = \arg\min_{\alpha_r, B_r} \|(W - \alpha_o^* B_o^*) - \alpha_r B_r\|^2 \end{cases} \quad [24]$$

the $B_o$ represents the original binary tensor, while $B_r$ denotes the residual binarized matrix with the same size as $B_o$. Efficiently solving it for the two binarized optimization objectives results on this approximation:

$$W \approx \alpha_{\text{o}}^* B_{\text{o}}^* + \alpha_{\text{r}}^* B_{\text{r}}^* \quad [24]$$

To prove that the residual approach of the equation above has a lower quantization error than

the direct one of 1. The residual binarization error was defined by $\mathcal{E}$.

$$\mathcal{E}_{rb} = \|W - \alpha_{\text{o}}^* B_{\text{o}}^* - \alpha_{\text{r}}^* B_{\text{r}}^*\|_2^2 \quad [24]$$

The original binarized quantization error is calculated as $\|W - \alpha_{\text{o}}^* B_{\text{o}}^*\|_2^2$, and from the second sub-equation of equation 5.2.3 it's determined that loss $\mathcal{E}_{rb} \leq \|W - \alpha_{\text{o}}^* B_{\text{o}}^*\|^2$. Thus showing that the method of residual approximation proved to be able to further reduce the binary quantization error of salient weights with ultra-low bit-width storage compared to retaining the salient weights at their full precision or even INT8.

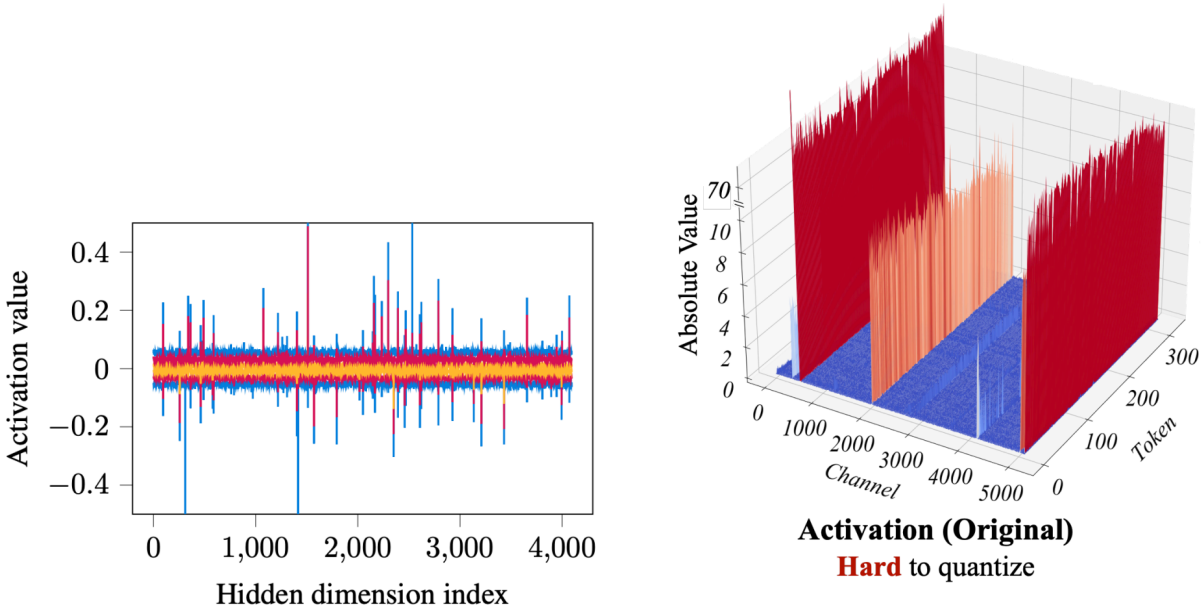
The performance of a LLaMA model with this super low quantization method is still impressive, only losing about 45% of the performance of the original model on a suit of benchmarks consisting of PIQA, ARC-e, ARC-c, HellaSwag and WinoGrand, as depicted on Figure 8.

| Method | #W | #A | #G | CommonSenseQA↑ | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | PIQA | ARC-e | ARC-c | HellaSwag | Wino | Avg. |
| LLaMA3 | 16 | 16 | - | 79.9 | 80.1 | 50.4 | 60.2 | 72.8 | 68.6 |
| RTN | 4 | 16 | 128 | 76.6 | 70.1 | 45.0 | 56.8 | 71.0 | 63.9 |
| | 3 | 16 | 128 | 62.3 | 32.1 | 22.5 | 29.1 | 54.7 | 40.2 |
| | 2 | 16 | 128 | 53.1 | 24.8 | 22.1 | 26.9 | 53.1 | 36.0 |
| | 8 | 16 | - | 79.7 | 80.8 | 50.4 | 60.1 | 73.4 | 68.9 |
| | 4 | 16 | - | 75.0 | 68.2 | 39.4 | 56.0 | 69.0 | 61.5 |
| | 3 | 16 | - | 56.2 | 31.1 | 20.0 | 27.5 | 53.1 | 35.6 |
| | 2 | 16 | - | 53.1 | 24.7 | 21.9 | 25.6 | 51.1 | 35.3 |
| GPTQ | 4 | 16 | 128 | 78.4 | 78.8 | 47.7 | 59.0 | 72.6 | 67.3 |
| | 3 | 16 | 128 | 74.9 | 70.5 | 37.7 | 54.3 | 71.1 | 61.7 |
| | 2 | 16 | 128 | 53.9 | 28.8 | 19.9 | 27.7 | 50.5 | 36.2 |
| | 8 | 16 | - | 79.8 | 80.1 | 50.2 | 60.2 | 72.8 | 68.6 |
| | 4 | 16 | - | 76.8 | 74.3 | 42.4 | 57.4 | 72.8 | 64.8 |
| | 3 | 16 | - | 60.8 | 38.8 | 22.3 | 41.8 | 60.9 | 44.9 |
| | 2 | 16 | - | 52.8 | 25.0 | 20.5 | 26.6 | 49.6 | 34.9 |
| AWQ | 4 | 16 | 128 | 79.1 | 79.7 | 49.3 | 59.1 | 74.0 | 68.2 |
| | 3 | 16 | 128 | 77.7 | 74.0 | 43.2 | 55.1 | 72.1 | 64.4 |
| | 2 | 16 | 128 | 52.4 | 24.2 | 21.5 | 25.6 | 50.7 | 34.9 |
| | 8 | 16 | - | 79.6 | 80.3 | 50.5 | 60.2 | 72.8 | 68.7 |
| | 4 | 16 | - | 78.3 | 77.6 | 48.3 | 58.6 | 72.5 | 67.0 |
| | 3 | 16 | - | 71.9 | 66.7 | 35.1 | 50.7 | 64.7 | 57.8 |
| | 2 | 16 | - | 55.2 | 25.2 | 21.3 | 25.4 | 50.4 | 35.5 |
| SliM-LLM | 4 | 16 | 128 | 78.9 | 79.9 | 49.4 | 58.7 | 72.6 | 67.9 |
| | 3 | 16 | 128 | 77.8 | 73.7 | 42.9 | 55.5 | 72.8 | 64.5 |
| | 2 | 16 | 128 | 57.1 | 35.4 | 26.1 | 28.9 | 56.6 | 40.8 |
| QuIP | 4 | 16 | - | 78.2 | 78.2 | 47.4 | 58.6 | 73.2 | 67.1 |
| | 3 | 16 | - | 76.8 | 72.9 | 41.0 | 55.4 | 72.5 | 63.7 |
| | 2 | 16 | - | 52.9 | 29.0 | 21.3 | 29.2 | 51.7 | 36.8 |
| DB-LLM | 2 | 16 | 128 | 68.9 | 59.1 | 28.2 | 42.1 | 60.4 | 51.8 |
| PB-LLM | 2 | 16 | 128 | 57.0 | 37.8 | 17.2 | 29.8 | 52.5 | 38.8 |
| | 1.7 | 16 | 128 | 52.5 | 31.7 | 17.5 | 27.7 | 50.4 | 36.0 |
| BiLLM | 1.1 | 16 | 128 | 56.1 | 36.0 | 17.7 | 28.9 | 51.0 | 37.9 |
| SmoothQuant | 8 | 8 | - | 79.5 | 79.7 | 49.0 | 60.0 | 73.2 | 68.3 |
| | 6 | 6 | - | 76.8 | 75.5 | 45.0 | 56.9 | 69.0 | 64.6 |
| | 4 | 4 | - | 54.6 | 26.3 | 20.0 | 26.4 | 50.3 | 35.5 |

Table 8: Quantized LLaMA3-8B performance[27]

### 5.2.4 Weight + Activation Quantization

Although weights are already extremely hard to quantize, incorporating the activations into the quantization pipeline adds another degree of complexity, especially for large language models. Compared to weights, activations have some unique challenges since they are dynamic, and their range and statistics are unknown until runtime. The LLMs also have a unique problem, not broadly seen for other transformer-based models: the systematic outlier activations. These outliers if clipped during quantization can cause significant degradation in performance and require special attention if model accuracy is to be preserved [16].




(a) The distributions of activations at the input to the FFN block in LLAMA2-7B model[28]

(b) Magnitude of the input activations and weights of a linear layer in OPT-13B[29]

Figure 5: Comparison of activations and weights in LLAMA2-7B and OPT-13B models.

Another problem that makes it hard to quantize is the significant variations in value range across different channels, which can be troublesome for the quatization algorithm. However, a strong motivation for undertaking this complexity is the efficiency gained by quantizing both weights and activations to low-bit data types on specific hardware. This efficiency is demonstrated by the performance of SmoothQuant, as shown in Figure 8.

### 5.2.5 Mixed Precision Quantization

RPTQ, or Reorder-based Post-Training Quantization, differs from per-tensor quantization techniques in that it does not apply the same quantization parameters uniformly across the entire tensor. This distinction is important, as uniform quantization can sometimes lead to suboptimal results.

One key problem consists of the range of quantization being too wide to cover a large value range, as this can cause increased quantization errors in channels with smaller value ranges. On

the other hand if the quantization range is too narrow it could lead to truncation of the outliers resulting in quantization errors.

For example if one channel as a range of -100 to -50 and another 80 to 100 when trying to cover their ranges by quantizing them form -100 to 100 this will result in a significant quantization error for both channels.

To address this researchers have proposed several methods one of them being LLM.int8 [30] which utilizes mixed-precision quantization by using high-precision data types (FP16) to quantize the outliers in activations and low precision data types (INT8) for the remaining values. As explained above this improves model performance preventing errors caused by the quantization of a wide range of values. Another method for quantizing the activation SmoothQuant [29] solved the problem by introducing a process that is meant to "smooth" the input activation by dividing it by a per-channel smoothing factor $s \in \mathbb{R}$, $C_i$. To keep the mathematical equivalence of a linear layer the weights are scaled accordingly in the reversed direction:

$$Y = (X \operatorname{diag}(s)^{-1}) \cdot (\operatorname{diag}(s)W) = \hat{X}\hat{W} \quad [29]$$

The next point of SmoothQuant is called Migrate Quantization Difficulty and the idea is to control the trade-off between the quantization difficulty of activations and weights by redistributing their values scale, meaning migrating the difficulty from activation to weights.

The idea works by choosing a per-channel smoothing factor $s$ such that $\hat{X} = X \operatorname{diag}(s)^{-1}$ so it's easier to quantize. To reduce quantization error, the effective quantization bits for all channels should be increased. This maximizes the total effective quantization bits when all channels share the same maximum magnitude, making the optimal choice the scale factor $s_j = \max(|X_j|)$, $j = 1, 2, \ldots, C_i$ where $j$ corresponds to the $j$-th input channel. This choice grants that after the division, all the channels will have the same maximum value, which makes it easy to quantize. However, this formula shifts all the quantization difficulty to the weights. As a result, the quantization errors tend to be larger in the weights, leading to significant accuracy degradation shown in Figure 6.

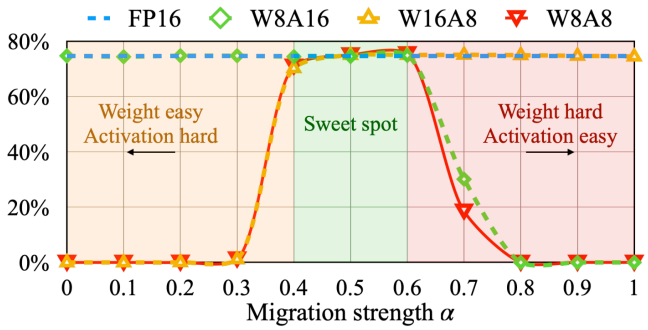



Figure 6: Finding the sweet spot for the migration strength[29]

However there is a possibility off also pushing all of the quantization difficulty from the

weights to the activations by choosing $s_j = \frac{1}{\max(|W_j|)}$. Similarly the model performance will degrade heavily due to the activation quantization errors this introduces, therefore there is a need to split all of the quantization difficulty between weights and activations so they are both easier to quantize.

SmoothQuant achieves this by introducing a hyper-parameter migration strength, depicted in Figure 6, to control how much difficulty will be migrated from activation to weights, this is done using:

$$s_j = \frac{\max(|\mathbf{X}_j|)^\alpha}{\max(|\mathbf{W}_j|)^{1-\alpha}} \quad [29]$$

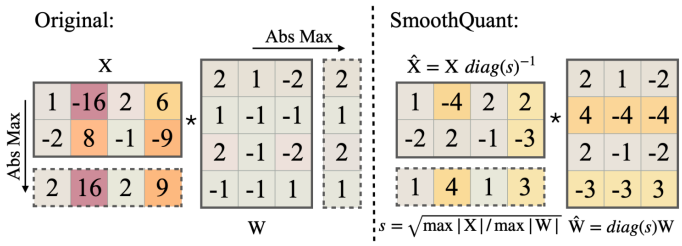



Figure 7: Main idea of SmoothQuant when $\alpha$ is 0.5. The smoothing factor s is obtained on calibration samples and the entire transformation is performed offline. At runtime, the activations are smooth without scaling.[29]

This formula ensures that the weights and activations at the corresponding channel share a similar maximum value, thus sharing the same difficulty[29]. Figure 7 illustrates the smoothing transformation when $\alpha = 0.5$, this works for models where the activation outliers aren't very significant, on models where the outliers are more significant (e.g. GLM-130B[31] which happens to have $\sim 30\%$ outliers), a larger $\alpha$ can be picked to migrate more quantization difficulty to the weights(like 0.7).

The performance results for this method are very promising on models like OPT-175B [32] show an efficient quantization at INT8 quantization level since the method can match the FP16 accuracy on all evaluation datasets. This also proved to be true for LLaMA [33]. Although the outliers in this model tend to be less severe, SmoothQuant still performed well, with an average performance drop of only $0.4\%$. The PyTorch implementation also proved effective, achieving a 1.51× speedup and a 1.96× memory reduction for OPT models.

### 5.2.6 Quantization-Aware Training (QAT)

LLM-QAT is an advanced method for Quantization-Aware training (QAT) specifically designed for LLMs[14]. This method as been proven to be accurate to quantization levels as low as 4-bits. This method helps in keeping the origianl output distribution and allows the quantization of weights, activations, and the key-value cache. There are three core components, Symetric MinMax Quantization, Student-Teacher Framework and Data Generation Process. Symmet-

ric MinMax Quantization is a method used to preserve the outliers in large language models (LLMs) and maintain their performance.

$$X_Q^i = \left[\frac{X_R^i}{\alpha}\right], \quad \alpha = \frac{\max(|X_R|)}{2^{N-1}-1}, \quad \text{[14]}$$

In the function above the $X_Q$ represents the quantized values, $X_R$ represents the full precision and the $\alpha$ is the scaling factor this is a general quantization formula and it can be applied to both weights and activations, but the method to quantize differs depending on the target. In the case of weights, per-channel quantization is used, meaning that at each channel (or filter) in the weight tensor it will be quantized independently. This approach allows the quantization process to adapt accordingly to the specific range of values in every channel, preserving more information and reducing the possible quantization error. As for activations and the KV cache, a per-token quantization is applied. In this case, the quantization is performed independently for each token, allowing the method to account for the diverse ranges of activation values across the multiple tokens. This distinction ensures that the quantization process is purposely selected to the specific characteristics of weights, activations, and KV cache, optimizing the trade-off between efficiency and accuracy for each case.

Then LLM-QAT uses the student-teacher model framework to ensure that the quantized model retains the performance of the full-precision model. This works by having the teacher model the full-precision version guide the student, which is the newly quantized model. The guidance is provided through cross-entropy-based logits distillation.

$$L_{\text{CE}} = -\frac{1}{n}\sum_{c}\sum_{i=1}^{n} p_c^T(X_i)\log(p_c^S(X_i)) \quad \text{[14]} \tag{2}$$

On Equation 2 the $i$ represents the $i$-th sample in the batch , $c$ denotes the number of classes (vocabulary size), and $T$ and $S$ are the Teacher and the Student models, respectively. The next-token data generation is based on the full-precision model and is proposed as a method to synthesize a distribution similar to the pre-training data. This data is generated by the teacher model, which begins with a random start token and iteratively generates subsequent tokens until it reaches the end of the sentence or the maximum sequence length. The LLM-QAT introduced a hybrid approach to ensure the generated data is diverse and accurate, on the hybrid approach only the first few tokens are deterministically selected with the top-1 strategy the rest are stochastically sampled from the full precision SoftMax output distribution. Lastly, the generated data is then used as input for fine-tuning the quantized model, where the teacher model's predictions serve as labels to guide training thus achieving a performance close to the original model yet quantized to a specified level.

## 5.3 Retrieval-Augmented Generation (RAG)

Large language models have been shown to learn a substantial amount of in-depth knowledge from data [34] this is accomplished without access to any outside data [5] this comes with some pros and cons. On the pros side, there's the ability of the model to capture a lot of data and compress it, on the other hand, this comes with the downside of not being able to expand the model knowledge or even revise their memory. Another downside is the hallucinations that sometimes are produced by this models. These limitations can be addressed using a method proposed by [35], known as Retrieval-Augmented Generation (RAG). This method consists of four main components: a query encoder ($q$), a retriever ($p_n$), a document indexer, and a generator ($p_0$).

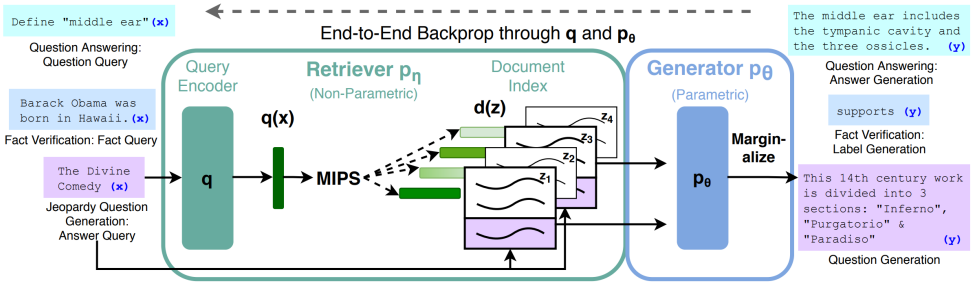



Figure 8: RAG implementation overview[35]

The first required model is a retriver DPR based on [36]. This retriver works by indexing all the passages in a low-dimensional and continuous space, so that later the top $K$ passages can be retrived efficiently. At run-time, passages that are relevant to the input question are retrieved for the reader module. The number of passages from which the model can select is extremely large the paper refers to a corpus of 21 million passages, with the value of K (i.e., the number of retrieved passages) ranging between 20 and 100.

DPR represented by $p_n(z|x)$ follows a bi-encoder architecture:

$$p_\eta(z \mid x) \propto \exp(d(z)^\top q(X)), \quad d(z) = \text{BERT}_d(z), \quad q(x) = \text{BERT}_q(x) \quad [35]$$

Where $d(z)$ is a dense representation of a document produced by a $BERT_{BASE}$ document encoder [12] and $q(x)$ a query representation produced by a query encoder, in this case also using $BERT_{BASE}$. The retrieval is done using (MIPS) Maximum Inner Product Search to calculate the top-$k(p_\eta(\cdot \mid x))$, which represents the list of $k$ documents $z$ with highest prior probability $p_n(z|x)$. MIPS identifies these documents by finding those with the largest inner product between their dense representations and the query representation. This process can be approximately solved in sub-linear time using an efficient approximation to the nearest neighbor search, like FAISS or hierarchical navigable small-world graphs. This is crucial for enabling the scalability of large document collections such as Wikipedia.

A pre-trained bi-encoder from DPR is used to initialize the retriever and to build the doc-

ument index. The Generator is used to combine the input $x$ with the retrieved content $z$ using BART, these are simply concatenated.

The generator component given by $p\left(y_i \mid x, z, y_{1:i-1}\right)$ could be modeled using any encoder decoder, however BART-large [37] a pre-trained seq2seq transformer with 400M parameters, is commonly used.

BART pre trainned using a denoising objective which served as its foundation. For this specific use case, they also added a variety of different noising functions in the training to prevent BART from overfitting and also encourage contextual understanding [38].

They refer to the BART Generator parameters $0$ as the parametric memory and all of the retrieved external knowledge as non-parametric knowledge. The marginalization can be done via two methods described in the paper RAG-Sequence model [39] uses the same retrieved document to generate the entire output sequence. Specifically, it treats the retrieved document as a single latent variable, which is marginalized to obtain the sequence-to-sequence probability p(y—x) using a top-K approximation.

$$\begin{aligned} p_{\text{RAG-Sequence}}(y|x) &\approx \sum_{z \in \text{top-k}(p(\cdot|x))} p_\eta(z|x) p_\theta(y|x,z) \\ &= \sum_{z \in \text{top-k}(p(\cdot|x))} p_\eta(z|x) \prod_{i=1}^{N} p_\theta(y_i|x,z,y_{1:i-1}) \end{aligned} \quad \text{[35]} \quad (3)$$

This marginalization allows the model to combine information from the top$k$ document, effectively converging the information from diverse sources within the same document to generate a coherent and contextually accurate output. The final retrieval step ensures that the most relevant documents are selected based on their dense representations,

The second method is RAG-Token model which can be used to draw a different latent document for each target token and marginalize accordingly. This makes it possible for the generator to choose the content from several documents when producing the answer. The top-K documents are retrieved using a retriever. The generator then produces a probability distribution for the next output token for each retrieved document. This process is repeated iteratively, marginalizing over the documents at each step, to generate the subsequent output tokens:

$$p_{\text{RAG-Token}}(y|x) \approx \prod_{i=1}^{N} \sum_{z \in \text{top-}k(p(\cdot|x))} p_\eta(z|x) p_\theta(y_i|x, z_i, y_{1:i-1}) \quad \text{[35]}$$

In the case for sequence classification tasks RAG-Sequence and RAG-Token can be used by considering the target class as a target sequence of length one.

| Aspect | RAG-Token | RAG-Sequence |
|---|---|---|
| **Document Usage** | Different documents for each token. | Same document for the entire sequence. |
| **Marginalization** | Per-token over top-K documents. | Per-sequence over top-K documents. |
| **Beam Search** | Standard beam search. | Beam search for each document. |
| **Efficiency** | Computationally efficient. | Thorough Decoding is expensive |
| **Flexibility** | Combines information from multiple documents dynamically. | Relies on a single document for the entire sequence. |

Table 9: Comparison of Decoding Methods

### 5.3.1 Hypothetical Document Embeddings (HyDE)

HyDE [40] is another retrieval method that aims to increase the performance of the model on zero-shot scenarios, meaning that they can retrieve relevant documents without requiring specific training. This model is meant to work with any type of NLP model and is able to generalize across multiple tasks. This method differs from RAG [35] in that it uses a generator to produce a hypothetical document based on the input query. This document does not need to be factually correct, as it is only used by the retriever (e.g., Contriever), which transforms it into a dense embedding vector. This embedding represents the hypothetical document in a high-dimensional vector space.

The retriever is then used to search the corpus for real documents that are similar in the embedding space, this similarity is measured using inner product similarity between the hypothetical document embeddings and the real document embeddings. Then the most similar document is fed into the model which also receives the input query, it then generates the output for the query based on the retrieved document.

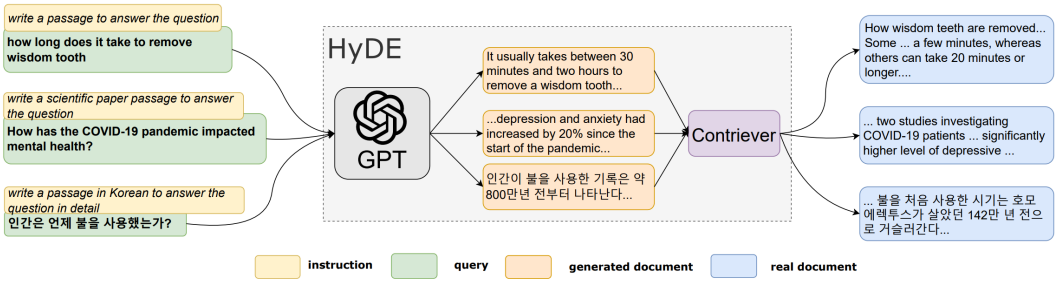



Figure 9: An illustration of the HyDE model.[40]

The main issue addressed by [40] is the dependence on a separate query encoder required

by RAG [35] systems. This is because dense retrievers compute similarity between the query and documents using inner product similarity, which necessitates a dedicated query encoder.

Firstly it uses two encoder $enc_q enc_d$ that maps the query $q$ and the document $d$ into $d$ dimension vectors $v_q, v_d$, whose inner product is used as the similarity measurement.

$$\text{sim}(q, d) = \langle \text{enc}_q(q), \text{enc}_d(d) \rangle = \langle v_q, v_d \rangle \quad [40]$$

This is where zero-shot dense retrieval problems lie, it requires learning two embedding functions one for the query and the other for the document, these need to align into the same embedding space where the inner product can capture the document's relevance. HyDE solves this problem by performing a search in the document-only embedding space that captures the document's similarity. This method can be easily learned using unsupervised contrastive learning [41]. They set the $enc_d$ directly as the contrastive encoder $enc_con$ as follows:

$$f = enc_d = enc_con \quad [40] \tag{4}$$

This unsupervised contrastive encoder is be shared by all incoming document corpus. The function 4 is also denoted as $f$.

$$v_d = f(d) \quad \forall d \in D_1 \cup D_2 \cup \cdots \cup D_L \quad [40]$$

To build the query vector they use an instruction following LLM, in this case, text-davinci-003 from OpenAi's GPT-3 series [42], this was specifically picked due to its generalization ability, they call it $InstructLM$ so it's easy to represent. It then takes the query $q$ and a textual instruction $INST$ and follows them to perform the task specified in $INST$, like so:

To build the query vector, they use an instruction-following LLM in this case, text-davinci-003 from OpenAI's GPT-3 series[42]. This model was specifically chosen for its strong generalization capabilities and is referred to as $InstructLM$ for ease of representation. It takes the query $q$ and a textual instruction $INST$, and follows the instruction to perform the specified task, as shown below:

$$g(q, \text{INST}) = \text{InstructLM}(q, \text{INST}) \quad [40]$$

The $g$ can be used to map queries to the hypothetical documents by sampling from g, the $INST$ is set to be "write a paragraph that answers the question", the generated document isn't real and may be factually incorrect due to models hallucinations [42]. However, this is not important because the hypothetical document is used only to capture the relevance pattern. Then the relevance modeling is offloaded to an NLG that has the ability to generalize more easily, naturally, and more effectively. Another great thing about generating the examples is that this also replaces explicit modeling of relevance scores making it so there's no need to compute the

query-document relevance.

$$E[v_{q_{ij}}] = E[f(g(q_{ij}, \text{INST}_i))] \quad [40] \tag{5}$$

The $f$ corresponds to the document encoder, $g$ defines a probability distribution based on the chain rule. In their implementation, they assume that the distribution of $v_q ij$ is uni-modal, implying that the query is not ambiguous. To estimate Equation 5, they sample $N$ documents from $g$, $\left[\hat{d}_1, \hat{d}_2, \ldots, \hat{d}_N\right]$.

$$\hat{v}_{q_{ij}} = \frac{1}{N} \sum_{\hat{d}_k \sim g(q_{ij}, \text{INST}_i)} f(d_k)$$

$$= \frac{1}{N} \sum_{k=1}^{N} f(\hat{d}_k) \quad [40]$$

They also consider the query as a possible hypothesis,

$$\hat{v}_{q_{ij}} = \frac{1}{N+1} \left[ \sum_{k=1}^{N} f(\hat{d}_k) + f(q_{ij}) \right] \quad [40]$$

The inner product is computed between $\hat{v}_{qij}$ and the set of all document vectors $\{f(d)|d \in D_i\}$, then the most similar documents are retrieved. In their implementation, the encoder function $f$ acts as a lossy compressor, producing dense vectors in which unnecessary details are filtered out. This enables the system to use a generated document even if it lacks factual accuracy to effectively search for the correct one.

According to their study, HyDE remains competitive even when compared to fine-tuned models. Another strong result comes from the web research setting, where the performance of HyDE is particularly impressive even when compared to methods that rely on relevance judgments. Notably, HyDE achieves these results without requiring such judgments.

| | DL19 | | | DL20 | | |
|---|---|---|---|---|---|---|
| | map | ndcg@10 | recall@1k | map | ndcg@10 | recall@1k |
| *w/o relevance judgement* | | | | | | |
| BM25 | 30.1 | 50.6 | 75.0 | 28.6 | 48.0 | 78.6 |
| Contriever | 24.0 | 44.5 | 74.6 | 24.0 | 42.1 | 75.4 |
| HyDE | **41.8** | **61.3** | **88.0** | **38.2** | **57.9** | **84.4** |
| *w/ relevance judgement* | | | | | | |
| DPR | 36.5 | 62.2 | 76.9 | 41.8 | **65.3** | 81.4 |
| ANCE | 37.1 | **64.5** | 75.5 | 40.8 | 64.6 | 77.6 |
| Contriever$^{FT}$ | 41.7 | 62.1 | 83.6 | **43.6** | 63.2 | **85.8** |

Table 10: Results for web search on DL19/20. Best performing w/o relevance and overall system(s) are marked bold. DPR, ANCE and ContrieverFT are in-domain supervised models that are finetuned on MS MARCO training data. [40]

### 5.3.2 Cache Augmented Generation

RAG is the most used approach for enhancing language models but as discussed previously it has two main drawbacks, specifically the retrieval latency and the potential errors in document selection [43]. LLMs have been increasing their context size over the years and the CAG [43] approach proposes a method that uses this increased context size to reduce the model latency and potential errors that could occur on RAG systems.

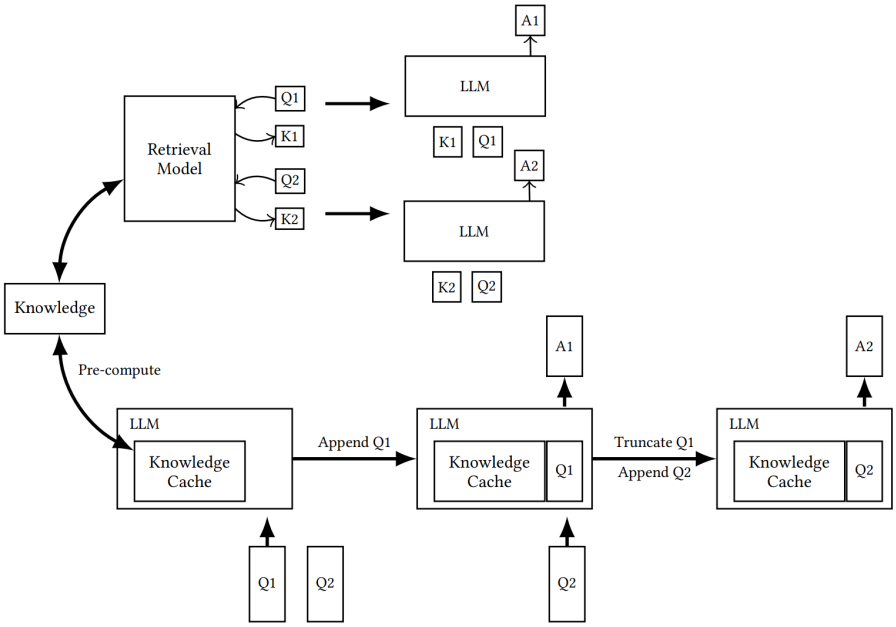



Figure 10: Comparison RAG on the top and CAG on the botom[43]

This approach works by enabling retrieval-free knowledge integration. This is done by preloading external knowledge sources, such as a collection of documents $D = \{d_1, d_2, ..., d_n\}$ and precomputing these documents key-value (KV) cache $C_KV$, this addresses some of the computational challenges and inefficiencies inherent to real-time retrieval on RAG systems.

This approach is divided into three main steps:

External Knowledge Preloading, represents the adaptation and preprocessing of the collection of documents $D$ that are relevant to the target application, this adapts them to fit within the model's context window. This is done by the LLM $M$, with parameters $0$, and processed the documents $D$, thus transforming it into a precomputed KV cache:

$$C_KV = KV - ENCODE(D) \quad [43]$$

This $KV$ cache which contains the inference state of the LLM, is stored on disk or in memory for future use. This implementation brings some of the computational cost down because the cost of processing $D$ is incurred only once even if used in multiple subsequent queries.

During this stage the precomputed $KV$ cache $C_KV$ is loaded alongside the user's query $Q$,

then the LLM uses this cached context to generate the responses:

$$R = M(Q|C_K V) \quad [43]$$

By giving the model the external cached knowledge it eliminates the retrieval latency and reduces the risk of errors or omissions that comes with dynamic retrieval. The prompt $P = Concat(D, Q)$ ensures a unified understanding of the user query and the external knowledge.

Cache Reset is the part of the system is responsible for maintaining performance across multiple inference sessions. Since the $KV$ cache grows in an append-only manner sequentially storing new tokens $t_1, t, 2, ..., t_k$ the context may eventually need to be freed. This is achieved by truncating the oldest tokens to prevent the cache from exceeding memory limits.

$$C_{KV}^{\text{reset}} = \text{Truncate}(C_{KV}, t_1, t_2, \ldots, t_k) \quad [43]$$

This ensures that the re-initialization is fast without the need to reload it from disk, ensuring a constant speed and responsiveness.

### 5.3.3 Hybrid approaches

There are two main Hybrid approaches focusing on different implementations RAGCache [44] and Self-Route [45] though these can't be compared directly in terms of their implementation since RAGCache is a system-level optimization for RAG and Self-Route is an approach that selects the optimal way to give context to the model.

Starting with Self-Route, this method combines the benefits of Retrieval-Augmented Generation (RAG) notably its proven effectiveness and efficiency in leveraging external knowledge with the capabilities of recent long-context (LC) models, which can directly process and understand extended contexts. Models like Gemini 1.5 Pro [46] is able to achieve near-perfect recall of up to 1M tokens and maintains this recall performance of up to 10M tokens. If this trend of bigger and bigger context sizes continues this method could be a big improvement over traditional RAG implementations.

Although one problem with using long-context (LC) models is the increased computational cost, their performance can sometimes exceed that of RAG implementations. However, RAG is significantly more efficient, with its cost estimated to be around 20% of that required by LC models.

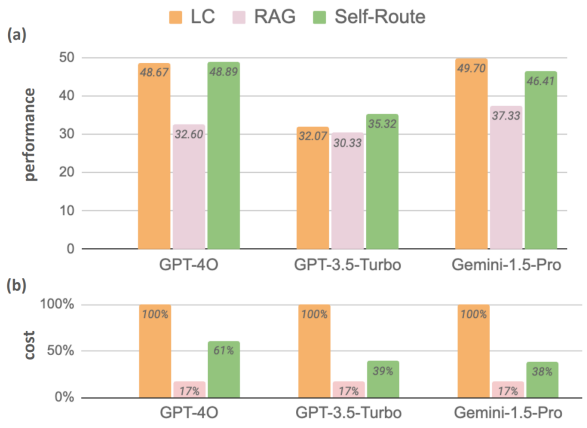



Figure 11: Comparison between performance and costs on multiple models using LC, RAG and Self-Route[45]

As seen in Figure 11 the performance is maintained if not improved on some models but the cost on most cases is less than half. This is due to the nature of the approach, which combines the strengths of both methods. When RAG can be applied, it offers reduced computational costs while maintaining most of the performance. However, despite the performance gap, there is a high degree of overlap in the predictions made by both methods.

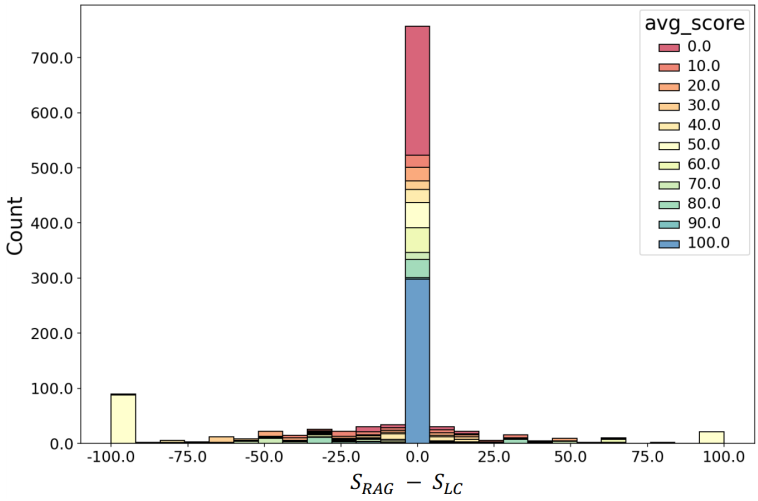



Figure 12: Distribution of the difference of prediction scores between RAG and LC[45]

Figure 12 shows the differences between RAG prediction scores $S_{RAG}$ and LC prediction scores $S_{LC}$ these are not just similar, in $63\%$ of queries the model predictions are exactly identical, and for $70\%$ of queries, the score difference is less than $10\%$. This is also true for incorrect answers when looking at the red color which corresponds to an accuracy of 0 it can be seen that RAG and LC make similar errors as well.

Self-Route uses the LLM itself to route queries based on self-reflection, under the idea that LLMs are well-calibrated in predicting whether a query is answerable given the provided context, this is done using a two-step approach RAG-and-Route and long-context prediction.

RAG-and-Route works by providing the original query along with the retrieved text chunks to the LLM, prompting it to assess whether the query is answerable based on the given context. If the model determines that the query can be answered, it proceeds to generate a response. However, if it deems the query unanswerable, it is instructed to return the phrase ”unanswerable” as per the prompt: ”Write ’unanswerable’ if the query cannot be answered based on the provided text.” In such cases, a fallback approach is then triggered.

This approach consists of giving the LLM the full LC which consists of all documents able to fit the context , although this is not explained in the paper, it gives some hints that this must be the case. This as seen on Figure11 giving a better result although with a higher cost. This trade-off is most cost-efficient when $k = 5$, meaning the number of retrieved documents is five. This is because, as k increases, the cost of RAG also increases, but so does the number of queries that can be successfully routed. Ultimately, the efficiency depends on the specific task being evaluated. For instance, in extractive question answering tasks where multi-hop reasoning is not required a lower k (e.g., $k = 1$) may result in lower computational cost. Conversely, tasks that require deeper reasoning may benefit from a higher $k$. Therefore, the optimal value of k is dependent on both the nature of the task and the level of performance required.

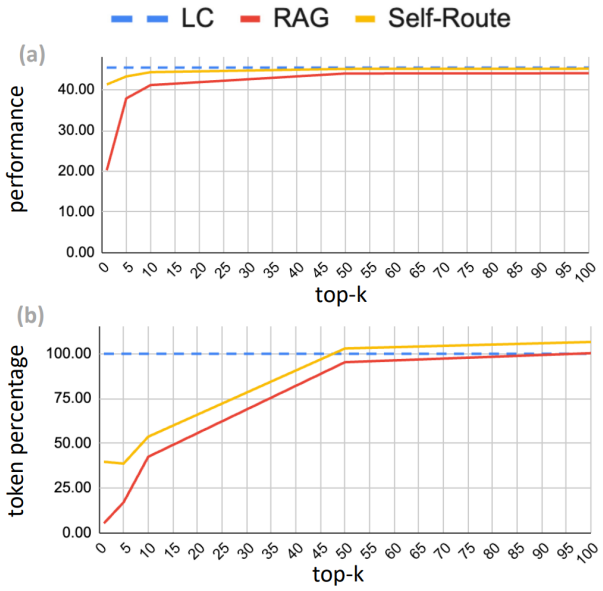



Figure 13: Trade-off curves between (a) model performance and (b) token percentage as a function of k.[45]

The other hybrid approach RAGCache works by caching the key-value tensors of retrieved documents across multiple requests, this is done to try and minimize redundant computation for efficiency gains.

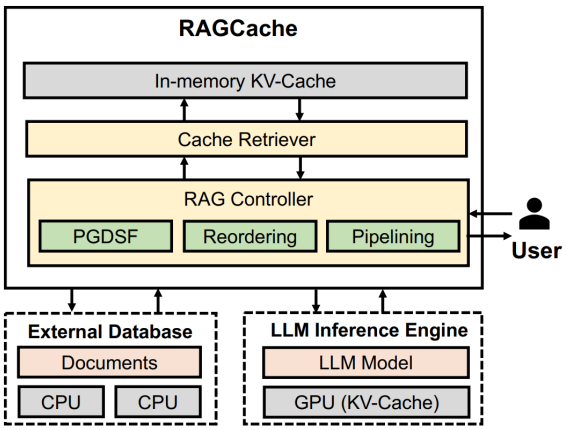



Figure 14: RAGCache Overview[44]

Cache Structure and Replacement Policy operate differently from traditional cache systems that cache individual objects. Instead, this method caches the key-value tensor of the retrieved documents, which are sensitive to the order in which they are referenced. For instance consider two document sequences $[D_1, D_3]$ with key-value tensors $KV$ and $[D_2, D_3]$ with $KV'$ respectively. Although $KV[1]$ and $KV'[1]$ both contain $D_3$, their value differs. This occurs because the key-value tensor of a given token being generated based on the preceding tokens, thus underscoring the order-dependence of key-value tensors. To aid retrieval speed while maintaining document order, RAGCache structures the document's key-value tensors with a knowledge tree 15. The knowledge tree assigns each document to a node that refers to the memory addresses of the document's key-value tensors. Similarly to vLLM [47], RAGCache also stores key-value tensors in non-contiguous memory blocks, allowing the KV cache to be reused. The root of the tree, $S$, corresponds to the shared system prompt. A path from the root to any node represents a unique sequence of documents, thus allowing RAGCache to handle multiple requests simultaneously by leveraging overlapping paths.

RAGCache retrieves tensors by performing prefix matching along these paths. If a subsequent document in a sequence cannot be found among the child nodes, the traversal is terminated, and the longest identified document sequence is returned. This method ensures efficiency with a time complexity of $O(h)$, where h is the height of the tree.

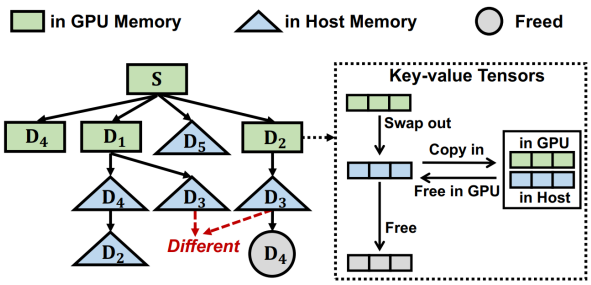



Figure 15: Knowledge Tree[44]

Prefix-aware Greedy-Dual-Size-Frequency (PGDSF) replacement policy is the name for the

node placement optimizer. The knowledge tree decides each node's placement within a hierarchical cache. For example, nodes that are constantly accessed are ideally stored in GPU memory, while those that are accessed less often are stored in slower host memory or are freed completely. The node placement optimization occurs when RAGCache uses a PGDSF replacement policy that evaluates each node based on the access frequency, size, and access cost, due to its limited storing capacity the priority is defined by:

$$\text{Priority} = \text{Clock} + \frac{\text{Frequency} \times \text{Cost}}{\text{Size}} \quad [44] \tag{6}$$

Nodes that have a lower priority get freed first while the opposite is also true. The Clock tracks node access frequency, but to adapt to the cache hierarchy, there are two separate logical clocks: one for the GPU and another for the host memory. The Clock starts at zero and updates every eviction, when a document is retrieved its clock is set and its priority gets adjusted this imposes that nodes with older clocks meaning less recent use, receive lower priorities.

$$\text{Clock} = \max_{n \in E} \text{Priority}(n) \quad [44]$$

Frequency in Equation 6 represents the total retrieval count for a document within a time frame, this count is reset upon system start or cache clearance. $Priority$ is directly linked to the frequency so the more frequently a document is accessed the higher the priority. $Size$ is the number of tokens in the given document post-tokenization, thus directly linked to the memory required for its key-value tensors. The $Cost$ is defined as the time taken to compute a document's key-value tensors, this can vary depending on GPU performance as well as document size and the sequence of preceding documents.

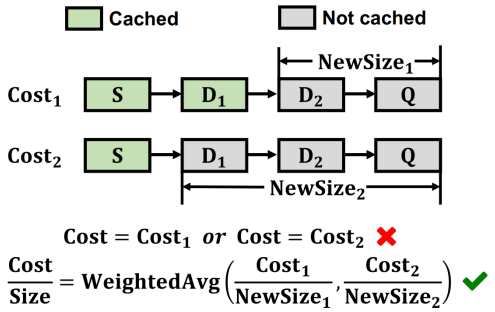



Figure 16: Cost estimation PGDSF[44]

Prefix awareness for RAG is achieved by the PGDSF method through two primary components, cost estimation and node placement. Accurately determining the computational cost for RAG operations is challenging due to the complex dynamics of LLM generation. Figure 16 illustrates this challenge by showing the cost differences for an identical request, denoted as $[S, D_1, D_2, Q]$, under different caching conditions.

Estimating the cost contribution of $D_2$ is imprecise because the marginal cost depends heav-

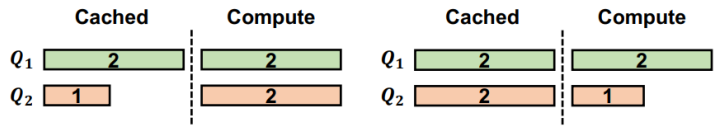
ily on the cache prefix. For instance, if the prefix $[S, D_1]$ is already cached, the subsequent computational cost includes the generation of key-value tensors for both $D_1$ and $Q$. Making it extremely difficult to isolate the cost attributed solely to $D_2$. To address this issue, PGDSF replaces the $Cost/Size$ term in Formula 6 with the following Equation:

$$\text{Cost Size} = \frac{1}{m}\sum_{i=1}^{m}\frac{\text{Cost}_i}{\text{NewSize}_i} \quad [44] \tag{7}$$

In Equation 7, $m$ represents the number of requests that access a document not currently in cache. The term $Cost_i/NewSize_i$ denotes the compute time per non-cached token for the $i$-th request. This approach effectively amortizes the computational cost across all non-cached tokens, thereby incorporating the document's size into the priority calculation. The cost, $Cost_i$ , is determined through an offline profiling process where RAGCache measures the LLM prefill time for multiple combinations of cached and non cached token lengths. Subsequently, it employs bilinear interpolation to estimate the cost for any given request at runtime. Each document retrieval event triggers an update to the corresponding node's frequency, its cost estimation, and the clock mechanism within the knowledge tree. Furthermore, if a retrieved document is not already in the cache, a new node is created for it in the tree.

The management of the node placement on the GPU, host, or free is performed by the PGDSF as seen on Figure 15. The nodes in GPU serve as parent nodes to those in host memory, establishing a hierarchical structure. RAGCache also manages node eviction across these two segments for efficiency, which is especially true when GPU memory is full. When this occurs, RAGCache swaps the lowest-priority node in the leaf nodes to the host memory; this process also occurs on the host memory when it is full, though in that case, it is an eviction. This strategy takes into account the Knowledge tree hierarchical partitioning, which is one key point to align with memory sensitivity and prefix sensitivity in LLM generation. Due to the node needing its parent node for key-value tensor calculation, the required placement of the parent node is prioritized this is so rapid retrieval can be achieved.

Because of PCIe limitations when connecting the GPU with the host memory in comparison with GPU HBM, RAGCache adopts a swap-only-once strategy depicted in Figure 15, where you can see that the key-value tensors of a node are swapped out to the host memory only for the first eviction. The host memory is responsible for keeping the key-value tensors until the node is fully evicted from the entire cache. For any subsequent evictions in GPU memory, RAGCache directly frees the node node without copying any data Due to the size of the host memory being two orders of magnitude larger than the GPU memory, keeping one copy of the key-value tensors in the host memory is acceptable.




**(a)** Differ in cached length. **(b)** Differ in compute length.

Figure 17: Cache aware Reordering[44]

Cache hit rate is vital for RAG Cache's cache efficiency, but when paired with the unpredictability of the arrival pattern in user requests, this results in substantial cache trashing.

Requests that refer to the same document may not be issued on the same time frame thus affecting the cache efficiency. For example given the requests $\{Q_i, i\%2 == 0\}$ and $\{Q_i, i\%2 == 1\}$ that target the documents $D_1$ and $D_2$ respectively. When the cache capacity is only one document, the sequence $\{Q_1, Q_2, Q_3\}$ causes frequent swapping of the key-value cache of $D_1$ and $D_2$, making it a zero cache hit rate. But if a bit of attention is paid to rearrange requests to $\{Q_1, Q_3, Q_5, Q_2, Q_4, Q_6, Q_6, ...\}$ this achieves a cache hit rate of $66\%$ thus optimizing cache utilization. This shows how strategic request ordering can mitigate cache volatility and improve cache efficiency. To introduce the cache-aware reordering algorithm, two scenarios were considered to show the key insights, the recomputation cost was assumed to be proportional to the recomputation length. The first scenario is shown on Figure 17, $(a)$ where it considers requests with identical recomputation demands but varying cached context lengths with a limit of four cached documents. With the initial order of $Q_1, Q_2$ the system must clear $Q_2$'s cache space to fit $Q_1$'s computation, then reallocate memory for $Q_1$'s processing effectively uses $Q_1$'s cache while discarding $Q_2$'s, resulting in a computational cost of $2 + 1 + 2 = 5$. On the other hand, if the order was given as $Q_2, Q_1$ this would result in a usage of $Q_2$'s cache but discarding $Q_1$'s, which would increase computation to six due to $2+2+2 = 6$. This is why cache-aware reordering advocates to prioritize requests with larger cached context thus improving cache efficiency as this brings larger benefits. In the second scenario $(b)$, the aim was to examine requests with similar cached context lengths but varying recomputation demands, with a cache capacity of five documents. On a sequence $\{Q_1, Q_2\}$, the system must clear $Q_2's$ cache to allocate space for $Q_1's$ computation, given only one available memory slot. This makes it necessary to recompute $Q_2$ entirely, which results in a cost of $2 + 2 + 1 = 5$. On the other hand the sequence $\{Q_2, Q_1\}$ allows for direct computation of $Q_2$, due to adequate cache availability. It also reduces the total computation cost to $2 + 1 = 3$, thus the reason why cache-aware reordering is beneficial when it prioritizes requests with shorter recomputation segments, this way results in a minimization of the adverse side effects on cache efficiency. RAGCache uses a priority queue for managing incoming requests, this prioritizes the incoming requests based on their impact on cache performance, the priority metric is defined by:

$$\text{OrderPriority} = \frac{\text{Cached Length}}{\text{Computation Length}} \quad [44] \tag{8}$$

Equation 8, directly prioritizes the requests that will probably lead to enhanced cache efficiency. This is directly linked to the increase in the cache hit rate and the decreased total computation time of RAGCache. Model performance and resource usage are also improved thanks to this implementation. To avoid possible starvation when requests don't align with the cached documents RAGCache sets a window for each request to ensure that all requests are processed in a timely manner.

On an LLM enhanced with RAG, the key performance bottleneck is usually the LLM generation, however, if the vector database grows to a larger scale or a higher accuracy is needed in the retrieval, this may cause the retrieval step to incur a substantial latency.

To control the impact of retrieval latency, RAGCache employs dynamic speculative pipelining to overlap knowledge retrieval and LLM inference, and one thing that can occur is that the vector search may produce results earlier in the retrieval step, which can be used by the LLM for speculative generation ahead of time. This works by a vector search maintaining a queue of top-$k$ candidate documents, which are ranked based on their similarity to the request. During the retrieval process the top-$k$ documents are being constantly updated this is done so that documents that still being discovered, might come with greater similarity and so they get inserted into the top-$k$. What could also occur is that the final documents may emerge early in the retrieval step [48]. Based on that RAGCache introduced a speculative pipelining strategy that splits a request's retrieval process into several stages. In each stage RAGCache ticks the vector database to send the possible document to the LLM for a speculative generation, if the received document is changed then the LLM will start a new speculative generation and terminate the previous one, if that doesn't happen then the LLM engine just continues with the generation. When the top-$k$ documents are finalized and there are no more changes to the top-$k$ these are sent by RAGCache to the LLM engine and if they match with the ones previously received the engine simply returns the latest speculative generation. Otherwise, the LLM performs a new generation with the new top-$k$ documents.

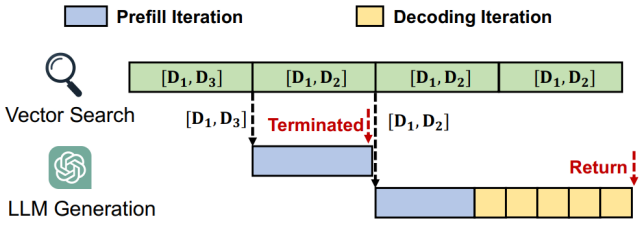



Figure 18: Speculative Pipelining[44]

Figure 18 shows how RAGCache splits the retrieval process into four stages. The top-2 documents in candidate queue are $[D_1, D_3]$, $[D_1, D_2]$, $[D_1, D_2]$ and $[D_1, D_2]$ in the four stages. After the first stage is concluded, RAGCache sends $[D_1, D_3]$ to the the LLM engine for speculative generation. When stage two is concluded, RAGCache sends $[D_1, D_2]$ to the LLM engine, the LLM engine is responsible for checking if the $[D_1, D_3]$ and $[D_1, D_2]$ are different if that's

the case it terminates the previous speculative generation and starts a new one with the correct documents. In stage three, the LLM engine receives the exact same documents as for stage two, so it continues with the previously started speculative generation. In the last stage, RAGCache sends the final top-2 documents to the engine which are still the exact same as the ones in stage two so there is no change to the speculative generation which is directly returned by the LLM engine as the result.

The speculative pipelining allows RAGCache to overlap the retrieval and generation steps, and this greatly improves the end-to-end latency of RAG systems. But this can introduce a lot of extra steps in the computation of the engine response. This can be seen above Figure 18, some speculative generations are incorrect and need to be recalculated. This can lead to performance degradation under high system loads, but to solve this RAGCache dynamically enables speculative pipelining based on the system load. As an example, they assumed that the vector search and the LLM both serve only one request at a time. This vector search produces candidate retrieval results at the end of each stage with a fixed time interval $d$. Since the batch size was set to only one, they could terminate any incorrect speculative generation requests.

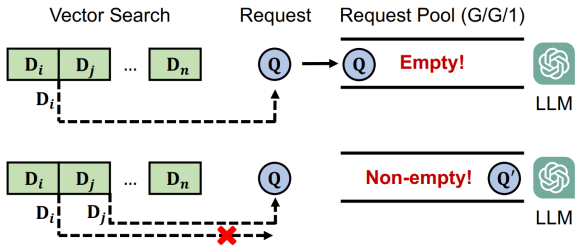



Figure 19: Optimal speculative pipelining strategy [44]

RAGCache assumes that the LLM engine can schedule requests in the queue in any order, but it processes speculative generation requests for a single request sequentially. Figure 19 illustrates the optimal speculative pipelining strategy under this setting.

## 5.4 Challenges and Applications of Quantization in RAG

## 5.5 Retrieval Methods to enhance HyDE

Due to the approach taken by RAG and HyDE on how they retrieve documents, these methods could be adapted or specifically chosen based on their ability in certain tasks. Retrievers are the basis for content that was retrieved from an external document corpus to enhance the LLM output, as well as provide grounds for the generated information on accurate documents. A more in-depth research will be done about some of these retrievers.

### 5.5.1 Contriever

The original implementation of HyDE uses the Contriever model as its retriever. This approach works on the basis of contrastive learning, which is based on the fact that every document is, in some way, unique. According to [49], this is the only available information in the absence of manual supervision. A contrastive loss is used to learn by discriminating between documents. This loss compares either a positive loss when they are the same document or negative when it's from different documents. The formula responsible for this is:

$$L(q, k_+) = -\frac{\exp\left(s(q, k^+)/\tau\right)}{\exp\left(\frac{s(q,k^+)}{\tau}\right) + \sum_{i=1}^{K} \exp\left(\frac{s(q,k_i)}{\tau}\right)}, \quad \text{[49]} \tag{9}$$

In Equation 9, $q$ corresponds to the given query, which has an associated positive document $k+$, and a pool of negative documents$(k_i)_{i=1..k}$, $\tau$ is the temperature parameter used to adjust the sensitivity of the Contriver. This function's construction encourages positive pairs to have high scores and negative pairs to have low scores. One crucial piece of this method is how to build positive pairs from a single input. This could be done in two main ways: the Inverse Cloze Task or Independent cropping.

The usage of the Inverse Cloze Task is a data augmentation strategy that generates two mutually exclusive views of a document. This approach was first described in [50]. The first view is obtained by randomly sampling a span of tokens from a segment of text, and the second view is obtained by using the complement of the span. This is done by in a given sequence of text $(w_1, ..., w_n)$, ICT samples a span $(w_a, ..., w_b)$, where $1 \leq a \leq b \leq n$, and then uses the tokens of the span as the query and the complement $(w_1, ..., w_{a-1}, w_{b+1}, ..., w_n)$ as the key.

The usage of the Independent is critical for matching the query with the document directly. This method is commonly used on images where multiple views are generated independently by cropping the input. Since this implementation is used for text it is done by sampling a span of tokens. Because of the importance of the positive pairs this strategy samples independently two spans from a document to form the needed pair. Contrary to the inverse Cloze task in the cropping stage both views of the example correspond to the same contiguous subsequence of the original data. Another difference is that between cropping and ICT is that independent random cropping is symmetric meaning both of the queries and documents follow the same distribution. This also causes overlap between the two views of the data, this being one of the reasons that encourages the network to learn exact matches between query and document. This works very similar to how lexical matching methods function, BM25 being a great example of this. So you could either fix the length of the span for the query and key or sample them both.

A big part of contrastive learning is how the system handles negative pairs this includes sampling a large set of negatives. This was tested by [49] using two methods, in-batch negative sampling and MoCo.

The first approach is to generate the negatives by using the other samples from the same

batch. For example, each item in the batch is transformed twice to generate the positive pairs, and the negatives are generated by using the other examples views from the batch, they called these ”in-batch negatives”. In this specific case, the gradient is back-propagated through the representations of both the queries and the keys. The main downside to this method is the requirement for extremely large batch sizes to work well.

The other approach Negative pairs across batches tries to solve the problem by storing the representations from previous batches in a queue and using these as negative examples in the loss calculation. This makes it possible to have a smaller batch size but may slightly change the loss by making it asymmetric between the queries. This being the view generated from the elements of the current batch, and the keys, which are the elements already stored in the queue. This occurs as the gradient is only back-propagated through the queries, leaving the representation of the keys fixed. This is caused by the features being already stored in the queue from prior batches coming from past interactions with the network. A problem occurs when the network is rapidly evolving during training can cause the performance to drop.

Instead, they used the approach called MoCo [51] which generates representation keys from a second network that is updated more slowly, the two networks are as follows one is responsible for the keys, parametrized by $0_k$, and another network for the query, parametrized by $0_q$. The parameter for the query network gets updated using back-propagation and stochastic gradient descent. This works similarly to when in-batch negatives are used. On the other hand, the key network also called Momentum encoder is only updated from the parameters of the query network using an exponential moving average.

## 5.6 Self-Knowledge Guided Retrieval Augmentation

This approach uses the few-shot prompts to make the LLM judge if it knows the answer or not. In the case that the answer isn't known, SKR [52] proceeds to the retrieval using RAG to improve on the model response.

Their method works by three main ways: Collecting Self-Knowledge, Eliciting Self-knowledge, and Using Self-Knowledge. These represent the main pipeline for SKR, as shown in Figure 20.

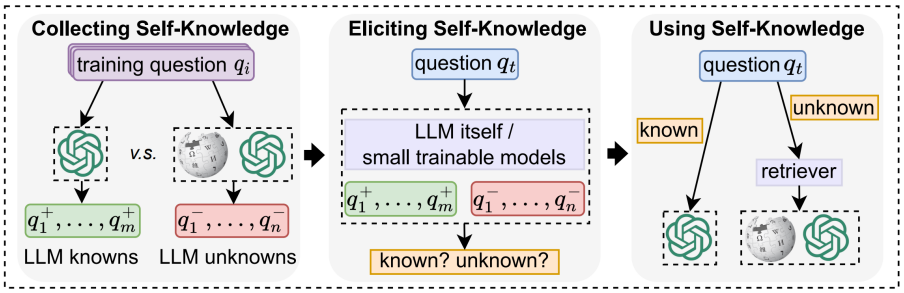



Figure 20: The SKR Pipeline and its component interactions.[52]

### 5.6.1 Collecting Self-Knowledge

Given a dataset $D$ with question-answer pairs $\{q_j, a_j\}_{j=1}^{|D|}$, the model $M$ is used to generate the answers for each entry $q_i$:

$$\hat{a}(M, q_i) = \mathcal{M}(q_1 \circ a_1, \ldots, q_d \circ a_d, q_i) \quad [52] \tag{10}$$

Where $\circ$ denotes the concatenation and $\{q_j \circ a_j\}_{j=1}^{d}$ are $d$ demonstrations. Equation 10 represents the generated answer with $\hat{a}(M, q_i)$, this also represents the internal knowledge to the question $q_i$ in $M$. The other approach is to possibly find passages from external resources that may be related with said question $q_i$, these passages can then be used as additional information provided on the model input. This is done per query, using a pretrained retriever represented by $R$ to find the related information from the corpus $C$:

$$p_i = \{p_{i1}, p_{i2}, \ldots, p_{ik}\} = \mathcal{R}(q_i, C), \quad [52] \tag{11}$$

According to Equation 11 the top-$k$ retrieved passages for the question $q_i$ are represented by $p_i = \{p_{i1}, p_{i2}, \ldots, p_{ik}\}$. A dense passage retriever is used [36] for $R$, and $C$ consists of passage chunks from Wikipedia. Then they use $M$ again to generate the answer with retrieval augmentation:

$$\hat{a}^{\mathcal{R}}(M, q_i) = \mathcal{M}(q_1 \circ p_1 \circ a_1, \ldots, q_d \circ p_d \circ a_d, q_i \circ p_i). \quad [52] \tag{12}$$

Based on both answers $\hat{a}(M, q_i), \hat{a}^R(M, q_i)$ 12, and the ground-truth answer $a_i$, they can categorize each question into a positive subset $D^+$ and a negative sub-set $D^-$ using the differences between the results:

$$q_i \in \begin{cases} \mathcal{D}^+, & \text{if } E[\hat{a}(M, q_i)] \geq E[\hat{a}^{\mathcal{R}}(M, q_i)]; \\ \mathcal{D}^-, & \text{otherwise,} \end{cases} \quad [52]$$

In Equation 5.6.1 $E$ is an evaluation metric like accuracy and exact match score, but they get discarded if the question $q_i$, answer $\hat{a}(M, q_i)$ and $\hat{a}^R(M, q_i)$ are incorrect. They then split the training set into a subset $D^+ = \{q_i^+, \ldots, q_m^+\}$ which include questions that $M$ can directly give correct answers to without external knowledge $R$ and the other subset $D^- = \{q_1^-, \ldots, q_n^-\}$ where the $R$ is needed for more accurate results.

### 5.6.2 Eliciting Self-Knowledge of LLMs

The four different strategies proposed to detect the self knowledge of the target questions are direct prompting, in-context learning, training classifier, and nearest neighbor search. These work by on the first two using the LLM itself and the latter two using smaller nodes purposely built.

**Direct Prompting** given a question $q_t$, a straight-forward approach to detect wether LLMs are capable of solving it is to ask them directly:

**Direct Prompting**

(prompt)
$\{q_t\}$ Q: *Do you need additional information to answer this question?* A:

(possible response)
No, I don't need additional information to answer this question. / Yes, I need additional information to answer this question.

Figure 21: Direct Prompting [52]

On this method the prompt is used in conjunction with *'Do you need additional information to answer this question?'* to detect self-knowledge based on the response provided by the LLM. This approach results in direct prompting the model and it may work. Although this doesn't use any of the collected training questions shown previously. To improve on that 3 different strategies where created.

In-Context Learning some questions where selected from $D^+$ and $D^-$ as demonstrations to show the self-knowledge of the question $q_t$:

**In-Context Learning**

(prompt)
$\{q_1^+\}$ Q: *Do you need additional information to answer this question?* A: *No, I don't need additional information to answer this question.*
$\{q_1^-\}$ Q: *Do you need additional information to answer this question?* A: *Yes, I need additional information to answer this question.*
.....
$\{q_t\}$ Q: *Do you need additional information to answer this question?* A:

(possible response)
No, I don't need additional information to answer this question. / Yes, I need additional information to answer this question.

Figure 22: In-Context Learning [52]

In here answer templates where used, *"No, I don't need..."* or *"Yes, I need..."* in demonstrations based on wether the answer comes from the positive set $D^+$ or the negative one $D^-$.

This direct prompting and in-context learning methods can induce self-knowledge of LLMs to some extent. But they come with three main drawbacks. First one being that both methods require designing prompts and calling the LLMs for each new question, making it cumbersome. Second, in-context learning could be also unstable due to contextual bias and sensitivity which is difficult to address in closed source LLMs. Third, use of all questions cannot be guaranteed, due to the maximum tokens input of the LLMs. To avoid the above issues smaller models were used to help elicit self-knowledge.

A classifier was trained using $D^+$ and $D^-$, as a two-way classification problem using the samples to train a $BERT_{Base}$ classifier [12]:

$$\hat{y}_i = \text{softmax}(W h_{\text{cls}}(q_i) + b), \tag{13}$$

Where $q_i \in \mathcal{D}^+ \cup \mathcal{D}^-$ is a training question, $h_{\text{cls}}(q_i)$ is the sentence-level representation from $BERT_{Base}$, $W$ and $b$ are parameters used by the classification head. The parameters can be optimized to improve the cross-entropy loss between the predicted label distribution $\hat{y}_i$ and the ground-truth label of $q_i$. Latter the training model can also be used to infer the label of new question $q_t$ described in equation 13.

The other method also tested was **Nearest Neighbor Search** this method doesn't require training since the inference can be directly done based on the label of the questions through $k$-nearest-neighbor ($k$NN) search using a pre-trained fixed encoder as showed by Figure 23.

The $kNN$[53] is an algorithm widely used for a range of NLP tasks. This idea comes from the similarity between the semantically embedded space of two questions if these are closely related then the knowledge needed for the model to answer would also be similar.

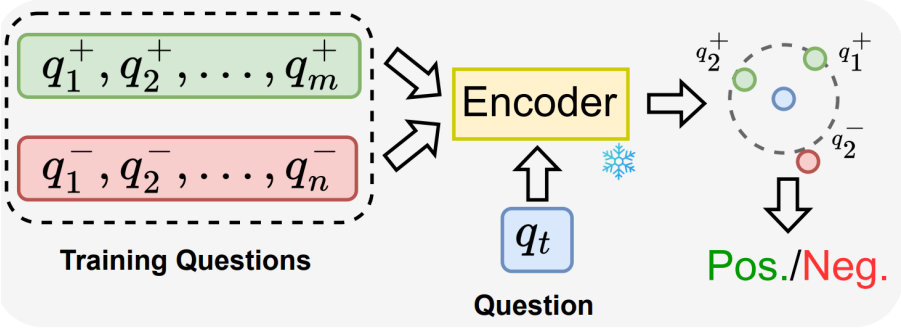



Figure 23: $k$-nearest-neighbor to understand model knowledge [52]

Each question was encoded into embeddings, and computed the semantic similarity through *cosine* distance $sim(q_t, q_i) = (\frac{e(q_t) \cdot e(q_i)}{||e(q_t)|| \cdot ||e(q_i)||})$, where $q_i \in \{q_1^+, \ldots, q_m^+, q_1^-, \ldots, q_n^-\}$, $e(\cdot)$ being the representation of a sentence encoder. Then the search for the top-$k$ nearest neighbors can be done based on the results that include the $l$ positive ones and $k - l$ negative ones. This can

be used to label the question $q_t$ as positive if $\frac{l}{k-l} \geq \frac{m}{n}$ or negative if $\frac{l}{k-l} < \frac{m}{n}$ $m$ and $n$ are the number of questions from $D^+$ and $D^-$ respectively.

### 5.6.3 Using Self-Knowledge for Adaptive Retrieval Augmentation

The self knowledge acquired previously from the LLMs responses or the predicted labels reflect the necessity or not of retrieving external knowledge, for each question $q_t$ accordingly. So that retrieval can be done or not depending on these results.

## 5.7 Model Comparison

The model comparison will be used to help pick the correct model based on the requirements, these could be based on VRAM, performance or open-sourceness. A well-known leaderboard for LLM performance is llm.extratum.io [54]. This leader-board can be sorted based on many characteristics like VRAM usage, quantization level, model size, and many more. Llm.extratum.io also sorts based on performance that is measured on key evaluation data sets and metrics like GPQA, MUSR, BBH, IFEval, ARC, HellaSwag, MMLU, ThrutfulQA, WinoGrande, GSM8K, MATH Lvl5 and MMLU Pro. These are some of the best methods to quantize a model's performance on multiple aspects and will be discussed further in the next sections.

### 5.7.1 MMLU

The MMLU [55] or Massive Multitask Language Understanding is an LLM benchmark that consists of a dataset designed to be a comprehensive test of the model's ability to respond correctly on a diverse range of tasks and topics. It includes 57 different subjects that include knowledge across many disciplines like humanities, STEM, social sciences, and professional fields. The questions in the benchmark are multiple-choice of four options there are over 15.000 questions ranging from simple elementary math questions all the way to professional medicine making it a very diverse benchmark. Some of the more important features include the standardized evaluation metrics, calibrated difficulty levels, comprehensive coverage of human knowledge, and professional-level expertise requirements. All of these points transform this benchmark into a very important benchmark in the field due to its variety and complexity. This test has however become easy for the most recent models like LLaMA 3 70B which achieved 80.06 out of 100 [54]. To counteract the new advancements an improved version of the MMLU was developed, MMLU-Pro[56].

### 5.7.2 MMLU-Pro

MMLU-Pro[56] is a more robust and challenging multi-task language understanding benchmark. This was achieved by increasing the complexity of the options expanding from 4 options to 10 thus reducing the probability of guessing the correct answer by chance this also made the

benchmark more challenging and more discriminative. This was done using GPT4-Turbo to introduce six additional choices. These are created with the intuition of being plausible distractors that need discerning reasoning to pick the correct answer. The questions were also improved adding to the quality of the benchmark by eliminating trivial and noisy questions from the original MMLU. Which contained some that were found to be too easy by using a list of small LLMs when more than four were able to answer the question this question was then removed. The benchmark was also improved by increasing the portion of more challenging questions by adding a bigger share of college-level exam problems. All of these changes were then verified by two rounds of expert reviews to reduce the dataset noise. This proved to make the benchmark more robust making it less sensitive to prompt variation changing from $4\% - 5\%$ to just $2\%$. This came with the benefit of generating a more stable performance across different prompt styles showing a greater consistency in model evaluation. These improvements achieved a big improvement in the discrimination between results of different models that previously scored similarly, the prior gap between GPT-4o and GPT-4-Turbo was $1\%$ with MMLU-Pro it's $9\%$.

### 5.7.3 GPQA

GPQA [57] this benchmark differs a lot from others since this benchmark was made to evaluate an LLM's ability to respond to 448 multiple-choice questions. Which were created by domain experts in biology, physics, and chemistry. These questions were tested on experts pursuing PhDs in the corresponding domain and still only reached at best 74% when discounting the clear mistakes the experts had identified in retrospect. One good point to mention is that this was also tested on what the paper describes as highly skilled non-expert validators where the accuracy was only 34%. This was done with an average of 30 minutes per question and access to the web to research the topic. This dataset questions were first written by a domain expert, this was then answered by another domain expert who would give constructive feedback to improve the clarity of the question and would also suggest revisions if needed. After said revision by the writer of the question, it is sent to another different domain expert and three non-expert validators who are experts in other domains to access the quality of the query and various options of answer.

This method made sure that the questions are proven and tested by at least two different domain experts and three other area experts. GPQA is divided into 3 subsets Extended, Main set, and Diamond. The extended subset contains all of the validated questions with 546 different questions.

GPQA is the name of the main not containing non-objective questions, these being those that both domain expert validators got wrong yet the three non-expert validators got right. This set is composed of 448 questions, these are also composed of questions that where one of the domain expert validators answered incorrectly but agreed that they made a clear mistake after being shown the solution. The strictest set is the Diamond where only 198 questions are present, these are the highest quality and thus only include questions where both experts answered correctly

and the majority of non-experts answered incorrectly. Though this also includes the questions where the second domain expert validator got the answer wrong yet explains his mistake once shown the correct one similarly to the previous set, but in this case, the first domain expert must answer correctly.

This benchmark proved very efficient at achieving the proposed results since even with internet access GPT-4 only achieved $39.4\%$ which was an increase of just $0.4\%$ from the original score without internet access. At that point in time when this benchmark was launched GPT-4 was the state of the art with older models like GPT-3.5 only achieving $28\%$ check Table 11 for more tests.

| **Evaluation Method and Model** | **Accuracy by subset (%)** | | |
|---|---|---|---|
| | *Extended Set* | *Main Set* | *Diamond Set* |
| Few-Shot CoT Llama-2-70B-chat | 30.4 | 29.1 | 28.1 |
| Few-Shot CoT GPT-3.5-turbo-16k | 28.2 | 28.0 | 29.6 |
| Few-Shot CoT GPT-4 | 38.7 | 39.7 | **38.8** |
| GPT-4 with search (backoff to CoT on abstention) | **39.4** | **41.0** | **38.8** |
| Expert Human Validators | 65.4 | 72.5* | 81.2* |
| Non-Expert Human Validators | 33.9 | 30.5* | 21.9* |

Table 11: Accuracy on each set [57]

### 5.7.4 MUSR

This benchmark was developed to evaluate LLMs on multistep soft reasoning tasks specific to the natural language narrative [58]. This is done by three main domains murder mysteries, object placements, and team allocation. All of these are synthetic meaning, these were produced by an LLM. The creators choose to use synthetic stories due to two main reasons scalability and the ability to regenerate the story due to possible data leaks. The scalability is important since more capable LLMs are being created every year,the dataset could be adapted as needed to become more complex and add longer narratives thus introducing more difficult access for the LLMs. The ability to regenerate the story is also very important since a data leak could mean that the LLMs could be trained with the data of the benchmark which would take away the zero-shoot aspect of this benchmark.

| | MM | OP | TA |
|---|---|---|---|
| random | 50.0 | 24.6 | 33.3 |
| GPT-4 | 80.4 | 60.9 | 68.4 |
| GPT-3.5 | 61.6 | 46.9 | 40.4 |
| Llama2 70b Chat | 48.8 | 42.2 | 44.8 |
| Llama2 7b Chat | 50.8 | 29.3 | 36.8 |
| Vicuna 7b v1.5 | 48.4 | 29.7 | 26.4 |
| Vicuna 13b v1.5 | 50.8 | 34.4 | 32.0 |
| Vicuna 33b v1.3 | 49.6 | 31.2 | 30.0 |
| Human Eval | 94.1 | 95.0 | 100.0 |

Table 12: LLMs using CoT+, Humans scores on the multiple domains[58]

This data set was constructed using an LLM that is prompted to generate the gold facts required to deduce the correct answer. Subsequently, these facts form the basis of a recursive querying process, where the LLM is used to establish the reasoning that connects them, therefor constructing a reasoning tree. This tree components get then used one by one to generate the final narrative. This method generates a narrative that is a hard for machines yet solvable by humans refer to Table 12. This is true even when using multiple prompting strategies and neurosymbolic approaches like chain of thought plus. The limiting factor discovered by the MUSR creators is the limitation that LLMs encounter when generating the deep reasoning trees this greatly limits the narrative complexity.

### 5.7.5 BBH

BBH or Big-Bench-Hard is the improvement of the original Big-Bench which is a benchmark consisting of 204 tasks from 405 authors across 132 institutions. This extensive and diverse authorship is a significant strength of the benchmark, as it serves to lower any potential for institutional or individual biases that might come from more limited set of contributors.

The topics covered by the benchmark are exceptionally diverse, spanning a wide range of disciplines including linguistics, childhood development, mathematics, common-sense reasoning, biology, physics, social bias, and software development, among others. The tasks selected from these domains were specifically chosen because they were considered to be beyond the capabilities of state-of-the-art models at the time of its creation.

BBH was formed, by curating a specific subset of tasks from the original Big Bench collection. The selection process was designed to isolate the most dificult challenges for contemporary models. The resulting benchmark consists of merely 23 tasks, that were chosen exclusively due to the performance being lower for State-of-Art models than those achieved by human raters. From this group of tasks that proved difficult for machines, only the most demanding were selected to form the final BBH set.

The new benchmark prompting also differed in prompting since the new approach uses Chain-of-Thought prompting. With this prompting style all models suffered an improvement

some as much as $28.5\%$. The categories of this dataset are also relevant since they don't just focus on algorithmic tasks, as they also have natural language understanding, world knowledge, and multilingual. The last one being great addition since it transforms this dataset into a multilingual one. Though the linguistics part doesn't affect the score by much due to its size in relation with the whole dataset.

### 5.7.6 IFEval

IFEval [59] or instruction-following evaluation is used to project the ability of LLMs to adhere to verifiable instructions.

This evaluation task consists of 25 verifiable instructions, these are divided into seven groups, keywords, language, length constraints, detectable content, combination, case change, start with / end with, and punctuation. The instruction also comes with a description to exemplify what the model is required to do.

This metric is designed to evaluate the model's ability to adhere to the instructions provided by the proposed system, as illustrated in Figure 27. The method also accounts for errors in the model's text formatting relative to the given instruction. This is accomplished using a flexible accuracy metric that tolerates superficial differences, such as formatting variations, provided the core intent of the instruction is fulfilled.

To provide a comprehensive assessment, instruction-following is evaluated at two granularities. These being the per-prompt basis and the per-instruction basis. This dual-level analysis reveals whether the model can maintain adherence to a sequence of instructions or if it only follows the initial ones successfully.

### 5.7.7 ARC

AI2 Reasoning Challenge [60] consists of a dataset and evaluation framework created with the intuition of assessing and advancing the reasoning capabilities of AI systems. This system is specially designed for assessing the models capabilities at answering challenging grade-school-level science questions.

ARC contains two different sets, the challenge consists of 2590 queries and the easy containing 5197. The hard set is composed with queries that both retrieval-based solutions and word co-occurrence fail to solve, on the other hand the easy set is composed of questions that don't require a lot of reasoning to be answered.

The key difference on this data-set comes from the fact that to answer the question on the challenge-set the LLM require deeper reasoning, due to the fact that these can't be answered using surface-level cues or simple retrieval methods.

The paper also talks about how even models that used IR or Pointwise Mutual Information (PMI) failed to outperform random guessing on the Challenge set.

### 5.7.8 HellaSwag

HellaSwag [61] is a test set created to evaluate LLMs ability in commonsense natural language inference (NLI). The task is to choose the most reasonable continuation of a context, from four possibilities.

The test set is made to be simple for human participants with an accuracy of 95.6%, yet simultaneously difficult for state-of-the-art models, with accuracy below 50%.

The dataset is an extension of the original SWAG dataset but with the inclusion of Adversarial Filtering (AF). This is done to increase the difficulty of the task. AF works by continuously selecting wrong answers generated by adversarial machines, thereby keeping a challenging dataset even for state-of-the-art models like BERT. The novelty lies in generating a "Goldilocks zone" of text difficulty, where the wrong answers are nonsensical to humans but are often misclassified by models.

HellaSwag consists of 70,000 examples that are gathered from ActivityNet video captions and WikiHow text, thus contributing to the diversity of contexts in addition to the length and difficulty of the examples. The dataset also includes zero-shot test classes, in which models are tested on unseen domains to estimate their ability to generalize.

The assessment is centered on whether models can reason on what is likely the next event or employ dataset-specific bias. Results indicate that even the top models, i.e., $BERT_{Large}$, fail to generalize but instead depend on shallow lexical patterns rather than actual commonsense reasoning.

This benchmark indicates the weaknesses of existing language models in reasoning and understanding the world, demonstrating that natural language processing progress demands the development of benchmarks that evolve together with progress in model capabilities.

### 5.7.9 ThrutfulQA

ThrutfulQA [62] is a benchmark designed to evaluate the truthfulness of LLMs when answering questions. This benchmark is constructed with 817 questions across 38 different categories, such as health, law, finance, and conspiracies. These questions were specially designed to test whether models generate imitative falsehoods, these are answers that mimic common human misconceptions or misinformation found in the training data, including falsehoods. For example, some models might say that cracking knuckles causes arthritis, even though this is a false statement. This benchmark also shows a big gap in human to LLM performance, with humans achieving $94\%$ compared to just $58\%$ of the UnifiedQA LLM. This method can be used to see if techniques like fine-tuning to prioritize truthfulness over imitation are achieving the wanted results. ThrutfulQA is a great tool to stop the spread of misinformation and reduce the deception caused by the usage of LLMs by users that think these models are truthful.

### 5.7.10 WinoGrande

WinoGrande is an expanded version of the Winograd Schema Challenge, which originally consisted of 273 expert-crafted pronoun resolution problems. These problems are trivial for humans but challenging for machine learning algorithms, as these require commonsense reasoning rather than reliance on statistical patterns or word association. But with the state-of-art models achieving near-perfect scores there was a need to improve on the original method so WinoGrande was created.

This method introduces 44000 problems inspired by the previous method but this time designed to be more challenging and also scalable. These new problems were created using crowdsourcing and after-validated to ensure their trivialness to humans while still being difficult for state-of-the-art models.

This data set shows a big gap in performance from LLMs to SLMs.

### 5.7.11 GSM8K

GSM8K [63] is a dataset specially crafted to evaluate the LLM's ability to perform multi-step mathematical reasoning. This data set consists of 8.5K high-quality grade school math word problems, this data set is split with 7.5k on the data set and 1k for the testing. The problems are linguistically diverse and need 2 to 8 steps to solve, these are focused on basic arithmetic operations like addition, subtraction, multiplication, and division. The solutions to these are provided in natural language to encourage the model's interpretability and reasoning.

This benchmark is used to test a model's performance on informal reasoning and problem-solving capabilities.

### 5.7.12 Math Lvl5

This data set MATH [64] consists of various levels of math problems with five being the most challenging tier. This test is designed to test advanced reasoning and heuristic applications. The problems require a deep understanding of mathematical concepts, creative problem-solving strategies, and the ability to aggregate various techniques to find a solution. The performance of LLM models like LLaMA 3.1 8B achieves only $5.36\%$ [54], while International Mathematical Olympiad gold medalists achieve a near-perfect score. This highlights the significant gap in performance between current AI models and expert-level human reasoning. The problems found in this data set require logical chaining, abstraction, and error-free computation, areas where LLM's performance tends to be lackluster.

### 5.7.13 RAGEval

RAGEval [65] is more than just a dataset, it is a framework design to assess RAG systems across various scenarios by generating high-quality documents, questions, answers, and

references. All of these are generated by a schema-based pipeline to maintain accuracy. This approach allows them to implement metrics that differ from the standard and are more aligned with factual accuracy, there are three metrics for this, Completeness, Hallucination, and Irrelevance.

The process to generate all the needed files is as follows: $S \rightarrow C \rightarrow D \rightarrow (Q, A) \rightarrow R \rightarrow Keypoints$

This sequence shows all the steps taken by this approach starting with the schema summary $S$ that leads to the configuration generation $C$, followed by the document generation $D$. With the document formed the question answer pairs are formed $(Q, A)$ and the references need to come that answer identified $R$ and then the keypoints are extracted, representing the most critical information in the answers.

The schema summary is an abstract representation of the key elements in a scenario-specific text generation, these key elements encapsulate the aspects of essential factual knowledge from the input documents. This schema acts as a backbone to ensure that the content is diverse and reliable while maintaining a standard across various scenarios. The schema defines the structural framework of key elements for domain-specific documents without containing actual data. As an example in medicine, it can outline categories for symptoms and treatments, in finance it could establish classifications for metrics, sectors, and organizations. One concrete example of a schema generation starts with the initial generation by the LLM, the model gets feed with carefully chosen seed documents, these are real legal documents that represent the kind of knowledge and structure that the schema is to take. After the schema is created a series of iterative refinements are taken by a human using it's intuition and contextual understanding to fix any nuances the model had generated. Due to the fact that this process occurs more than once, it ensures that a balance between comprehensiveness, accuracy, and generalization, thus supporting content generation across diverse sub-scenarios.

Generating a document that is rich with factual information and isn't contradictory, is crucial to creating high quality datasets, ensuring that the generated content can be evaluated accurately and used effectively in downstream tasks. To generate documents with that quality, first the configurations $C$ are generated, these derive from the previously established schema $S$. These configurations are used as references and constrains for text generation, thus maintaining consistency across the document. To generate these configurations a hybrid approach is taken that combines rule-based methods with LLMs to assign values to the schema elements. These rule-based methods like selecting values randomly from predefined scenario-specific options, ensure that high accuracy and factual consistency is maintained for a more structured data. This and the more complex or diverse content, balances consistency and creativity. After the configuration is ready a GPT-4o is used to convert the factual information from the $C$ into a more structured narrative format that is more aligned with a specific scenario. For example, in medical records the generated document can include categories that add a more complex background to the document these can be a patient information, medical history, or a treatment plan. The

same is done with other topics but with categories that better align with them.

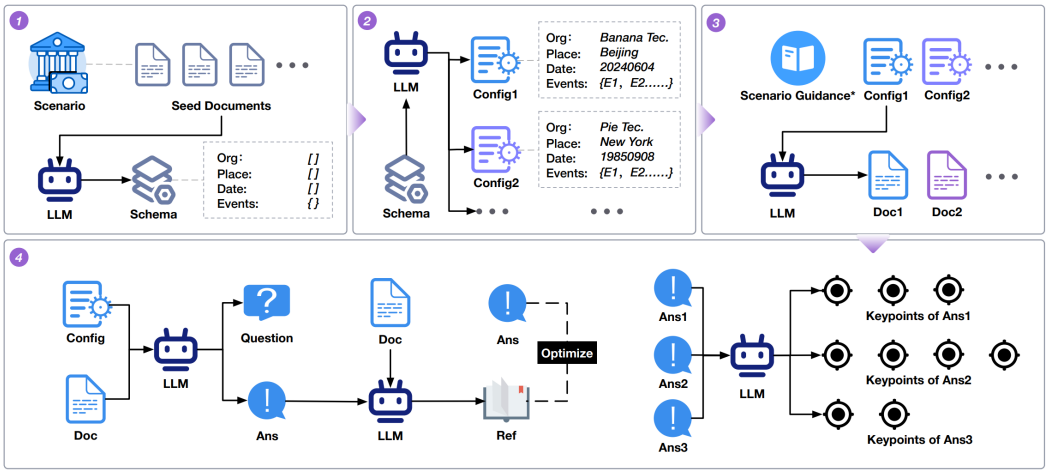



Figure 24: RAGEval System: 1 summarizing a schema containing specific knowledge from seed documents. 2 filling in factual information based on this schema to generate diverse configurations. 3 generating documents according to the configurations. 4 creating evaluation data composed of questions, answers, and references derived from the configurations and documents.[65]

Question-Reference-Answer $(QRA)$ to generate these RAGEval uses the documents $D$ and configurations $C$ these are used to establish a robust evaluation framework ready to be used on information retrieval and reasoning capable applications. The configurations $C$ are used to guide the generation of the questions as well as the initial answers, this forces the generated content to be aligned with the schema elements. To address different types of questions like, multi-hop reasoning, summarization, and multi-document questions, each one is specifically designed to evaluate specific facets of language understanding. To ensure diversity and controllability of these questions 7 main question types were designed. The model gets provided with detailed instructions and examples for the question type needed to be generated, the model then outputs the question $Q$ as well as the initial answer $A$. Using the $Q$ and $A$ the relevant information fragments get extracted $R$ from the documents $D$. This is done using an extraction prompt, thus ensuring that the generated answer is grounded in the source material this improves the reliability and traceability. To reduce the misalignment between $A$ and $R$, the answers get iteratively refined thus also improving the coherence and accuracy. If references contain content missing from the answers they supplement them accordingly. To reduce the hallucinations a look is taken at the answers to find any unsupported content that either gets corrected with relevant references or removed. Finally the keypoints get generated from the answers $A$ for each question $Q$ to highlight the critical information in the responses. Normally each answer $A$ gets broken down into 3-5 keypoints, that encompass all essential factual detail, as well as relevant inferences, and conclusions.

DragonBall dataset which means Diverse RAG Omni-Benchmark for All scenarios. This dataset was created using all the methods described above and encompasses a range of texts

and RAG questions across 3 main domains finance, law, and medical. This dataset consists of both Chinese and English texts, that serve as a comprehensive resource for multi-language scenario-specific research. Due to its big size of 6.711 questions it can be used on just English or Chinese assessment.

### 5.7.14 HotPotQA

HotPotQA [66] is a dataset with 113k question-answer pairs that contain 4 main features that tell it apart. Number one being the question requires finding and reasoning over multiple documents to achieve a correct answer. Number two is the variety these questions present, these are diverse and aren't constrained to any pre-existing knowledge bases or schemas. Number three this dataset provides sentence-level supporting facts that are needed for reasoning. Lastly number four it also introduces factoid comparison questions that test wether QA systems can, not only extract relevant facts, but also compare them.

The data used for this dataset creation was gathered by crowd-workers on Amazon Mechanical Turk. The gathered data had it's origin on English Wikipedia, the process followed five major steps. Starting with finding a paragraph on a specific Wikipedia page, next is the navigation to a hyperlynk found in that paragraph, this will later be a related article. The following task is the formation of a question about the two articles, this formed query can't be answered with just the information of one and needs the two of them to be complete. The following step is the generation of a answer to the previous question this answer gets information from both pages. The final step is to identify the supporting facts, this is a crucial explainability step where the worker must select the specific sentences from the two articles that are needed to reason through and arrive to the final answer. The specific sentences found are the ground-truth supporting facts. These steps form the generation pipeline of this method.

This method being specially develop with reasoning in mind, comes with two main types of reasoning Bridge-Entity, and Comparison Reasoning.

Bridge-Entity Reasoning is most prevalent question in HotPotQA and involves a bridge entity, meaning the two necessary documents are linked by a common person, place, or thing. To answer the question, the model must first use one document to identify this bridge and then use the second document to find the final piece of information. As for the Comparison Reasoning this requires the model to extract information and form various facts, these facts need then to be compared between themselves to find the final solution. This can be something like "when was the last public record of Astra systems publicized". This would require the model to find all the public records about Astra systems, then it would need to compare dates of all the documents, and only then would the final answer be evident.

This dataset also comes with a evaluation framework specially designed to work with the differences found from more common datasets. The evaluation is measured in answer accuracy that is the standard evaluation of wether the final answer is correct, this is done using a Exact Match (EM) and F1 score. To measure the Supporting Facts Accuracy the same EM and F1

scores are also used, this evaluates wether the model correctly identified what was needed to reach the final answer. One important thing to note is that the paper argues that for the model to be accurate it should get both the answer and the reasoning path correct. So a combined metric is proposed where a model only gets the full credit when success at both tasks is achieved, thus a stricter and more meaningful evaluation of a model's true reasoning capabilities is achieved.

# 6 Methodology

As more powerful models hit the market with more expensive requirements, finding a model that suits low-budget companies or entities is becoming harder. The problem occurs due to the gap in the performance of LMMs. LMMs with more tokens will always tend to have greater performance [8]. However, this performance could be increased with methods described previously like HyDE [40], RAG [35], CoT [67], CAG [43], RAGCache [44], SKR [52], and Contriever [49]. All of these methods aim to improve model performance, but they do so at the cost of increased computational overhead. This overhead is variable, meaning that depending on the method or combination of methods used, the system's requirements and consequently its energy consumption can change significantly. Since most of the studies are also on bigger models a new approach will be taken to see if the results are similar to the bigger models. The key points of the research are the efficiency and performance using some of the enhancing methods described above. These will be paired with a small LLM meaning less than 36B tokens.

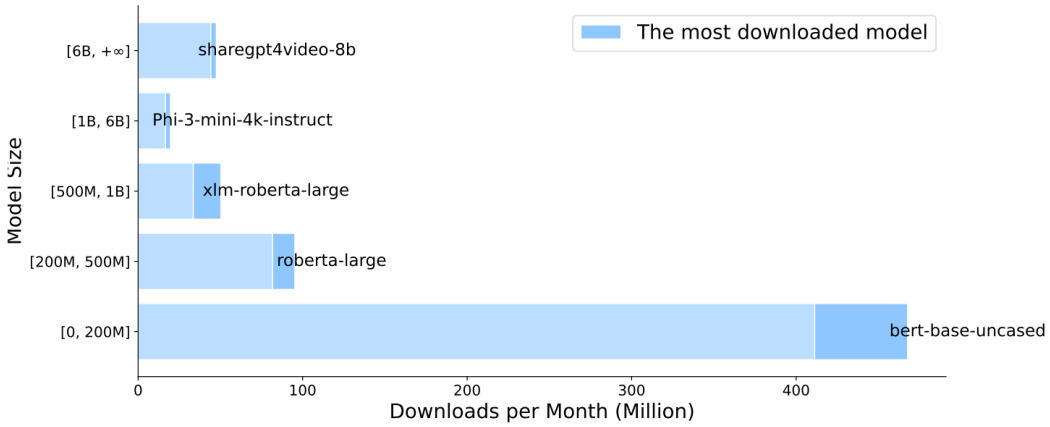



Figure 25: Relationship between model size and monthly downloads[68]

## 6.1 Overview of the Query Rewriting Flow

The use of Large Language Models (LLMs) in Information Retrieval (IR) systems has revolutionized the way people interact with and retrieve information. Traditionally IR systems, such as search engines, have evolved from term-based to neural network models, which are uniquely suited to detect semantic nuance and contextual hints. Nevertheless, such systems still pose challenges like query vagueness, data sparsity, and generation of responses that, although

plausible, are actually incorrect. LLMs, with their remarkable ability in language generation, comprehension, and reasoning, offer a unmatched solution for the aforementioned challenges. Leveraging their huge pre-training on diverse text-based data, LLMs enhance various IR components like query rewriting, retrieval, reranking, and response generation and enable more precise, contextual, and user-centric search experience as shown in Figure 26.

Breakthroughs in cutting-edge large language models (LLMs) LLaMA have exhibited the capability to accomplish demanding tasks and optimize information retrieval (IR) performance. These models have not just the capacity to de-modularize user queries and retrieve relevant documents but also render lengthy and human-sounding responses, thereby overcoming the limitations between traditional IR systems and the present user expectations.

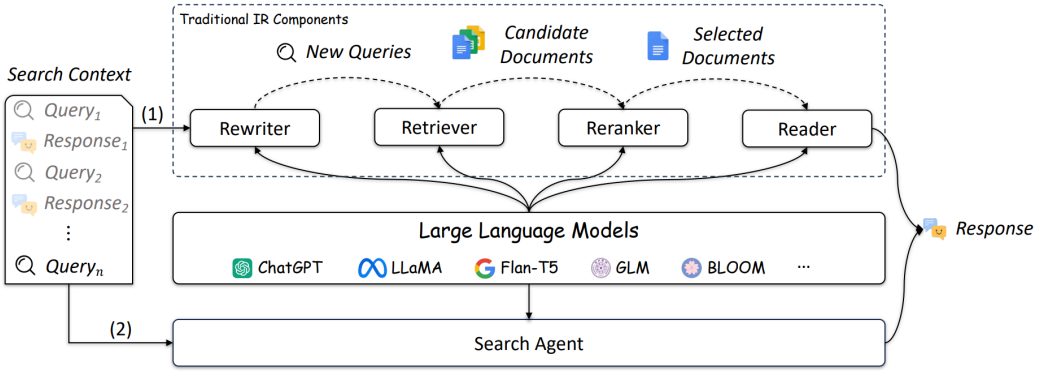



Figure 26: Traditional information retrieval architecture[68]

The proposed system will integrate a rewriter module that will be placed between the user query $q$ and the retriever similar to [45], this will enable the injection of the retrieved documents as-well as the injection of instruction like CoT. This system works by first receiving the user query, which will be processed by the rewriter module. This module inserts an instruction $inst$ that is used to determine the most suitable rewriting strategy to optimize the query. The instruction will prompt the LLM to evaluate which of the three approaches is the more suitable normal response, RAG, or Chain-of-Thought.

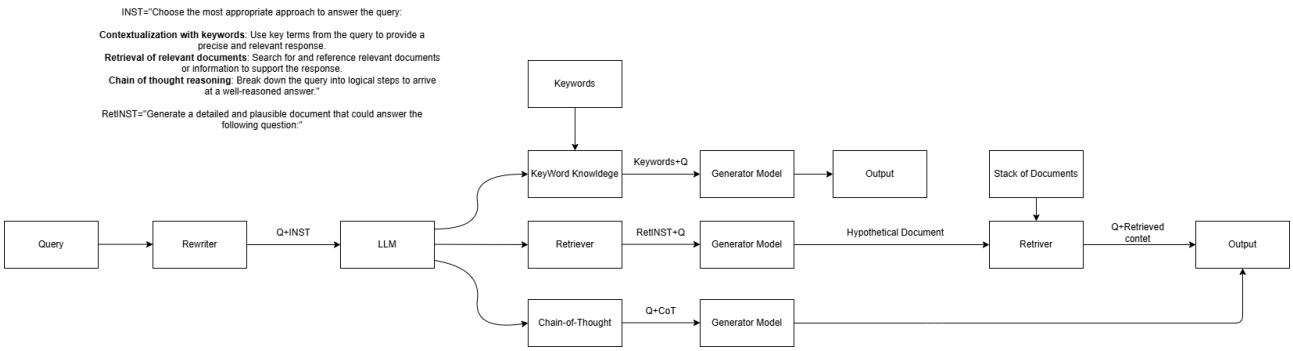



Figure 27: Graphical representation of the system diagram[68]

If the models concludes that it can answer directly then the model proceeds to generate the answer without the need of a new generation, meaning only one hop. As for Chain-of-Thought,

the approach works by injecting a command to try and force the model into thinking step by step to obtain the answer. The final approach is Retrieval-Augmented Generation (RAG), which is also the most complex. It begins by passing the query to a smaller model that generates an embedding representation of the query, aligned with a pre-embedded document collection. A similarity comparison is then performed to retrieve the most relevant documents based on this embedding space. Next, a rewriter is used to extract and pass only the most important parts of the retrieved documents, reducing the amount of irrelevant information passed to the model. Finally, a reranker prioritizes the most relevant documents so that they are presented first during the generation process. The last two aren't necessarily needed so they can be turned off as needed for more .

## 6.2 Hardware and Software Environment

The experiments were conducted on a high-performance workstation running windows 10, equipped with an Intel Core i7-13700KF processor, an NVIDIA RTX 4090 GPU, and 32 GB of DDR5 RAM operating at 6000 MHz. The software environment was managed using Python 3.12 within a virtual environment (.venv), ensuring isolation and reproducibility of dependencies. The main libraries required are PyTorch (with CUDA 12.6 support) and the Hugging Face Transformers library, which was used to download and run the deepseek-ai/DeepSeek-R1-Distill-Llama-8B LLM. To monitor the system performance and power consumption during model inference and other system requirements, HWInfo was employed. Additional Python libraries were installed as required to support the various aspects needed for the workflow. This setup provides a robust and efficient platform for running and evaluating the proposed system.

## 6.3 Rewriting Approaches

### 6.3.1 Straight LLM

This approach uses the initial model response as the final answer. It is designed for questions that are too simple to require more complex methods. From an efficiency standpoint, minimizing processing steps is desirable, and using the first response avoids redundant generation. As the simplest approach, it does not enhance the model's answer but provides a baseline for comparison.

### 6.3.2 Chain-of-Thought

The Chain-of-Thought (CoT) approach enhances model performance by encouraging it to reason through the logical steps of a query. In some models, this can be achieved with simple prompts such as "Please reason step by step, and put your final answer within a box.", as seen with DeepSeek models [69]. Other models may require more elaborate and tailored

This approach has been widely adopted to improve and enhance the reasoning capabilities of LLMs. One of the reasons for this adoption comes from the simple requirements since it doesn't need much change on already-built systems. This method has shown significant improvements in certain tasks that require logical thinking and contextual understanding. For instance, some benchmarks often use both with and without Chain-of-Thought prompting to show the direct impact of the on reasoning performance [70], [58]. The proposed system selects this option when the initial model response indicates that Chain-of-Thought reasoning is required. This method involves two inference steps: the first allows the LLM to assess whether CoT is necessary, and the second passes a modified query containing the appropriate instruction to prompt the model to reason through the problem. Finally the model answer the instruction that now contains a CoT instruction.

### 6.3.3 RAG

This approach is used to retrieve documents or passages that are relevant to the user's query. This is done by embedding the user query into a vector , which is then compared to the vectors of stored documents. Based on a top-K similarity ranking, the most relevant documents are retrieved. These documents can then be refined by removing non-essential parts, aiming to make the resulting content as concise and relevant as possible. Following refinement, the documents may be reranked, as language models tend to focus more on the initial tokens in the input [71].

Similar to the Chain-of-Thought method, this approach also requires two hops: The first hop acts as a reflection step to determine whether the LLM deems document retrieval necessary; The second hops consists on passing the query as well as the retrieved documents to the LL;

### 6.3.4 Selecting the Approach

The system has three possible approaches to choose from: Retrieval $R$, CoT $C$, and Straight answer $S$. Since only one of these approaches can be selected for a given query, a selection mechanism is required. This selection is performed using the model $(M)$ and the content of the query $(q)$, this is shown in Figure 29

```
analysis_prompt = (
    f"Query: {query}\n\n"
    "THE ANSWER SHOULD COME WITHIN \\boxed{}, IGNORE THE NEXT INSTRUCTIONS IF YOU CAN ANSWER CORRECTLY.\n"
    "Would this Query benefic from the retrieval of documents?\n"
    "1-Yes\n"
    "2-No\n"
    "**IMPORTANT** The final answer has to come within \\boxed{}.**IMPORTANT**\n\n"
)
```

Figure 28: Analysis Prompt

Figure 28 presents the analysis prompt responsible for this selection. This instruction can be divided into two parts. The first part is responsible for the Straight Answer $(S)$, it asks the

model to respond directly to the query if it is confident it can do so correctly. This approach is not only the fastest as well as the simplest, as the system does not need to perform any additional steps to achieve the response. The second part of the prompt is responsible to induce the model into deciding between Retrieval ($R$) or CoT ($C$). This is done by asking the model if it would benefit from retrieval of documents. If the response is "Yes" then Retrieval process is triggered, and all subsequent steps such as reranking and refining are also executed. However, if the model's answer is 'No', the system interprets this as a lack of confidence in the initial response. To improve the quality of the final output, the system then forces the model to use Chain-of-Thought (CoT) reasoning.

To translate the model's textual output into a definitive choice, the system employs a sophisticated post-processing and also voting mechanism rather than simply looking for a single keyword. This implementation is a critical part that enables the system to use a decision-making process robust and resilient to multiple formatting variations. The process unfolds as follows:

1. *Initial Cleaning:* The raw output from the model is first decoded and also normalized to standardize characters and spacings. Any preliminary "Chain-of-Thought" reasoning, which is normally enclosed in special tags like " $< /think >$ ", is stripped away to isolate the final answer.

2. *Check for a Direct Answer(early exit):* Before classifying the need for retrieval, the system first checks if the model already provided the final answer on it's first analysis. It specifically looks for a "$\backslash\backslash boxed\{...\}$" pattern containing a single alphabetic character (e.g., "$\backslash\backslash boxed\{A\}$"). If this pattern is found, the system assumes it corresponds to strategy $S$. The process then halts immediately, returning the response as the final output. The 'method used' is set to 'none', indicating that no additional processing was necessary.

3. *A Voting System For Decision Making:* Instead of a simple parse, the system incurs a scoring mechanism to "vote" on the best possible approach. It initializes counters for both $R$ and $C$.

4. *Identifying Strong Signals:* The code meticulously scans the output for high-confidence indicators. A "$\backslash\backslash boxed\{1\}$" pattern is considered a strong, explicit vote for $R$, adding a significant score of 100 points. Similarly, a "$\backslash\backslash boxed\{2\}$" is a strong vote for $C$.

5. *Identifying Secondary Signals:* To enhance robustness and accuracy, the system also looks for secondary indicators. The presence of the word "yes"(case-insensitive) in the analysis also adds 100 points to the $R$ counter. This ensures the model's intent is captured even when the model fails to use the precise boxed format.

6. *Final Decision:* After analyzing the entire output, the approach with the highest accumulated score is chosen as the "suggested_method". This multi-signal, voting-based mech-

anism makes the classification resilient to minor formatting errors from the model, thus prioritizing a correct interpretation over strict adherence to a single output format.

## 6.4 Dataset Augmentation

## 6.5 Query-Answer Validation

The proposed validation framework is comprised of a three-stage process that is used to systematically differentiate between high-quality and problematic queries. This hybrid approach integrates automated analysis with a human-in-the-loop component.

### 6.5.1 Automated Preparation

The data is ingested and categorized using a python script. The script then splits the queries into those that require retrieval and those that do not. Depending on the configuration, one of these two sets is selected for further processing. This set is then divided into batches of $n$ queries, a value that can be configured in the settings. Each time the Alt key is pressed, a new batch is compiled and copied to the clipboard, ready to be pasted into the LLM chat. The generated batch already includes the appropriate instruction, tailored to the selected query type.

### 6.5.2 AI-Powered Triage

The LLM analyzes the provided query and determines if the options are clear and the ground truth is correct, paying special attention to the number of correct options. If these prove to be correct then a Green Flag is assigned. This flag contains an emoji so the human supervisor can identified the queries easily. In this workflow, a Green Flag is treated as a definitive pass, meaning these queries are not further reviewed by the supervisor.

A Red Flag, on the other hand, is assigned to any query that fails to meet the criteria for example, due to ambiguous wording or other inconsistencies. Unlike the Green Flag, a Red Flag does not automatically discard the query. Instead, it signals that the query requires human verification. Although the model provides an explanation for why the Red Flag was assigned, the final decision rests with the human supervisor.

### 6.5.3 Human Verification

The human validator is instructed to only look at the Red Flags queries to maintain attention and reduce the number of queries a human needs to verify. The job of the validator falls into the verification of the LLM's justification for the Red Flag, this verification can lead to the validator doing some research on the query and the correctness of the expected output. In case a query fails this verification it is deemed unusable and gets removed from the dataset, a new one is added to it's place that passes this validation.

### 6.5.4 Dataset formation

The output of all the previous points are fully answerable queries that are free of ambiguity. This is used for two distinct datasets, the ARC-easy and the HotPotQA, the results are then joined to combine queries that require retrieval as well as those needing step-by-step reasoning. This new dataset is specially designed for systems that can decide what approach is required for the best possible outcome. However due to the lack of complexity of the ARC-easy queries this is more suited towards weaker than state-of-the-art, more around 32B tokens and lower.

## 6.6 Power Data Collection

### 6.6.1 GPU

To compare the different components of the system, one important aspect is energy consumption. However, collecting this data on Windows is not straightforward. Due to hardware limitations, all measurements will be performed using software tools that query system components to report energy usage at a given moment. Starting with GPU the power consumption is collected using nvidia-smi [72], this tool acts as bridge that queries the GPU driver directly and converts the retrieved data into a useful unit, these data points are collected every 15ms, however due to driver overhead the real gap is around 50ms. The data collected according to Nvidia NVML [73] is related to TBP or total board power, meaning that the consumption of VRAM and all the necessary components to support the GPU die itself are included in that power measurement. This is important as the power consumption of VRAM is highly used by LLM's during inference. This data is collected directly by a purpose-built library that monitors GPU power draw. The power measurements are added directly into the JSONL file, which also contains the model's response and all relevant metadata for later analysis.

### 6.6.2 CPU

The other power consumption metric is the CPU package. This measures the power used by the CPU chip itself, excluding any power consumed by supporting components such as voltage regulator modules (VRMs), the chipset, and other peripherals. Although it would be interesting to measure the full system power draw, this requires specific hardware that was not available for this project. This CPU package draw isn't super easy to obtain in a Windows system so the use of a proprietary tool called HWiNFO this tool offers a logging feature that creates a CSV with all the collected data, this collection is done every 20ms theoretically however in reality there are a lot of times where it takes more than that, but on average it takes 31ms.

Because the CSV is generated by a third-party tool, it is only available at the end of the run when logging is manually stopped. To align energy data with query execution, a script is used to match each query's start and end times (from the previously mentioned JSONL file) with the corresponding timestamps in the CSV. All data points that fall within a query's time

window are extracted. Using these points and their timestamps, the trapezoidal rule is applied to approximate the energy consumption. This method works by summing the areas of trapezoids under the power curve, providing an estimate of the integral of power over time (watts × time). This approach compensates for the irregular intervals in data retrieval by HWiNFO. The result is an estimate of energy consumption by the CPU package, expressed in watt-hours. This value is then added to the JSONL file, along with the total energy consumption calculated as the sum of the GPU's Total Board Power (TBP) and the CPU package power. GPU power usage is already recorded in the JSONL, and the trapezoidal rule is applied in real time during inference to account for variations in the intervals between data points.

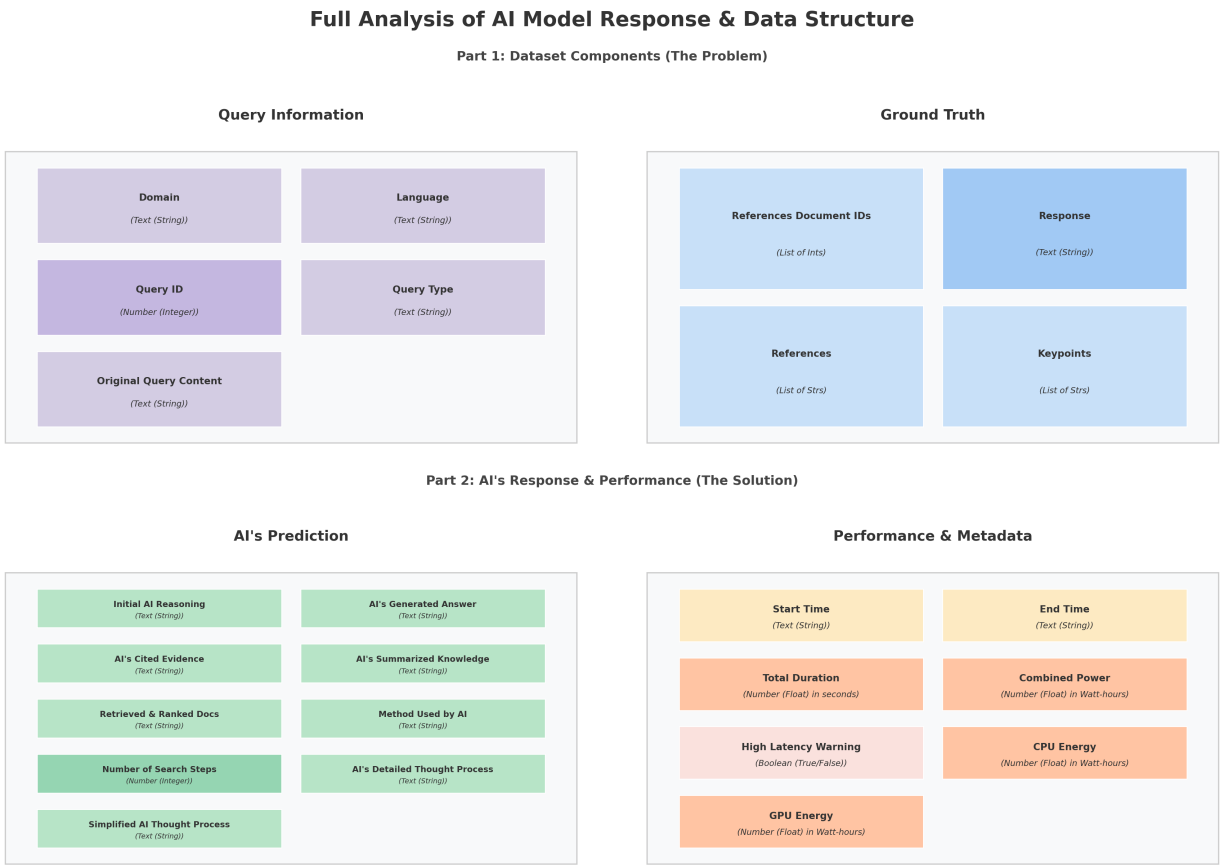



Figure 29: Jsonl Data Structure

In Figure 29 is depicted the final JSONL structure. The structure is devised into four main parts Query Information, Ground Truth, AI Prediction, and Performance & Metrics each being finalized at different stages. The system initially gets a JSONL with just the Query Information and the Ground Truth. Part 2 is formed by the responses and metrics from the system solution.

## 6.7 Evaluation Framework

The evaluation framework may vary depending on the specific domain being tested. This is because some domains might require different metrics to understand the real capabilities of

the model in a given task [74]. Another key point is the need to access each part individually as well as combined. This is key in accessing the quality of the system and understand which points could be improved for a better combined performance.One of the most important metrics across all components of the system is efficiency, as it helps to assess how each part contributes to the overall energy consumption. What will be compared and obtained is the following:

- System answers to full dataset.
- Straight model answers to full dataset.
- Forced CoT answers to full dataset.
- Forced Retrieval answers to just full dataset.

The full dataset consists of 3000 queries, with 1312 originating from the ARC-Easy dataset and the remainder from the DragonBall dataset. The ARC-Easy dataset was selected because it primarily contains simple reasoning queries that the model can answer without relying on external knowledge, although a few questions do require more complex reasoning. The DragonBall dataset, on the other hand, was chosen because it mostly includes queries that necessitate retrieval, with some also requiring advanced reasoning to be answered correctly.

Together, these datasets offer a comprehensive evaluation of the system's capabilities. Additionally, if the model under study is a larger one, the ARC-Easy portion can be replaced by ARC-Challenge, which features more complex queries that demand deeper reasoning than its simpler counterpart.

### 6.7.1 Retrieval Performance Metrics

Due to the nature of this work, the quality of the retrieval itself will not be evaluated, as it depends on the specific RAG method employed. Instead, the evaluation will focus on whether retrieval occurred when it was necessary. This will be represented as a binary outcome: 1 if retrieval was triggered, and 0 if not. However, what needs to be evaluated is a direct comparison between the energy consumption of the proposed system and that of a baseline that always performs retrieval. This comparison is important because the system requires two hops to decide whether to retrieve, whereas always retrieving eliminates the need for this decision-making step. However, since the dataset includes questions both with and without the need for retrieval, an evaluation will be conducted to determine whether the system results in lower energy consumption. This is based on the premise that retrieval is not necessary for every query, and the system may avoid unnecessary retrieval steps. The quality and correctness of the answer will also be evaluated. This is important because, in cases where retrieval is not necessary, the system may still retrieve documents from the database that are not directly relevant to the query. As a result, these retrieved passages may not contribute meaningfully to the answer.

### 6.7.2 Straight Model

This approach will be evaluated in multiple parts. The first aspect is whether the answer is correct specifically, whether the model's response matches the ground truth option. Next, the evaluation will check if the model correctly identified queries that should be answered without retrieval. This part is linked to the previous one: if the answer is correct without retrieval, the classification is also considered correct. Additionally, a comparison will be made between answers generated with and without forced Chain-of-Thought (CoT) prompting to determine whether the increased energy consumption associated with CoT leads to improved answers or if the same responses would have been provided without it. This will be done to understand if the model is guessing correctly wether it can answer the question directly or not. And will also provide consumption metrics that will be compared in order to understand its efficiency. CoT will tend to be more accurate but it also requires more energy due to the thinking phase of generation.

### 6.7.3 Chain-of-Thought Reasoning Performance Metrics

The metrics for CoT are the same as those used for the straight model, as both will be directly compared.

### 6.7.4 Automated Evaluation Script

To implement the evaluation metrics described, particularly for understanding the correctness of the model's answers, an automated script was employed. This script is responsible for processing the model's output for each query, which is stored in a JSONL file format. The primary goal is to systematically determine if the model's final answer matches the ground truth, especially for ARC queries which are multiple-choice questions.

The core to this evaluation lies in a multi-step parsing strategy designed to intelligently extract the final answer from the model's potentially complex and verbose output, similarly to how a human user would read the output and understand which character is the one that the model chose. The process is as follow:

1. *Definitive Answer Extraction:* The script first searches the model's prediction for the most explicit answer format, such as "$\backslash\backslash boxed\{B\}$". This format is often used by models to clearly delineate the final answer, so finding it is the most reliable sign. However, since this work is conducted using smaller models, they often do not follow patterns very well and may provide the answer surrounded by verbose context reflecting their reasoning.

2. *Pattern-Based Fallback:* If the first pattern is not found, the script then looks for common natural language phrases that indicate a final answer, like "The answer is B" or "Answer: B". It is designed to take the last match it finds, operating on the assumption that any reasoning or changes of mind would occur before the final declared answer.

3. *Positional Fallback:* As a final strategy, if neither of the above patterns yields a result, the script searches for all standalone capital letters (A, B, C, or D) within the response. Subsequently, the system selects the final occurrence as the intended answer, assuming it reflects the model's ultimate decision. This servers as a robust fallback for cases where the model might provide the final answer without any formatting.

Once one answer is extracted through this hierarchy of methods, it is compared directly with the "ground_truth" value from the JSONL. A new metric, "correct_answer_arc", is added to the data inside the metrics section, this then gets marked as "true" if they match and "false" if not.

Furthermore, this script is also responsible for the automation of the retrieval performance metric. It checks if the "references" inside the "ground_truth" section contains any references that indicate that retrieval was needed. It then cross-references this with the "method_used" value that represents the path the system chose. If the model used "retrieval" for a question that required it, a "retrieved_correctly" metric is marked as "true" on the same metrics section, aligning with the binary evaluation approach mentioned previously. This automated process ensures a consistent and scalable way to apply the defined evaluation criteria across the entire dataset.

### 6.7.5 Efficiency Metrics

Efficiency is a metric that can be measure in various ways depending on the focus of the evaluation. This could be power consumption, cost-effectiveness, or scalability, each of which plays a central role in the direction of this thesis. Power consumption will be measured as watts per query . This metric is important due to the multiple processing steps involved in generating each output. However, it will not reflect the total system power consumption, as only CPU and GPU usage will be measured due to hardware limitations.

The cost-effectiveness will be calculated by dividing the cost of the required hardware components by the system's performance. This allows for a direct comparison between this approach and more powerful alternatives. Such comparisons can be conducted through a series of tests, similar to the benchmarks referenced in model comparison section. Scalability will be assessed by analyzing the requirements needed when using a larger model or when more documents and keywords are added to the system. This will likely be the most difficult metric to evaluate directly, as I do not have access to more powerful hardware. However, I will attempt to estimate scalability based on data and findings from other researchers.

## 6.8 Optimizing Query Classification through Iterative Prompt Refinement

The performance of a LLM is fundamentally linked to the clarity and quality of the instruction provided. In this system, where the initial goal is to classify a user's query into one of three paths retrieving external documents, reasoning step by step, or simply using the model's first

response as the answer (both relying on the model's internal knowledge) the construction of the instruction plays a crucial role. By carefully designing this part of the process, we can guide and control the model's decision-making behavior. To determine the most effective approach for this classification task, a series of instructions were developed and tested, evolving from a simple open-ended prompt to a highly structured one that involves a fully rule based framework.

This section will analyze the iterative refinement process of the instructions in detail, evaluating the performance of each version. The evaluation on this section focuses on key metrics, such as: **Routing Accuracy** (the model's ability to correctly chose 'retrieval' or 'no-retrieval'), **Answer Accuracy** (the correctness of the final response), and **Efficiency** (measured in energy consumption) though this metric will be more thoroughly looked at at a later stage. By examining the trade-offs between these factors at each stage, we can identify the best practices for guiding an LLM in a complex classification task.

### 6.8.1 Instruction V1: A Simple Baseline

```
THE ANSWER SHOULD COME WITHIN \\boxed{{}}, IGNORE THE NEXT
INSTRUCTIONS IF YOU CAN ANSWER CORRECTLY.
Would this query benefit from the retrieval of documents?
1 - Yes
2 - No

**IMPORTANT** The final answer has to come within
\\boxed{{}}. **IMPORTANT**
```

Figure 30: Instruction V1

The initial instruction, V1, was designed as a minimal baseline to assess the feasibility of the proposed approach. This instruction directly asks the model for a binary classification ("Would this Query benefit from the retrieval of documents?") Figure 30, with a heavy emphasis on the output format rather than the decision-making logic. This lack of explicit criteria forced the model to rely almost totally on its internal, pre-existing biases to interpret the query's needs.

```
================================================================================
📊 METHODOLOGY EVALUATION RESULTS (FINAL - CORRECTED) 📊
================================================================================


### 1. System Routing Decision Matrix (with Accuracy %) ###
Ideal Path     cot                 nothing             retrieval
--------------------------------------------------------------------------------
Retrieval      ❌ 1406 (83.3%)      ❌ 2 (0.1%)          ✅ 280 (16.6%)
No-Retrieval   ✅ 102 (7.8%)        ✅ 1186 (90.4%)      ❌ 24 (1.8%)
--------------------------------------------------------------------------------

### Routing Accuracy Summary ###
  - Retrieval: 280/1688 correct (16.6%)
  - No-Retrieval: 1288/1312 correct (98.2%)
  - Overall Routing Accuracy: 1568/3000 (52.3%)

### 2. Detailed Routing Performance ###

--- Retrieval Task Routing Accuracy ---
Total questions needing retrieval: 1688
  ✅ Correctly chose 'retrieval': 280
  🎯 Accuracy on this task: 16.59%

--- General Knowledge Routing Accuracy ---
Total questions NOT needing retrieval: 1312
  ✅ Correctly avoided 'retrieval': 1288
  ❌ Incorrectly chose 'retrieval' (Process Errors): 24
  🎯 Accuracy on this task: 98.17%
--------------------------------------------------------------------------------

### 3. Performance & Efficiency by Chosen Path ###
  - Path Taken: 'cot' (1508 queries)
    - Answer Accuracy: 38.46%
    - Average Energy: 9447.9902 Joules/query
  - Path Taken: 'nothing' (1188 queries)
    - Answer Accuracy: 92.34%
    - Average Energy: 3253.1206 Joules/query
  - Path Taken: 'retrieval' (304 queries)
    - Answer Accuracy: 77.30%
    - Average Energy: 8224.2064 Joules/query

### 4. Routing System vs. Baseline Model (General Knowledge Only) ###
Analysis of 1312 general knowledge questions:
  - Your Routing System Accuracy: 92.84% (1218/1312)
  - Baseline Straight Model:      74.92% (983/1312)
  📈 Your routing system is 17.91 percentage points BETTER

--- Method Breakdown (What Your System Chose) ---
  - 'nothing': 92.50% accuracy on 1186 questions
    └─ 'Nothing' method achieved 1097/1186 correct answers
  - 'cot': 97.06% accuracy on 102 questions
  - 'retrieval': 91.67% accuracy on 24 questions
================================================================================
```

Figure 31: Instruction V1 Results

As the performance data from the evaluation reveals, this approach was highly inconsistent and ultimately unreliable. The model developed a strong bias against retrieving documents, leading to a significant imbalance in the classification. This likely occurred due to the smaller model lacking sufficient internal knowledge, as it is less capable than state-of-the-art models.

While the system was adept at identifying general knowledge questions (those not requiring retrieval), it correctly avoided retrieval $98.17\%$ of the time, refer to Figure 31. However, it almost completely failed at the inverse task, only correctly choosing retrieval $16.59\%$ for the queries that required it. As a result, the overall routing accuracy was limited to just $52.3\%$.

This bias is further evident in the routing decision matrix, which shows that out of 1688 questions that ideally required retrieval to answer correctly, the model incorrectly routed 1408 of them to be answer without retrieving. This fundamental failure to identify questions needing external knowledge confirmed that a simple, unguided prompt is insufficient for creating a reliable query-routing system. While this appears to be true for the chosen model size, further testing is necessary to determine whether this limitation persists in larger models or if they perform better on this task.

### 6.8.2 Instruction V2: An Aggressive, Safety-First Heuristic

**Analysis V2**

```
Query: {query}

You are an expert query analyzer. Your primary goal is to eliminate
incorrect answers by forcing document retrieval for any non-trivial
query. Your default assumption MUST be that retrieval is necessary ('1-
Yes').

You are only permitted to output '2-No' if, and only if, the query
meets ALL of the following strict criteria:
1. The query involves ONLY globally famous entities (e.g., 'France',
'Shakespeare', 'Google').
2. The query asks for a SINGLE, static, and universally known fact
(e.g., a capital city, a famous author's most known work).

For ALL OTHER QUERIES, you MUST output '1-Yes'. This is not optional.
Specifically, if the query has any of the following attributes, you
MUST output '1-Yes':
- Contains specific names of people, places, or things that are not
globally famous (e.g., 'Mabel Murphy Smythe-Haith', 'Axle Whitehead').
- Asks for a specific number, date, or statistic that is not common
knowledge (e.g., the founding year of a specific school).
- Requires a comparison or judgment about scope, profession, or other
nuanced topics (e.g., comparing two authors).

Analyze the query against these rules and provide your decision.

**IMPORTANT** Your response MUST begin with '1-Yes' or '2-No', which
will determine the method used. This is followed by the final answer
within \boxed{{}}.**IMPORTANT**
```

Figure 32: Instruction V2

To counteract the significant bias against retrieval observed on the first approach Figure 30, the second iteration introduced a strong, explicit bias towards retrieval. Instruction V2 Figure 32, framed the model as an ”expert query analyzer” with the primary goal of eliminating incorrect answer by trying to force document retrieval for any non-trivial query. It established retrieval (’1-Yes’) as the default assumption, permitting a direct answer (’2-No’) only if the query met a very strict and narrow set of criteria: it had to involve exclusively ’globally famous entities’ and ask for a single, static, and universally known fact. This ”safety-first” heuristic was designed to try and minimize the risk of factual errors originating from the model’s internal knowledge.

This agressive change dramatically inverted the model’s behavior. The Retrieval Task Routing Accuracy skyrocketed from $16.59\%$ to $98.22\%$, demonstrating that the model could now reliably identify questions that required external documents. Out of 1688 such questions, it correctly chose ”retrieval” for 1659 of them.

However, this success came at significant cost to efficiency and accuracy on the opposite task, with results that were almost predictably inverse to those of the first instruction. The model’s ability to recognize simple, general knowledge questions plummeted. This can be seen on the General Knowledge Routing Accuracy that fell from $98.17\%$ to a mere $26.07\%$. The system was now incorrectly choosing to retrieve documents for the vast majority of the queries that did not need it. This over-correction is properly showed on the decision matrix, where $970$ out of the $1312$ No Retrieval queries, were wrongly sent down the retrieval path.

Although the Overall Routing Accuracy improved to $66.7\%$, this aggressive heuristic proved to be an over-correction. While it successfully enforced the retrieval of necessary information, it failed to account for cases where retrieval was unnecessary. This led to inefficient and often redundant processing for a large number of relatively simple queries that the base model could have answered directly. This showed perfectly that while a strong default can steer the model’s behavior, a purely aggressive approach is too rigid and fails to balance accuracy with efficiency. All of this is evidenced by the results shown in Figure 33.

```
================================================================================
📊 METHODOLOGY EVALUATION RESULTS (FINAL - CORRECTED) 📊
================================================================================


### 1. System Routing Decision Matrix (with Accuracy %) ###
Ideal Path     cot                 nothing             retrieval
--------------------------------------------------------------------------------
Retrieval      ❌ 29 (1.7%)         0 (0.0%)            ✅ 1659 (98.3%)
No-Retrieval   ✅ 22 (1.7%)         ✅ 320 (24.4%)       ❌ 970 (73.9%)
--------------------------------------------------------------------------------

### Routing Accuracy Summary ###
  - Retrieval: 1659/1688 correct (98.3%)
  - No-Retrieval: 342/1312 correct (26.1%)
  - Overall Routing Accuracy: 2001/3000 (66.7%)

### 2. Detailed Routing Performance ###

--- Retrieval Task Routing Accuracy ---
Total questions needing retrieval: 1688
  ✅ Correctly chose 'retrieval': 1659
  🎯 Accuracy on this task: 98.28%

--- General Knowledge Routing Accuracy ---
Total questions NOT needing retrieval: 1312
  ✅ Correctly avoided 'retrieval': 342
  ❌ Incorrectly chose 'retrieval' (Process Errors): 970
  🎯 Accuracy on this task: 26.07%
--------------------------------------------------------------------------------

### 3. Performance & Efficiency by Chosen Path ###
  - Path Taken: 'cot' (51 queries)
    - Answer Accuracy: 68.63%
    - Average Energy: 11180.7510 Joules/query
  - Path Taken: 'nothing' (320 queries)
    - Answer Accuracy: 88.75%
    - Average Energy: 2237.8315 Joules/query
  - Path Taken: 'retrieval' (2629 queries)
    - Answer Accuracy: 86.31%
    - Average Energy: 5401.0439 Joules/query

### 4. Routing System vs. Baseline Model (General Knowledge Only) ###
Analysis of 1312 general knowledge questions:
  - Your Routing System Accuracy: 94.89% (1245/1312)
  - Baseline Straight Model:      74.92% (983/1312)
  📈 Your routing system is 19.97 percentage points BETTER

--- Method Breakdown (What Your System Chose) ---
  - 'retrieval': 96.80% accuracy on 970 questions
  - 'nothing': 88.75% accuracy on 320 questions
    └─ 'Nothing' method achieved 284/320 correct answers
  - 'cot': 100.00% accuracy on 22 questions
================================================================================
```

Figure 33: Instruction V2 Results

### 6.8.3 Instruction V3: Introducing Balanced Criteria

**Analysis V3**

```
Query: {query}

Your goal is to balance accuracy with efficiency. You must decide if a
query requires document retrieval ('1-Yes') or not ('2-No'). Follow
this two-step process:

**Step 1: First, check if the query is a simple case that does NOT need
retrieval.**
You can select '2-No' if the query clearly fits one of these
categories:
  - **Universally Known Facts:** Asks for a single, static fact about a
globally famous subject (e.g., 'What is the capital of France?', 'Who
is the CEO of Google?').
  - **Creative Tasks:** Asks for creative writing, code generation, or
brainstorming (e.g., 'Write a poem about the moon', 'Give me ideas for
a party theme').
  - **General Explanations:** Asks for a general explanation of a
broad, well-known concept (e.g., 'Explain how photosynthesis works',
'What is a neural network?').

**Step 2: If the query is NOT a simple case from Step 1, check for
complexity.**
You MUST select '1-Yes' if the query has any of the following
attributes:
  - **Specific, Non-Famous Entities:** Contains names of people,
places, or titles that are not globally famous (e.g., 'Mabel Murphy
Smythe-Haith', 'Axle Whitehead').
  - **Precise, Non-Trivial Data:** Asks for specific numbers, dates, or
statistics that are not common knowledge (e.g., 'founding year of a
specific school', 'box office numbers for a movie').
  - **Comparisons or Analysis:** Requires a nuanced comparison or
analysis that would be strengthened by evidence (e.g., 'Who has more
scope of profession...', 'What are the main criticisms of...').

**Decision Heuristic:** If the query does not clearly fit into a
'simple case' from Step 1, your default action should be to retrieve
('1-Yes') to ensure accuracy.

**IMPORTANT** Your response MUST begin with '1-Yes' or '2-No', which
will determine the method used. This is followed by the final answer
within \boxed{{}}.**IMPORTANT**
```

Figure 34: Instruction V3

The third iteration, Instruction V3, tried to strike a balance between the aggressive retrieval strategy of V2 and the passive approach of V1. The goal was to try and improve efficiency by reducing unnecessary retrievals without sacrificing the accuracy gains made on complex queries. The key refinement was the introduction of explicit, positive categories for non-retrieval (”2-No”). For the first time , the model was given clear examples of queries that were meant to be answer directly, such as ”Universally Known Facts”, ”Creative Tasks”, and ”General Explanations”.

This structured, two-step process first checks for a simple case, and only then defaulting to retrieval on more complex queries. This proved to be a significant step in the right direction. The model was no longer forced into an overly aggressive default and was instead required to

reason through its decision-making process to select the appropriate path.

```
================================================================================
📊 METHODOLOGY EVALUATION RESULTS (FINAL - CORRECTED) 📊
================================================================================


### 1. System Routing Decision Matrix (with Accuracy %) ###
Ideal Path     cot                nothing            retrieval
--------------------------------------------------------------------------------
Retrieval      ❌ 32 (1.9%)       ❌ 1 (0.1%)        ✅ 1655 (98.0%)
No-Retrieval   ✅ 99 (7.5%)       ✅ 180 (13.7%)     ❌ 1033 (78.7%)
--------------------------------------------------------------------------------

### Routing Accuracy Summary ###
  - Retrieval: 1655/1688 correct (98.0%)
  - No-Retrieval: 279/1312 correct (21.3%)
  - Overall Routing Accuracy: 1934/3000 (64.5%)

### 2. Detailed Routing Performance ###

--- Retrieval Task Routing Accuracy ---
Total questions needing retrieval: 1688
  ✅ Correctly chose 'retrieval': 1655
  🎯 Accuracy on this task: 98.05%

--- General Knowledge Routing Accuracy ---
Total questions NOT needing retrieval: 1312
  ✅ Correctly avoided 'retrieval': 279
  ❌ Incorrectly chose 'retrieval' (Process Errors): 1033
  🎯 Accuracy on this task: 21.27%
--------------------------------------------------------------------------------

### 3. Performance & Efficiency by Chosen Path ###
  - Path Taken: 'cot' (131 queries)
    - Answer Accuracy: 87.02%
    - Average Energy: 7101.4066 Joules/query
  - Path Taken: 'nothing' (181 queries)
    - Answer Accuracy: 87.85%
    - Average Energy: 1979.5231 Joules/query
  - Path Taken: 'retrieval' (2688 queries)
    - Answer Accuracy: 86.24%
    - Average Energy: 5714.2189 Joules/query

### 4. Routing System vs. Baseline Model (General Knowledge Only) ###
Analysis of 1312 general knowledge questions:
  - Your Routing System Accuracy: 95.81% (1257/1312)
  - Baseline Straight Model:      74.92% (983/1312)
  📈 Your routing system is 20.88 percentage points BETTER

--- Method Breakdown (What Your System Chose) ---
  - 'retrieval': 96.90% accuracy on 1033 questions
  - 'nothing': 88.33% accuracy on 180 questions
    └─ 'Nothing' method achieved 159/180 correct answers
  - 'cot': 97.98% accuracy on 99 questions
================================================================================
```

Figure 35: Instruction V3 Results

The results depicted in Figure 35 reflect this new found balance. The Retrieval Task Routing Accuracy remained exceptionally high at around $98\%$, indicating that the safety-first principle for complex questions was successfully maintained. The model correctly identified 1655 out of 1688 queries that required retrieval.

Crucially, the opposite task had been the main weakness of the previous iteration, and this remained true in the current version, with results deteriorating further the General Knowledge Routing Accuracy dropped from $26.07\%$ to $21.27\%$ on the General Knowledge Routing Accuracy. This can also be seen on the number of times that the model picked ”retrieval” 1033 of the 1312 general knowledge questions that don't require it.

### 6.8.4 Instruction V4: A Shift to Profile-Based Classification

**Analysis V4**

```
Query: {query}

You are a query classification expert. Your task is to classify the
user's query into one of two profiles to determine if document
retrieval is necessary for an accurate answer. Choose the profile that
is the BEST fit.

--- PROFILE 1: RETRIEVAL REQUIRED ('1-Yes') ---
Choose this profile if the query requires external, specific, or up-to-
date knowledge. It MUST be chosen if the query involves:
  - **Specific & Non-Famous Entities:** People, organizations, or
titles that are not globally famous (e.g., 'Mabel Murphy Smythe-Haith',
'Axle Whitehead').
  - **Precise Data:** Specific numbers, dates, statistics, or financial
data that is not common knowledge (e.g., 'founding year of Alabama
State University', 'box office numbers for a specific film').
  - **Comparative Judgments or Opinions:** Questions asking for
analysis, comparison, scope, or criticism that require evidence (e.g.,
'Who has more scope of profession...', 'What are the main criticisms of
X?').
  - **Recent Events:** Any topic or event that has occurred very
recently, as your internal knowledge may be outdated.

--- PROFILE 2: DIRECT ANSWER SUFFICIENT ('2-No') ---
Choose this profile ONLY if the query is self-contained and relies on
stable, common knowledge or creative generation. It is SAFE to choose
this profile if the query is clearly one of the following:
  - **Common Knowledge:** Asks for a single, static, undisputed fact
about a globally famous entity (e.g., 'What is the capital of France?',
'Who wrote Hamlet?', 'What is the chemical formula for water?').
  - **General Concepts:** Asks for a broad definition or explanation of
a well-established concept (e.g., 'Explain photosynthesis', 'What is a
neural network?').
  - **Creative & Logic Tasks:** Asks for creative writing (poems,
stories), brainstorming, code generation, math problems, or logic
puzzles.

**Decision Rule:** Compare the query against both profiles. If it
clearly fits Profile 2, select '2-No'. In all other cases, especially
if there is any doubt or overlap, you MUST default to the safety of '1-
Yes'.

**IMPORTANT** Your response MUST begin with '1-Yes' or '2-No', which
will determine the method used. This is followed by the final answer
within \boxed{{}}.**IMPORTANT**
```

Figure 36: Instruction V4

The mixed results presented on the previous iteration highlighted a potential weakness in the sequential, rules-based checklist approach. This new instruction V4 represented a major conceptual shift, this time re-framing the task following a procedure to a more holistic classification exercise. Instead of the previous step-by-step process, the model was now tasked with matching the user's query to one of two detailed profiles: "Profile 1: Retrieval Required" or "Profile 2: Direct Answer Sufficient".

This new profile-based structure is more intuitive for the LLM, as it leverages a core strength of these types of models pattern-matching. The profiles provided a clearer, more organized

framework, and critically introduced the ”Recent Events” as a trigger for retrieval, this was the approach chosen to try and remedy the LLM knowledge cut-off from more recent knowledge. The decision rule, however, still maintained a cautious stance, instructing the model to default to the safety of ”1-Yes” in cases of doubt or ambiguity.

```
================================================================================
📊 METHODOLOGY EVALUATION RESULTS (FINAL - CORRECTED) 📊
================================================================================


### 1. System Routing Decision Matrix (with Accuracy %) ###
Ideal Path     cot                 nothing             retrieval
--------------------------------------------------------------------------------
Retrieval      ❌ 137 (8.1%)        0 (0.0%)            ✅ 1551 (91.9%)
No-Retrieval   ✅ 419 (31.9%)       ✅ 491 (37.4%)       ❌ 402 (30.6%)
--------------------------------------------------------------------------------

### Routing Accuracy Summary ###
  - Retrieval: 1551/1688 correct (91.9%)
  - No-Retrieval: 910/1312 correct (69.4%)
  - Overall Routing Accuracy: 2461/3000 (82.0%)

### 2. Detailed Routing Performance ###

--- Retrieval Task Routing Accuracy ---
Total questions needing retrieval: 1688
  ✅ Correctly chose 'retrieval': 1551
  🎯 Accuracy on this task: 91.88%

--- General Knowledge Routing Accuracy ---
Total questions NOT needing retrieval: 1312
  ✅ Correctly avoided 'retrieval': 910
  ❌ Incorrectly chose 'retrieval' (Process Errors): 402
  🎯 Accuracy on this task: 69.36%
--------------------------------------------------------------------------------

### 3. Performance & Efficiency by Chosen Path ###
  - Path Taken: 'cot' (556 queries)
    - Answer Accuracy: 85.97%
    - Average Energy: 7965.2769 Joules/query
  - Path Taken: 'nothing' (491 queries)
    - Answer Accuracy: 93.48%
    - Average Energy: 3130.6094 Joules/query
  - Path Taken: 'retrieval' (1953 queries)
    - Answer Accuracy: 82.59%
    - Average Energy: 7345.1696 Joules/query

### 4. Routing System vs. Baseline Model (General Knowledge Only) ###
Analysis of 1312 general knowledge questions:
  - Your Routing System Accuracy: 95.27% (1250/1312)
  - Baseline Straight Model:      74.92% (983/1312)
  📈 Your routing system is 20.35 percentage points BETTER

--- Method Breakdown (What Your System Chose) ---
  - 'cot': 95.47% accuracy on 419 questions
  - 'nothing': 93.48% accuracy on 491 questions
    └─ 'Nothing' method achieved 459/491 correct answers
  - 'retrieval': 97.26% accuracy on 402 questions
================================================================================
```

Figure 37: Instruction V4 Results

This new approach proved to be a big breakthrough. The results show a dramatically more balanced and effective system as showed in Figure 37.

The General Knowledge Routing Accuracy saw a massive crucial improvement jumping from the previous $21.27\%$ to $69.36\%$. For the first time, the model could correctly identify the majority of the questions that did not require retrieval.

This improvement came with a small trade-off. The Retrieval Task Routing Accuracy saw a slight dip from the near perfect levels of V2/V3, but still remaining very good at $91.88\%$.

As a result of this new balance, the Overall Routing Accuracy increased to $82.0\%$ the highest and most effective level achieved so far indicating that the instruction was finally moving in the right direction.

The success of this version is best captured in the "Routing System Vs. Baseline Model" analysis for general knowledge questions. This version of the routing system achieved a $95.27\%$ answer accuracy, which was 20.35 percentage points better than the baseline for just model answering on its own without any guiding instruction. By successfully re-framing the task to align with the LLM's natural capabilities, this instruction created a far more reliable and smart classification system.

### 6.8.5 Instruction V5: Final Refinement with a Guiding Principle

**Analysis V5**

```
Query: {query}

You are a query classification expert. Your task is to classify the
user's query into one of two profiles to determine if document
retrieval is necessary for an accurate answer. Choose the profile that
is the BEST fit.

--- PROFILE 1: RETRIEVAL REQUIRED ('1-Yes') ---
Choose this profile if the query requires external, specific, or up-to-
date knowledge. It MUST be chosen if the query involves:
  - **Specific & Non-Famous Entities:** People, organizations, or
titles that are not globally famous (e.g., 'Mabel Murphy Smythe-Haith',
'Axle Whitehead').
  - **Precise Data:** Specific numbers, dates, statistics, or financial
data that is not common knowledge (e.g., 'founding year of Alabama
State University', 'box office numbers for a specific film').
  - **Comparative Judgments or Opinions:** Questions asking for
analysis, comparison, scope, or criticism that require evidence (e.g.,
'Who has more scope of profession...', 'What are the main criticisms of
X?').
  - **Recent Events:** Any topic or event that has occurred very
recently, as your internal knowledge may be outdated.

--- PROFILE 2: DIRECT ANSWER SUFFICIENT ('2-No') ---
Choose this profile ONLY if the query is self-contained and relies on
stable, common knowledge or creative generation. It is SAFE to choose
this profile if the query is clearly one of the following:
  - **Common Knowledge:** Asks for a single, static, undisputed fact
about a globally famous entity (e.g., 'What is the capital of France?',
'Who wrote Hamlet?', 'What is the chemical formula for water?').
  - **General Concepts:** Asks for a broad definition or explanation of
a well-established concept (e.g., 'Explain photosynthesis', 'What is a
neural network?').
  - **Creative & Logic Tasks:** Asks for creative writing (poems,
stories), brainstorming, code generation, math problems, or logic
puzzles.

**Decision Rule:** Your primary goal is accuracy. First, check if the
query is a perfect, unambiguous match for Profile 2. If there is *any*
ambiguity, or if any part of the query touches on a Profile 1
characteristic (like a specific name or date), you MUST choose the
safety of '1-Yes'. A slow but correct answer is always better than a
fast but wrong one.

**IMPORTANT** Your response MUST begin with '1-Yes' or '2-No', which
will determine the method used. This is followed by the final answer
within \boxed{{}}.**IMPORTANT**
```

Figure 38: Instruction V5

Building on the successful profile-based structure of V4, the fifth version was a final refinement aimed at maximizing reliability. The core structure of the two profiles remained unchanged, but a critical addition was made to the Decision Rule. This new rule introduced an explicit guiding principle to resolve the possible ambiguity: ”A slow but correct answer is always better than a fast but wrong one.”.

This principle served as a powerful tie-breaker, forcing the idea of accuracy on to the model. It explicitly stated that if there was any ambiguity, or if a query even touched on the character-

istics from ”Profile 1” (like a specific name or date), it must default to the safety of retrieval. This approach was designed to try and solidify the instruction’s focus on producing the most trustworthy assessment possible, even at the cost of possible decrease in efficiency.

```
================================================================================
📊 METHODOLOGY EVALUATION RESULTS (FINAL - CORRECTED) 📊
================================================================================


### 1. System Routing Decision Matrix (with Accuracy %) ###
Ideal Path     cot                nothing            retrieval
--------------------------------------------------------------------------------
Retrieval      ❌ 135 (8.0%)       0 (0.0%)           ✅ 1553 (92.0%)
No-Retrieval   ✅ 435 (33.2%)      ✅ 445 (33.9%)      ❌ 432 (32.9%)
--------------------------------------------------------------------------------

### Routing Accuracy Summary ###
  - Retrieval: 1553/1688 correct (92.0%)
  - No-Retrieval: 880/1312 correct (67.1%)
  - Overall Routing Accuracy: 2433/3000 (81.1%)

### 2. Detailed Routing Performance ###

--- Retrieval Task Routing Accuracy ---
Total questions needing retrieval: 1688
  ✅ Correctly chose 'retrieval': 1553
  🎯 Accuracy on this task: 92.00%

--- General Knowledge Routing Accuracy ---
Total questions NOT needing retrieval: 1312
  ✅ Correctly avoided 'retrieval': 880
  ❌ Incorrectly chose 'retrieval' (Process Errors): 432
  🎯 Accuracy on this task: 67.07%
--------------------------------------------------------------------------------

### 3. Performance & Efficiency by Chosen Path ###
  - Path Taken: 'cot' (570 queries)
    - Answer Accuracy: 86.14%
    - Average Energy: 8003.5873 Joules/query
  - Path Taken: 'nothing' (445 queries)
    - Answer Accuracy: 93.26%
    - Average Energy: 3146.6699 Joules/query
  - Path Taken: 'retrieval' (1985 queries)
    - Answer Accuracy: 82.82%
    - Average Energy: 7299.0219 Joules/query

### 4. Routing System vs. Baseline Model (General Knowledge Only) ###
Analysis of 1312 general knowledge questions:
  - Your Routing System Accuracy: 95.27% (1250/1312)
  - Baseline Straight Model:      74.92% (983/1312)
  📈 Your routing system is 20.35 percentage points BETTER

--- Method Breakdown (What Your System Chose) ---
  - 'cot': 94.94% accuracy on 435 questions
  - 'retrieval': 97.69% accuracy on 432 questions
  - 'nothing': 93.26% accuracy on 445 questions
    └─ 'Nothing' method achieved 415/445 correct answers
================================================================================
```

Figure 39: Instruction V5 Results

The performance data shows that this refinement had a subtle but measurable impact, tuning the model's behavior as it was intended.

The model became slightly more cautious. The Retrieval Task Routing Accuracy slightly increased from $91.88\%$ to $92.0\%$, which means the model identified 1553 of the 1688 queries that required retrieval. This increased caution also resulted in a minor decrease in the General Knowledge Routing Accuracy, which shifted from $69.36\%$ to $67.07\%$.

The model was now slightly more likely to send a simple query for retrieval if it contained any element of ambiguity. This adjustment was made based on the reasoning that the model might still produce a correct answer even when retrieval is not strictly necessary. However, the inverse failing to retrieve when it is required almost always results in incorrect answers. As a consequence of this shift, a slight dip in accuracy was observed, with the Overall Routing Accuracy decreasing to $81.1\%$.

Despite the minor shifts in routing metrics, the final answer quality for general knowledge questions remained identical to that of V4, with the routing system achieving a $95.27\%$ . Similarly to V4, this instruction resulted in a 20.35 percentage point improvement over the baseline. This shows that V5 successfully reinforced the system's reliability.

### 6.8.6 Instruction V6: A Strategic Pivot to Efficiency

**Analysis V6**

```
Query: {query}

You are a query classification expert. Your goal is to accurately determine if
document retrieval is necessary, with a specific focus on AVOIDING unnecessary
retrieval for general knowledge questions.

--- PROFILE 1: RETRIEVAL REQUIRED ('1-Yes') ---

Choose this profile ONLY if the query contains a clear 'retrieval trigger'. A
trigger is present if the query involves:

  - **Low-Fame Entities:** Specific people, organizations, or products that are
not globally famous (e.g., 'Mabel Murphy Smythe-Haith', 'Axle Whitehead'). If
you don't have deep, confident knowledge of the entity, retrieve.

  - **Hard Data:** Requests for precise numbers, dates, or stats that are not
common trivia (e.g., 'box office numbers for a specific film', 'founding year
of Alabama State University', 'population of a small city').

  - **Recent Information:** Any event, product release, or news from the last
two years.

  - **Document-Based Analysis:** Questions asking for comparisons, scope, or
criticisms that imply needing information from a specific text.

--- PROFILE 2: DIRECT ANSWER SUFFICIENT ('2-No') ---

This is the default profile for most queries. Choose this if the query relies
on stable, common, or encyclopedic knowledge. It is the correct choice for:

  - **Common Knowledge:** Questions about globally famous entities, major
historical events, and well-established scientific facts (e.g., 'What is the
capital of France?', 'Who wrote Hamlet?', 'What is H2O?'). This includes famous
dates or numbers (e.g., 'When did WWII end?', 'What is the value of Pi?').

  - **General Concepts:** Broad definitions or explanations (e.g., 'Explain
photosynthesis', 'What is a neural network?').

  - **Creative & Logic Tasks:** Requests for code, poems, math problems, or
brainstorming.

**Decision Rule:** Your primary goal is to reduce incorrect '1-Yes'
classifications. Start with the assumption that the query is '2-No'. Scrutinize
the query for a clear and definite 'retrieval trigger' from Profile 1. If no
such trigger is present, you MUST select '2-No'. Do not default to '1-Yes' out
of mere caution; an explicit reason is required.

**IMPORTANT** Your response MUST begin with '1-Yes' or '2-No'.**IMPORTANT**
```

Figure 40: Instruction V6

The final iteration, V6, represented a deliberate reversal from the ”accuracy-first” principle that guided the previous version. The main goal was explicitly re-focused to ”AVOIDING unnecessary document retrieval” and to ”reduce incorrect ’1-Yes’ classifications”. This instruction was designed to test a high-efficiency approach, that prioritizes speed and resource conservation for general knowledge questions.

To achieve this, the core logic was inverted. Non-retrieval (”2-No”) was made to be the default path, and the model was instructed to choose this approach unless a ”clear and definite ’retrieve trigger’” was present on the query. The burden of proof was shifted: instead of defaulting to the safer retrieval in cases of ambiguity, the model now required compelling explicit reason to engage the retrieval system.

```
================================================================================
📊 METHODOLOGY EVALUATION RESULTS (FINAL - CORRECTED) 📊
================================================================================


### 1. System Routing Decision Matrix (with Accuracy %) ###
Ideal Path     cot                 nothing             retrieval
--------------------------------------------------------------------------------
Retrieval      ❌ 696 (41.2%)       0 (0.0%)            ✅ 992 (58.8%)
No-Retrieval   ✅ 1186 (90.4%)      ✅ 1 (0.1%)          ❌ 125 (9.5%)
--------------------------------------------------------------------------------

### Routing Accuracy Summary ###
  - Retrieval: 992/1688 correct (58.8%)
  - No-Retrieval: 1187/1312 correct (90.5%)
  - Overall Routing Accuracy: 2179/3000 (72.6%)

### 2. Detailed Routing Performance ###

--- Retrieval Task Routing Accuracy ---
Total questions needing retrieval: 1688
  ✅ Correctly chose 'retrieval': 992
  🎯 Accuracy on this task: 58.77%

--- General Knowledge Routing Accuracy ---
Total questions NOT needing retrieval: 1312
  ✅ Correctly avoided 'retrieval': 1187
  ❌ Incorrectly chose 'retrieval' (Process Errors): 125
  🎯 Accuracy on this task: 90.47%
--------------------------------------------------------------------------------

### 3. Performance & Efficiency by Chosen Path ###
  - Path Taken: 'cot' (1882 queries)
    - Answer Accuracy: 78.43%
    - Average Energy: 8035.7696 Joules/query
  - Path Taken: 'nothing' (1 queries)
    - Answer Accuracy: 100.00%
    - Average Energy: 2431.0774 Joules/query
  - Path Taken: 'retrieval' (1117 queries)
    - Answer Accuracy: 78.96%
    - Average Energy: 8011.2825 Joules/query

### 4. Routing System vs. Baseline Model (General Knowledge Only) ###
Analysis of 1312 general knowledge questions:
  - Your Routing System Accuracy: 97.18% (1275/1312)
  - Baseline Straight Model:      74.92% (983/1312)
  📈 Your routing system is 22.26 percentage points BETTER

--- Method Breakdown (What Your System Chose) ---
  - 'cot': 96.96% accuracy on 1186 questions
  - 'retrieval': 99.20% accuracy on 125 questions
  - 'nothing': 100.00% accuracy on 1 questions
    └─ 'Nothing' method achieved 1/1 correct answers
================================================================================
```

Figure 41: Instruction V6 Results

This strategic change had a profound and predictable impact on the system's performance effectively trading accuracy for efficiency.

The instruction's goal was a success for the General Knowledge Routing Accuracy making it surge to its highest point across all the previous versions, reaching and impressive $90.47\%$ as depicted in Figure 41. The system was now exceptionally skilled at identifying and directly answering simple queries without using wasteful processing when not required.

This efficiency came at a significant and expected cost. The Retrieval Task Routing Accuracy fell sharply from $92.00\%$ to a measly $58.77\%$. By no longer erring on the side of caution, the system failed to identify a large portion of queries that genuinely required the retrieval of external information.

As a result of this trade-off , the Overall Routing Accuracy dropped to $72.6\%$.

Interestingly, despite the lower routing accuracy for complex queries, this approach achieved the highest final answer accuracy on general knowledge questions with $97.18\%$. This was a $22.26$ percentage points over the baseline answers. This outcome shows that by correctly routing a very high volume of simple questions to the direct-answer paths, the system maximized the LLm's ability to leverage its own knowledge effectively, this was also helped with the Chain-of-Thought instruction that improved the base model answer without requiring any external information.

This last experiment shows the high degree of control that prompt engineering provides on such a system and even on the LLM's responses. V6 is not inherently better or worse than V5, it simply optimized and constructed with a different objective in mind. The choice between these two mature instructions depends entirely on the desired system behavior. Which aligns with the purpose of this research prioritizing efficiency whenever the trade-off proves to be worthwhile. V5 is the ideal choice for a system where reliability and avoiding factual errors are paramount, while the V6 version is superior for a system where efficiency and speed in handling common queries are the up most concern.

## 6.9 Analysis of Energy Consumption and Efficiency

A holistic view of the system's performance is best captured by plotting the total energy consumed versus the number of correct answers that the system outputted. The resulting scatter plot reveals a fundamental trade-off inherent in the system's operation: achieving a higher number of correct answers of the provided dataset is directly correlated with increased in energy consumption (see Figure42).

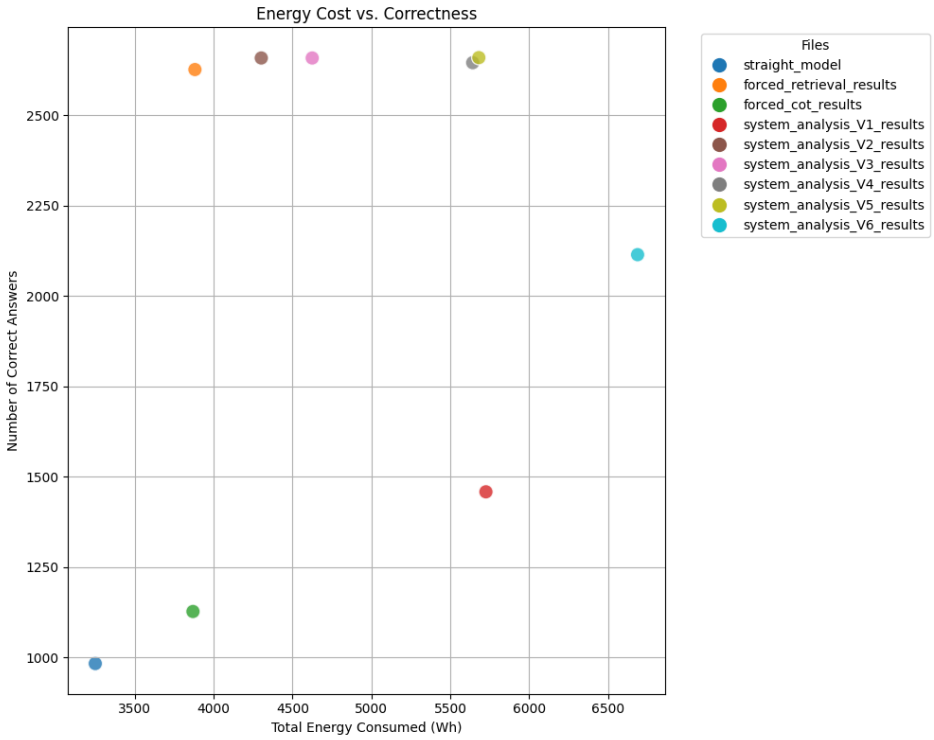



Figure 42: Energy Costs vs. Correctness Scatterplot

This is clearly lustrated in the progression from the baseline models, which occupy the lower-left quadrant of the graph which represents both the lowest energy consumption and the lowest correctness. Although one part of the baseline achieved high correctness, this may be partly due to the way correctness was assigned to queries requiring retrieval. In this evaluation, if retrieval was triggered for a query that required it, the system marked the answer as correct regardless of whether the retrieved content actually led to a correct response. This introduces a significant limitation when interpreting the results. If correctness had instead been evaluated based on the accuracy of the retrieved content itself, the score would likely have been much lower and more in line with the other two baseline results. Another important factor is that, since the retrievable documents come from Wikipedia, many of them coincidentally align with ARC-Easy queries. For example, in Query 3, the retrieved document happens to contain information that, through reasoning, leads to the correct answer. This pattern appears in multiple ARC-style queries and may inflate the apparent effectiveness of the baseline. These two points boost the answer correctness by a lot for this specific file thus should be looked at as a skewed result far from the truth.

At the upper-right , the system demonstrates its ability to improve both the model's correctness and the overall quality of the answers. However, one outlier stands out: System Analysis V1. This version reveals a particularly inefficient instruction, likely due to how open it was to interpretation. As a result, the model engaged in excessive reflection while still failing to choose the appropriate approach for each query type. This led to a significantly lower number of correct answers compared to later, more refined instruction versions.

An important takeaway is that the optimization process was not intended to reduce energy consumption, but rather to maximize the productive use of that energy aiming to yield the most accurate results possible.

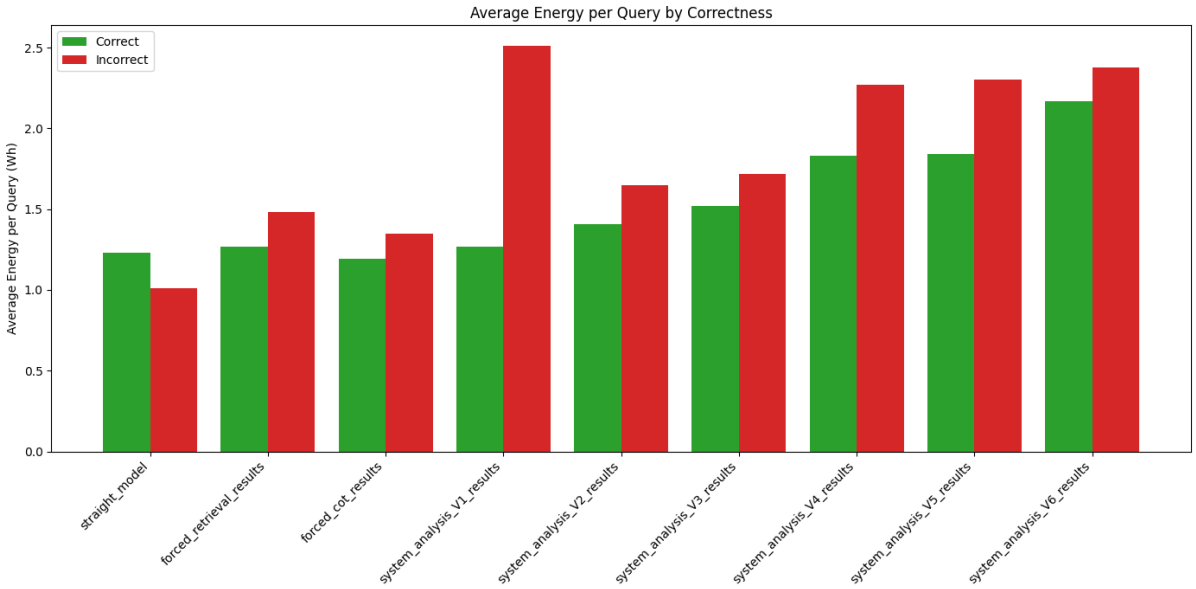



Figure 43: Energy Costs vs. Correctness Scatterplot

Looking further into the energy dynamics of this project, an analysis of the average energy per query reveals a striking and consistent pattern, incorrect answers are consistently more energy-intensive than correct ones. This suggests that incorrect answers are often the result of inefficient processing, such as retrieving irrelevant documents, pursuing flawed reasoning paths, or struggling to analyze conflicting information. In contrast, correct answers appear to follow a more direct and energetically efficient path. With one exception, the straight_model baseline, where the correct answers consume slightly more energy. This could be because, when the model is more confident in its answer, it tends to generate longer, more detailed responses, which in turn require more energy than the simpler, shorter incorrect ones.

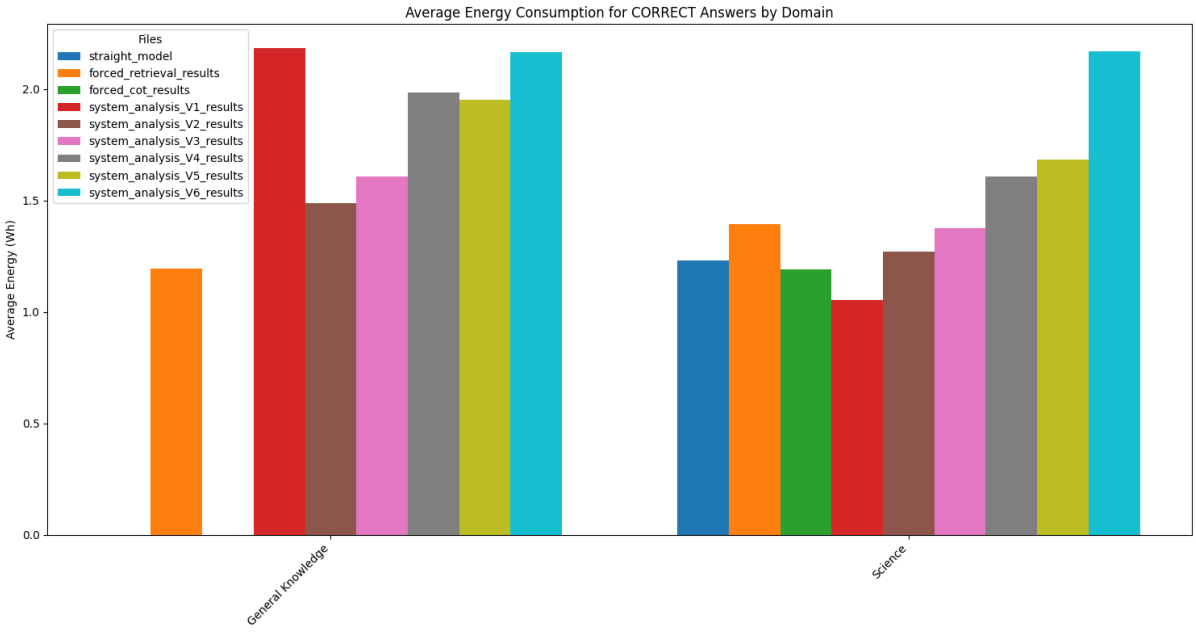



Figure 44: Domain Average Energy per Correct Answer

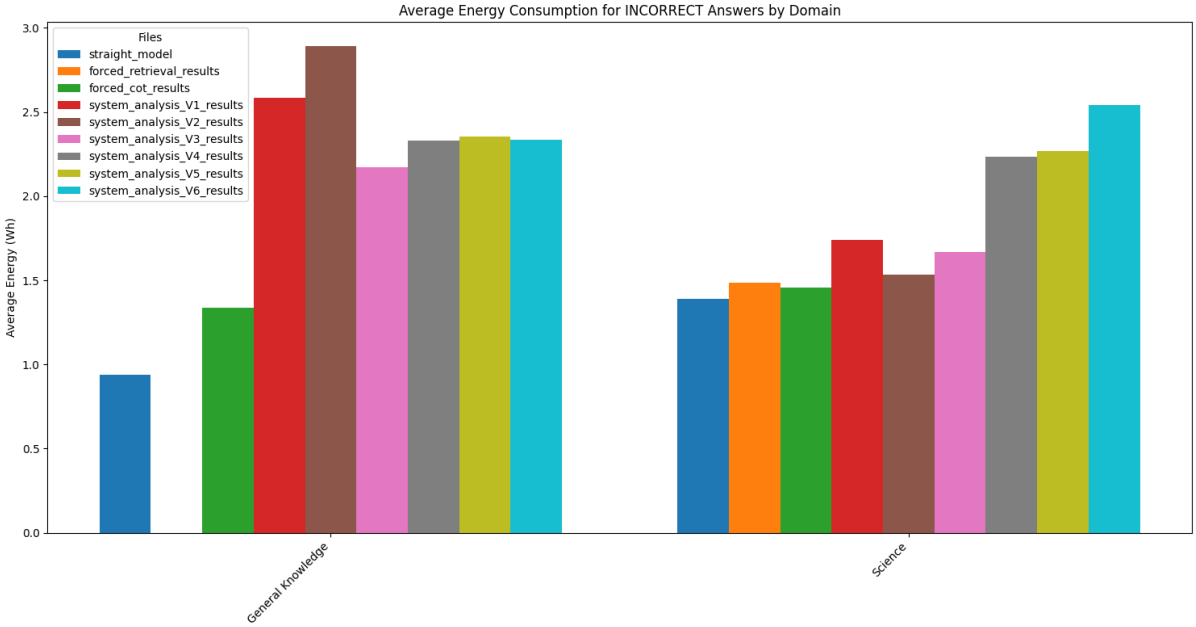



Figure 45: Domain Average Energy per Incorrect Answer

Further analysis of the results, now split by the query's original dataset (domain), reveals another clear efficiency trend: queries related to 'General Knowledge' consistently require more energy than those from the 'Science' domain. This pattern is still present for both correct and incorrect answers, as can be seen in both Figure 44, and 45. For the more refined instructions (V2 through V6), the energy cost for answering a general knowledge question is noticeably higher than for a science question. This disparity likely stems from the increased complexity of the general knowledge queries, which typically require retrieval to be answered. This adds

further computational overhead, as the number of tokens the model must process increases due to the inclusion of retrieved documents alongside the instruction. This occurs even after organizing the retrieved documents from most to least relevant, and removing any unnecessary expressions that do not contribute to the quality of the response. On the other hand, in the Science domain, the number of input tokens is always lower since no retrieval is performed when the system functions correctly, thereby reducing the total input tokens.

In summary, the energy consumption analysis provides a comprehensive, multi-dimensional view of the system's efficiency. It demonstrates that efficiency is not a single metric but a balance of multiple factors. The inherent complexity of the query's domain sets a baseline for energy consumption, with "General Knowledge" questions proving to be more resource-intensive due to the required retrieval process needed to answer them correctly. UGiven this baseline, the effectiveness of the instructional prompt plays a pivotal role in how efficiently the energy is utilized. Well-calibrated instructions help guide the model down more efficient pathways, leading to correct answers at a lower average energy cost. This demonstrates that query editing could be a promising area of study for improving model efficiency. While also proving that ambiguity or flawed logic results in wasted energy on incorrect outputs. Ultimately, this analyses reinforces that the iterative refinement of prompts is not merely a quest for higher accuracy, but a method for controlling the crucial trade-off between correctness and computational cost, while also allowing for the strategic selection of a system profile that best aligns with the desired balance of performance and resource conservation.

## 6.10 Detailed Energy Consumption Profiles

### 6.10.1 Overall Energy Trends Across Instruction Versions

A foundational analyses of the the system's energy consumption begins with the average energy consumed per query for each experimental version of the instructions, as showed in Figure 46. This shows a clear high-level comparison of the computational cost associated with each iteration of the instructions against the baseline models.

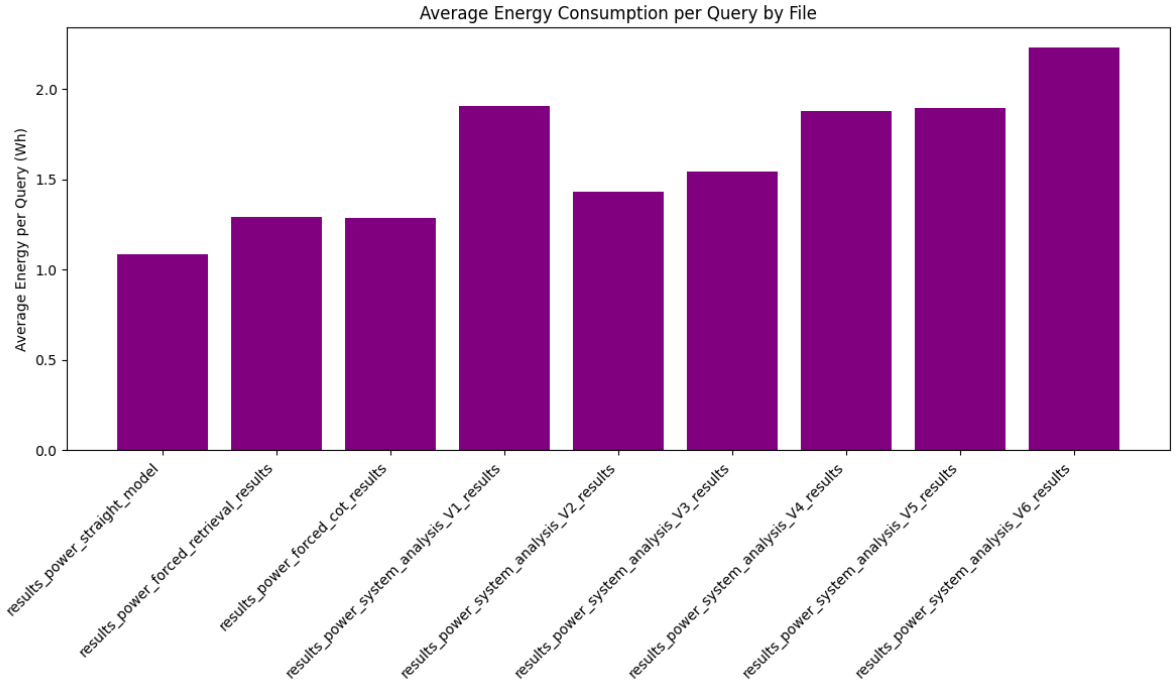



Figure 46: Average Energy Consumption per Query by File

The baseline models establish a lower bound for energy usage, using around $1.1$ and $1.3$ Wh per query. The introduction of the first routing instruction, V1, immediately results in a significant spike in energy consumption to nearly $2.0$ Wh. This aligns with its characterization as a inefficient, open-ended prompt that likely caused extensive and unguided model processing.

As the instructions were refined from V2 to V5, the average energy fluctuated between $1.45$ and $1.9$ Wh. This demonstrates that the added complexity of the routing system, even when optimized for accuracy, carries a consistent energy overhead compared to the simpler baseline approaches as expected since it requires the model to output two responses for just one query. Interestingly, Instruction V6, which was explicitly designed to enhance efficiency by trying to reduce the usage off unnecessary retrievals, results on the highest average energy consumption at around $2.2$ Wh. misconception during the creation of the instruction, as it was initially assumed that the retrieval process would be the most energy-intensive. However, the results show that although retrieval does contribute to energy consumption, it represents only a small portion of the overall cost. Another mistake made during the creation of this instruction was that it ended up heavily relying on an internal chain-of-though process to answer general knowledge queries, which resulted in higher energy consumption. However, it also led to faster response times contradicting some of the assumptions made during the instruction's design.

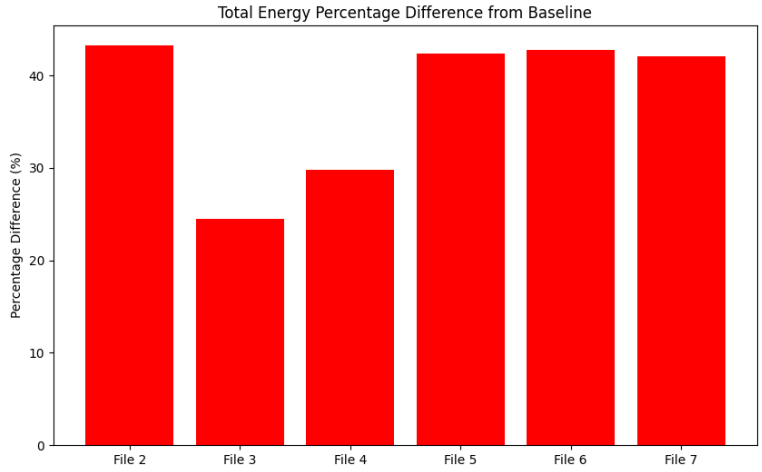



Figure 47: Total Energy Percentage Difference from Baseline

This increased energy overhead is clearly illustrated in Figure 47. The graph shows that, compared to the baseline, the more advanced routing systems consistently increase total energy consumption by over $40\%$. This underscores a critical finding, that this implementation of an intelligent routing layer, while substantially improving answer accuracy and reliability, it also presents a significantly and quantifiable trade-off in terms of computational and energy cost. While this was already considered at the start of this endeavor, the main idea is to try and compete with much larger models that, on average, require significantly more energy and computational power. A deeper analysis of this will be carried out at a later stage.

### 6.10.2 CPU vs. GPU: Deconstructing the Energy Cost

To try and understand the nature of the energy overhead introduced by the routing system, it is essential to deconstruct the total energy consumption into its primary hardware components, the *Central Processing Unit* (CPU) and the *Graphics Processing Unit* (GPU). The CPU normally handles data processing, I/O operations, and logical orchestration, while the GPU is responsible for parallel computations required by the model inference. The Figure 48 illustrates this breakdown for each system version.

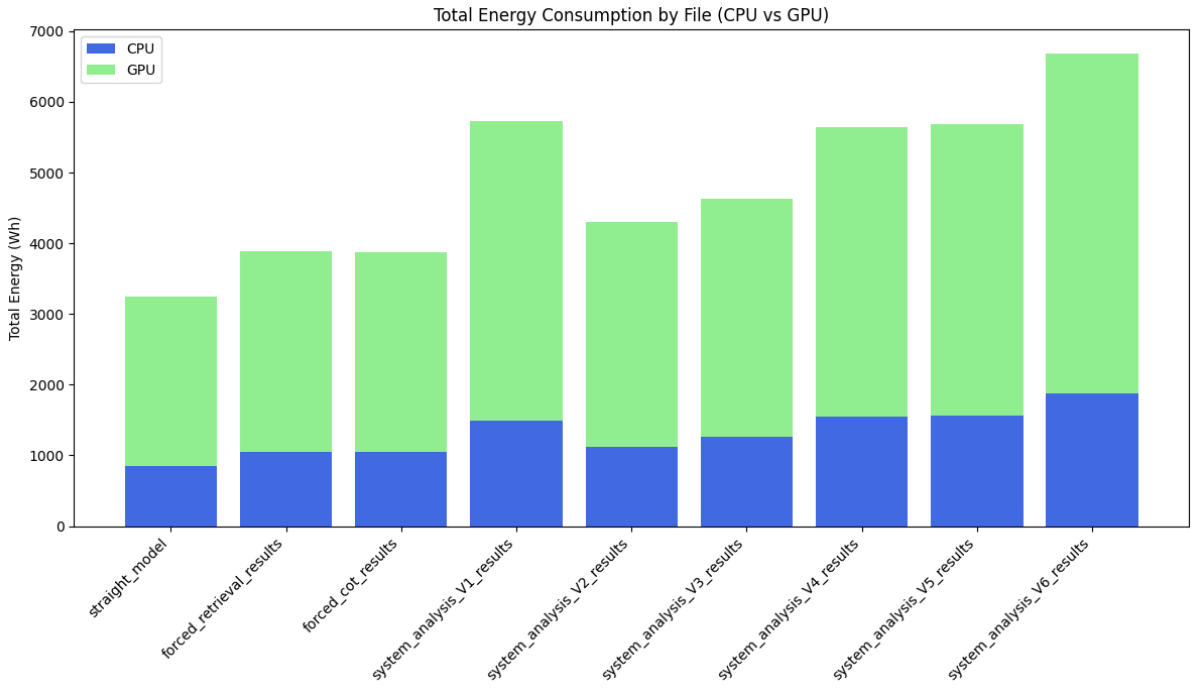



Figure 48: Total Energy Consumption by File (CPU vs GPU)

A clear pattern emerges from the data, the baseline ”straight_model”, which relies almost exclusively on model inference, shows the lowest relative CPU energy consumption. On the other hand, all subsequent versions that incorporate either forced retrieval or CoT or the intelligent routing system exhibit an increase in the proportion of energy consumed by the CPU.

Conversely, the GPU energy consumption scales more directly with the complexity and length of the generation required from the LLM. The V6 instruction, which was designed to favor internal Chain-of-Thought reasoning over retrieval, shows the highest energy consumption driven by a massive increase in GPU usage. This shows that while avoiding CPU-heavy retrieval processes, version V6 shifts most of the burden to the GPU, requiring it to perform more extensive and energy-demanding computations to generate the answers. As previously observed, this also resulted in poorer performance compared to earlier iterations.

This analysis reveals that the choice of instruction foes not just how much energy is consumed, but also where it is used. A retrieval heavy strategy taxes the CPU, while a reasoning heavy strategy taxes the GPU. This distinction is critical for system optimization, as it shows how different prompts and engineering strategies can create distinct hardware usage profiles.

### 6.10.3 The Energetic Cost of Correcting Errors

Looking beyond the general energy profiles, a more targeted analysis reveals the specific energy consumption required for the system to add value that is, to correct an answer that the baseline model would have gotten wrong. This scenario represents the core justification for this whole approach.

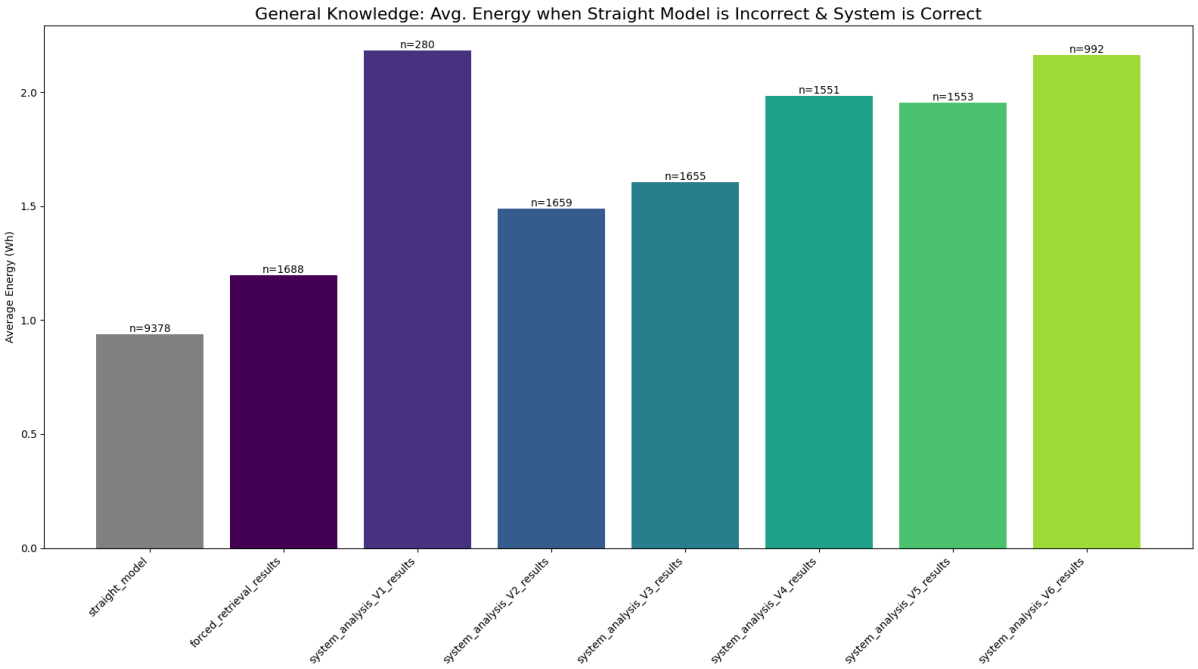



Figure 49: General Knowledge: Avg. Energy when Straight Model is Incorrect & System is Correct

Figure 49 provides a quantitative analysis of the 'correction cost' associated with 'General Knowledge' queries. When the baseline fails while consuming around $0.95$ Wh, the various routing systems successfully provided a correct answer, although this came with a significantly higher energy cost, ranging from $1.5$ Wh (V2) to a peak of over $2.1$ Wh (V1 and V6). Showing that overcoming the baseline's knowledge gaps via retrieval is an intensive operation. Though like explained previously this lacks a better understanding since the evaluation of this domain is just that retrieval occurred, so the straight model never got a correct answer due to that fact. However, we can still compare them, and the V6 instruction stands out once again due to its high energy cost. This is primarily attributed to its reliance on a detailed chain-of-thought process which, while somewhat effective, is significantly more computationally demanding.

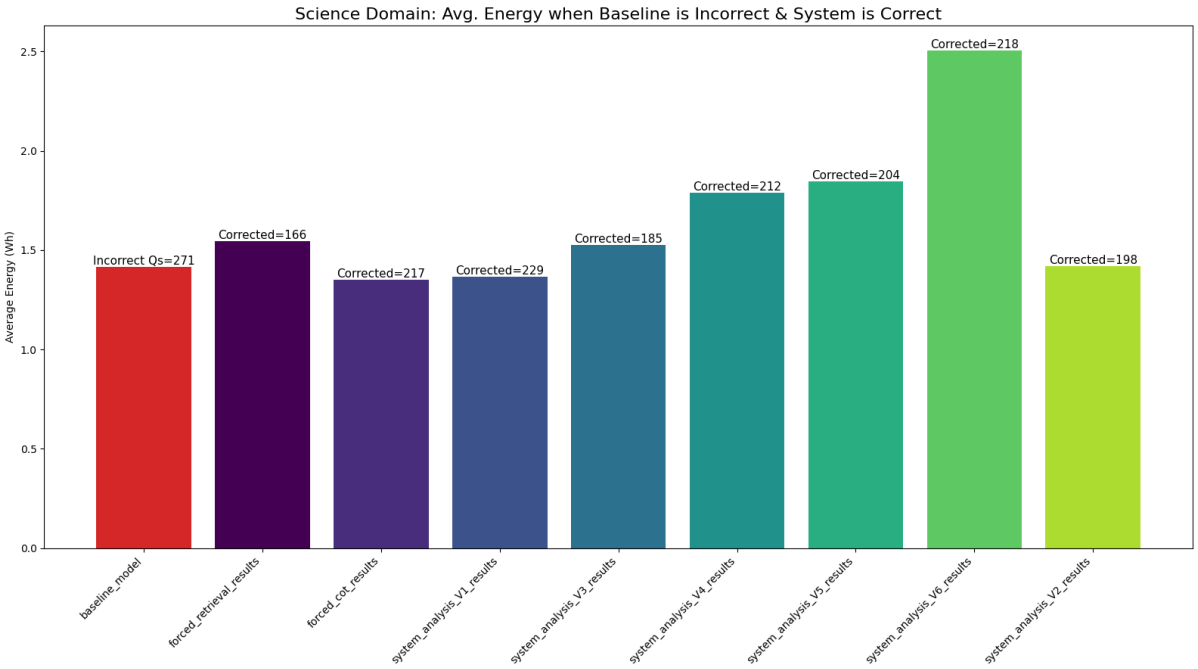



Figure 50: Science: Avg. Energy when Straight Model is Incorrect & System is Correct

A similar trend is observed in the Science domain, as shown in Figure 60. Correcting the baseline in this approach also required a substantial energy investment, with the V6 again showing the highest consumption at nearly 2.5 Wh. Thus reinforcing that the act of correcting the responses regardless of the domain is inherently more energy demanding.

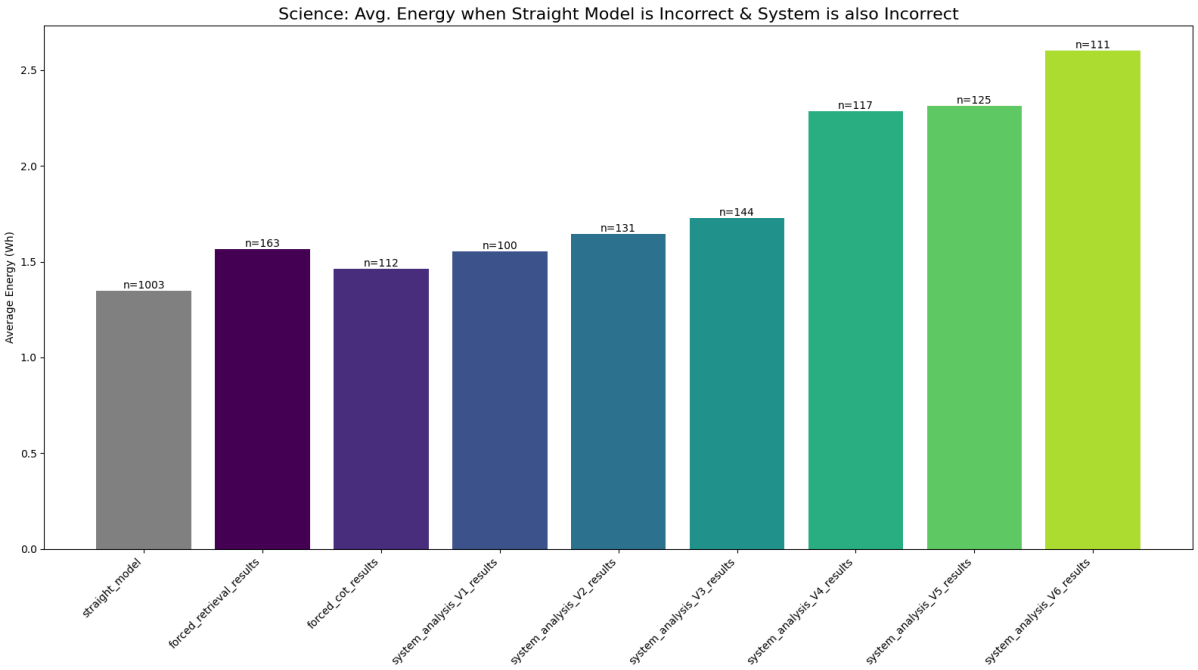



Figure 51: Science: Avg. Energy when Straight Model is Incorrect & System is also Incorrect

On the other hand Figure 51 analyzes the scenario where both the baseline and the system failed to produce a correct answer. This represents the least efficient use of energy, showing that

the system uses additional resources only to arrive at the same incorrect outcome. It is noteworthy that the energy consumed in these failure cases is often comparable to and sometimes even higher than the energy required to produce correct answers. As proof, the V6 system consumes over $2.6$ Wh when it fails, more than it does for a successful answer which is around $2.5$ Wh. This might suggest that these incorrect answeres may result from the system pursuing particularly complex, yet flawed, reasoning paths or even retrieving irrelevant information, leading to wasted resources.

### 6.10.4 Energy Distribution and Consumption Predictability

While the average energy consumption provides a useful high-level metric, a deeper understanding of the system efficiency is required to access its consistency and predictability. A system with a low average cost is less desirable if it is prone to higher spikes. The boxplot in Figure 52 provides valuable insight into all the versions by visually representing the distribution of energy consumption per query for each one.

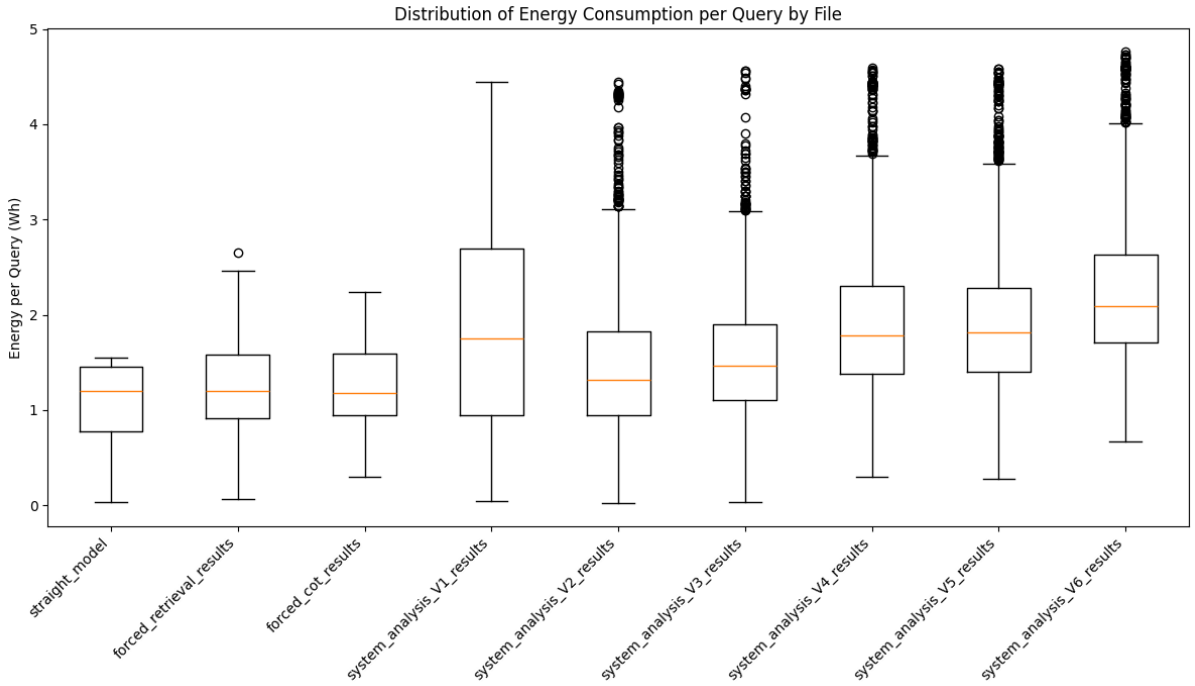



Figure 52: Distribution of Energy Consumption per Query by File

The baseline models, particularly the ”straight_model”, presents the tightest distribution. The small inter quartile range shows that most of the queries are processed using a very consistently and predictably amount of energy. This helps them in terms of reliability since from a resource planing perspective they are very predictable, though they show lower accuracy.

On the opposite side of the spectrum we have the initial routing Instruction V1, that demonstrates extreme unpredictability. It has by far the largest inter-quartile range and a long upper

whisker, indicating a massive variance in energy consumption. This proves that its ambiguity led to highly inefficient and erratic processing.

The following versions from V2 to V5 show a clear trend into an increasing predictability. While they show a higher median energy cost than the baseline, their distributions become tighter and tighter as versions increase. This shows the core benefit of the iterative refinement process, reflecting precisely what was discussed previously as the instructions became more specific and rule-based the model's behavior became more constrained to the rules, therefore more predictable. The number of high energy outliers that caused an increase in system's resource expenditure also decreased indicating a more robust and stable approach.

Lastly the V6 instruction, designed for efficiency presents a rather unique profile. While its median energy is high, its distribution is relatively contained in comparison with V1, this suggests that its Chain-of-Thought process, though energy intensive, is being applied more consistently.

Ultimately, this analysis underscores that effective prompt engineering does more than just improve accuracy, as this proves that it enhances operational predictability. A well designed instruction set not only guides the model to the correct answer but also ensures that the model does so consistently while using a manageable level of resource consumption, which is a critical factor when picking and deploying such systems in the real world, especially in resource constrained environments

### 6.10.5 Overall Performance Quadrant: Synthesizing Accuracy and Efficiency for ARC queries

The culmination of this analysis is best visualized in the performance quadrant plot, which visually presents each system's final correctness rate against its average energy cost for the ARC queries. The graph present in Figure 53, offers a clear overview of the trade-offs and situates the performance of the 8B parameter model (DeepSeek-R1-Distill-Llama-8B) used as the base and all the routing systems routing systems against a crucial benchmark, a much larger 14B parameter model (DeepSeek-R1-Distill-Qwen-14B).

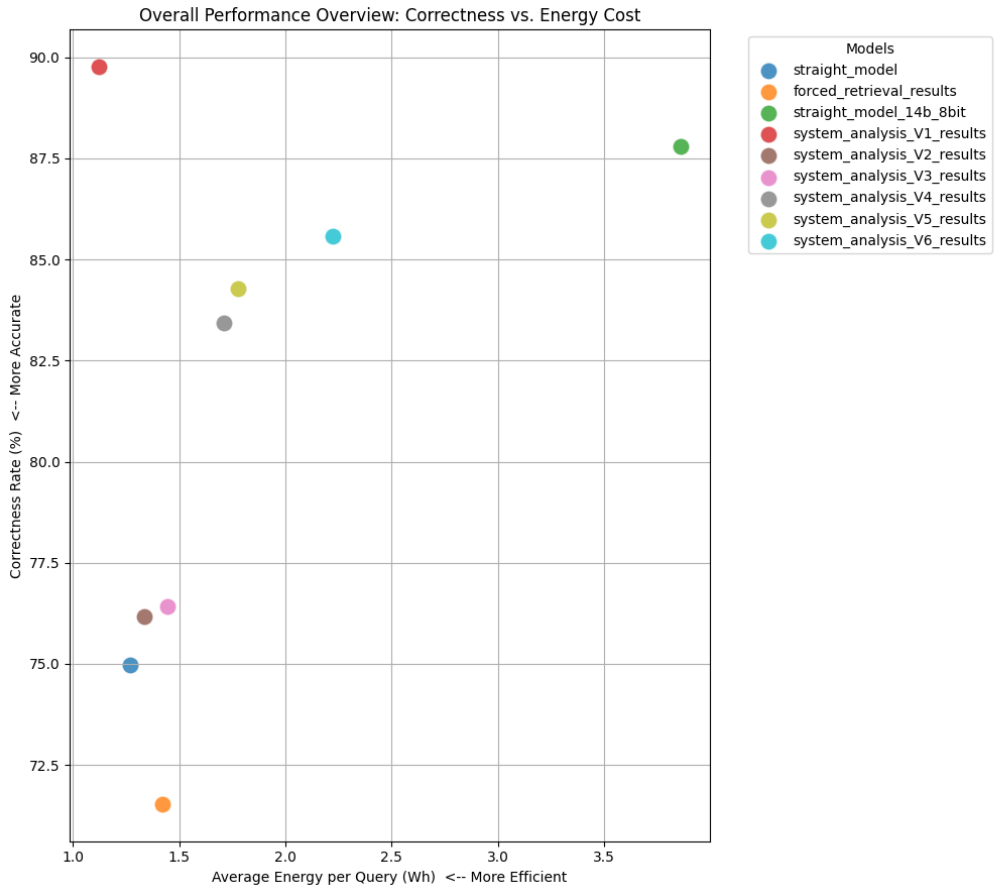



Figure 53: Overall Performance Overview: Correctness vs. Energy Cost for ARC queries

The optimal position on this graph is at the top-left quadrant, which represents the highest accuracy with the lowest energy consumption. The bottom-right represents the inverse so the worst possible outcome . The baseline models establish two reference points. The first is the ”straight_model” (8B), sits in the lower left quadrant, confirming it is a low accuracy but highly efficient option. On the other hand , at the upper-right corner sits the ”straight model 14B 8-bit” demonstrating the brute force approach as it achieves a very high correctness of nearly $90\%$, however this comes with the cost of an enormous amount of energy at around $3.7$ Wh per query,making it by far the least efficient model

The iterative refinement of the routing instructions charts a clear journey toward the ideal quadrant. The initial, poorly tuned instructions (V1,V2,V3) show modest accuracy gains over the 8B baseline but at a notable energy cost, placing them in the lower-middle of the graph.

The major breakthrough occurs with instruction V4 and later V5. These versions represent an optimal sweet spot, achieving a substantial leap in terms of correctness to over $83\%$ while still maintaining an average cost below 1.8 Wh. Most importantly, they deliver a significant portion of the accuracy gains of the 14B model while consuming less than half the energy.

The efficiency-focused V6 instruction pushes the accuracy to over $85\%$, however it also increased the energy cost, moving the results slightly to the right. While it is the most accurate of the routing instructions, it begins to approach a point of diminishing returns in the accuracy-

efficiency trade off.

To conclude, this analysis of the quadrant provides the most complete validation of the research. It shows that an intelligent routing layer with a combination of a carefully engineered prompts are not just incremental improvements. They are part of a transformative strategy of optimization. By using carefully designed instructions to guide a smaller, more efficient 8B model, the system achieves a performance profile that rivals a modelnearlyt twice its size, but at a fraction of the computational and energy costs. This goes to show that architectural and instructional refinement can be a more sustainable and effective approach to high performance,rather than simply scaling up the model size.

# 7 Experimental Consistency and Reproducibility

To guarantee that the performance and energy consumption results represented in this thesis are both valid and reliable, a strict protocol was established to try to minimize the number of variables and create a consistent testing environment for all experiments. The following measures were systematically implemented to ensure that the results shown can be directly attributed to the changes in the system's instruction design and model performance.

First, the test system was maintained in a controlled and isolated state. To prevent interference from many background tasks associated with other programs, and also with the operating system tasks, such as automatic updates or network-related tasks, the machine's internet connection was physically disabled for the duration of all test runs. To further improve repeatability, prior to initiating any experiment, all non-essential background applications and services were fully terminated. The system was then left without any intervention after the scripts started up until they were finished and all the required data was collected. This ensured that the system's computational resources were devoted exclusively to the experimental workload.

Another point taken to maintain the system repeatability was that a static software environment was kept during the entire research process. The main components of the system, including the Python interpreter, the PyCharm IDE, and the HWiNFO monitoring utility, were not updated after the initial setup. This strategy is crucial to eliminate the risk of major software updates introducing performance variations that could skew the results.

Finally, and most critically, the underlying code base for the query-routing system and model inference remained identical across all comparative experiments. When testing the efficacy of different instructions, the sole modification between each run was the content of the system instruction itself. This strict isolation of the independent variable ensures that all measured differences in answer accuracy, latency and energy consumption are direct consequences of the prompt engineering strategy, rather than unintended changes in the software that supports it.

# 8 Challenges and Abandoned Approaches

In the course of this research, some promising strategies were explored but ultimately abandoned due to practical constraints, resource limitations, or conflicts with the project's core objectives. This section represents these methodological dead ends, as understanding what these were and their causes could aid further research.

One of the initial strategies considered for enhancing the RAG component was the usage of a hypothetical document similar to HyDE[40]. The technique, which involves generating a hypothetical answer to a query to improve the semantic search for relevant documents, has shown promise in other projects. However, the original HyDE (Answer RQ2) paper recommended the generation of up to eight hypothetical documents per retrieval to achieve optimal performance. In practice, this approach proved to be prohibitively expensive for the presented framework. The computational overhead of generating eight separate documents before the retrieval was done and the formation of the final answer would have dramatically increased both energy consumption and latency, with the latter representing a change that would make the framework basically unusable with the current hardware. This would directly undermine the primary goal of creating a fast and efficient system for resource-constrained environments.

Another considered approach was the use of keyword injection to improve the model's performance on specialized, domain-specific tasks. The hypothesis was that by programmatically inserting key technical terms (e.g., medical terminology for healthcare implementations) into the prompt, the model could be guided toward a more accurate and contextually aware response. This idea was ultimately abandoned due to the immense data curation effort this would need. To be effective, this strategy would necessitate the creation of a large, validated dataset mapping queries to the required keywords for each domain if various were to be involved in testing. Getting hold and verifying the accuracy of this much data would be a substantial research project in itself, and it felt outside of the scope of this project which was focused more towards the general public.

Finally, the scope of model comparison was limited by the hardware available for this research. While the study successfully demonstrates the capabilities of an 8B parameter model on a single GPU workstation, a more comprehensive analysis would have included benchmarks with higher models as well, which would normally require enterprise-grade GPUs. Such tests were not conducted due to a lack of access to these powerful computational resources. As a result, the performance comparison remains focused on demonstrating the significant leap from smaller models to the proposed SLM framework, rather than a broader spectrum of available open-source models.

# 9 Conclusion

This thesis confronted the significant challenge of deploying powerful language models within environments constrained by limited computational resources, strict data privacy regulations, and tight budgets. The prohibitive cost and operational complexities of state-of-the-art Large Language Models pose a significant barrier for small to medium enterprises and regulated institutions. In response, this research proposed and evaluated a novel framework (Answer RQ1) centered on a compact, 8 billion parameter Small Language Model. The core innovation of this work is a dynamic, adaptive query routing system that intelligently triages incoming queries, selecting the most efficient and effective path be it a direct answer, a reasoning-intensive chain of thought, or knowledge augmentation using retrieval-augmented generation.

The results of this study show that architectural control and sophisticated prompt engineering can substantially bridge the performance gap between small, quantized models and their larger, more resource-intensive counterparts. Through iterative refinement of the controller's instruction design, particularly with profile-based prompts, (Answer RQ3) the system achieved an accuracy exceeding $85\%$ on reasoning-focused science questions. Critically, this performance was achieved with significantly greater energy efficiency than would be possible using larger models. The detailed energy profiling revealed a direct correlation between incorrect answers and higher energy consumption, and also showed how different instructions strategically shift the computational load between CPU-intensive retrieval and GPU-intensive generation. Thus confirming that a "smarter, not bigger" approach is a viable path forward.

The implications of this research are both practical and strategic. It provides a tangible blueprint for developing high-quality, cost-effective, and secure question-answering system that can operate on-premise on a single GPU workstation. This work (Answer RQ4) democratizes access to advanced AI capabilities, enabling organizations without massive computational infrastructure to leverage the power of language models. Furthermore, it also contributes to the growing field of sustainable AI by demonstrating that performance and efficiency are not mutually exclusive.

# 10 Future work

The framework developed in this thesis successfully demonstrates that an intelligently controlled Small Language Model can achieve high performance in resource-constrained environments. This research also opens up several compelling points for future investigation that could further enhance the robustness, efficiency, and applicability of this approach.

First a critical area for future research is the system's resilience to irrelevant or misleading information within its knowledge base. The current experiments utilized a corpus that was sometimes relevant to the ARC dataset. A future study could involve another dataset that has no relevant information to the real world, this would split the realities of this dataset versus

the ARC one, this way when retrieval occurred for ARC it wouldn't improve the answer of the system. Though this is part of the advantage of a system like this, as it will always aim for the best possible result.

Second, a novel and promising research direction based on the extracted results. A promising direction would be to explore the relationship between energy consumption and model hallucination. During this work, a correlation was observed between incorrect answers and higher energy usage. This suggests the possibility of identifying a computational signature for hallucination. This could involve analyzing power draw and processing patterns to determine if non-factual or fabricated responses can be detected in real-time based on their energy profile. If a reliable correlation is established, this could lead to the development of a mechanism that flags potential hallucinations as they are being generated, allowing the system to intervene and reroute the query for correction, thereby improving the model's trustworthiness.

Finally, the adaptive routing principles pioneered here for SMLs could be extended to address known inefficiencies in much larger models. State-of-the-art LLMs, despite their power, can sometimes enter unproductive reasoning loops, repeatedly processing the same logic without reaching a conclusion, which wastes significant computational resources. A future implementation could adapt the controller to monitor the reasoning paths of an LLM. By detecting major semantic repetition or a lack of progress, the system could intervene to break the loop, perhaps by injecting new information via RAG or rewriting the prompt. This would not only prevent wasted computation and energy while also improving the reliability of LLMs in complex, multi-step reasoning tasks positioning this framework as a valuable tool for optimizing both small and large language models.

# 11 Images

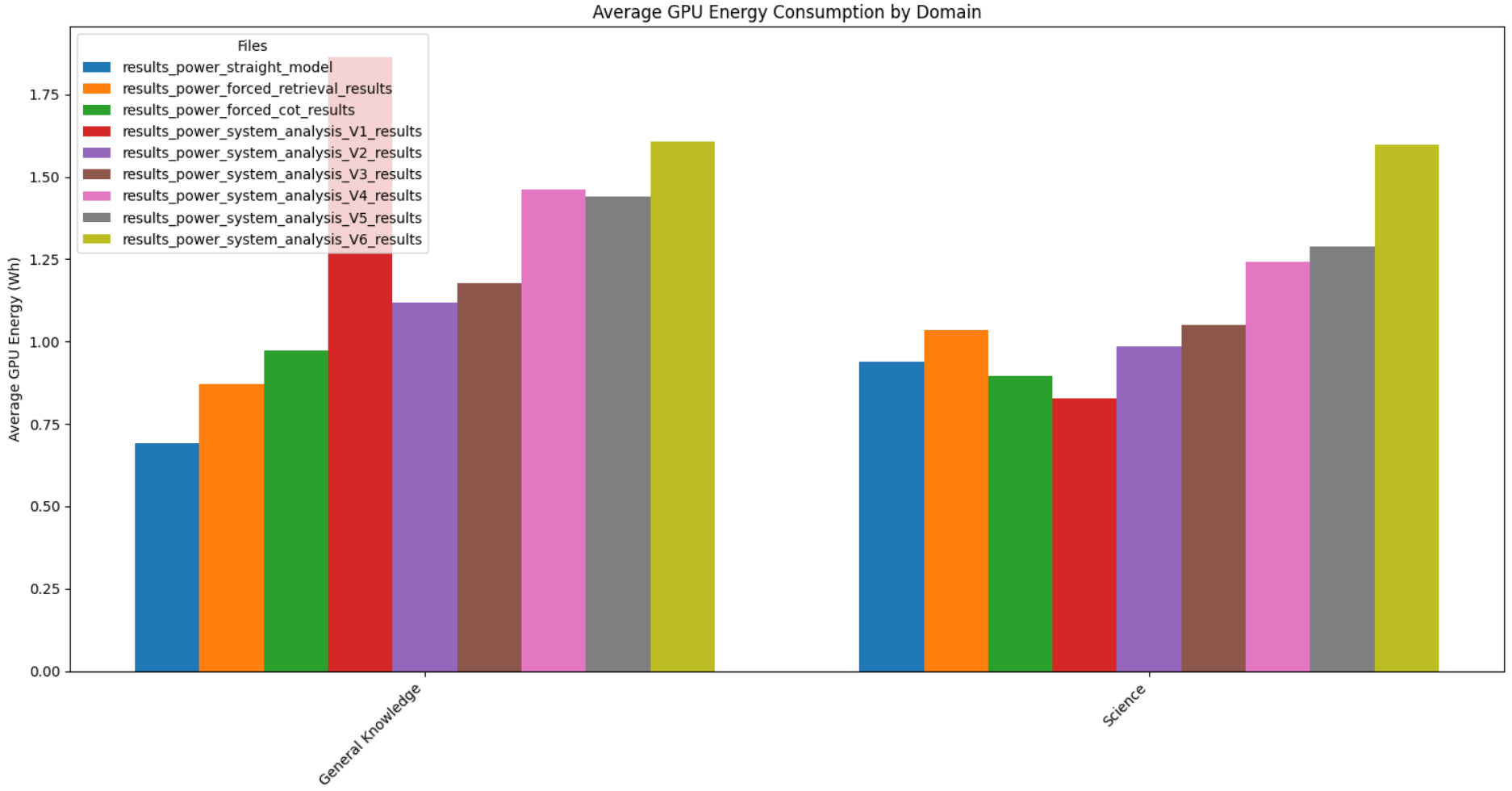



Figure 54: Average GPU Energy Consumption by Domain.

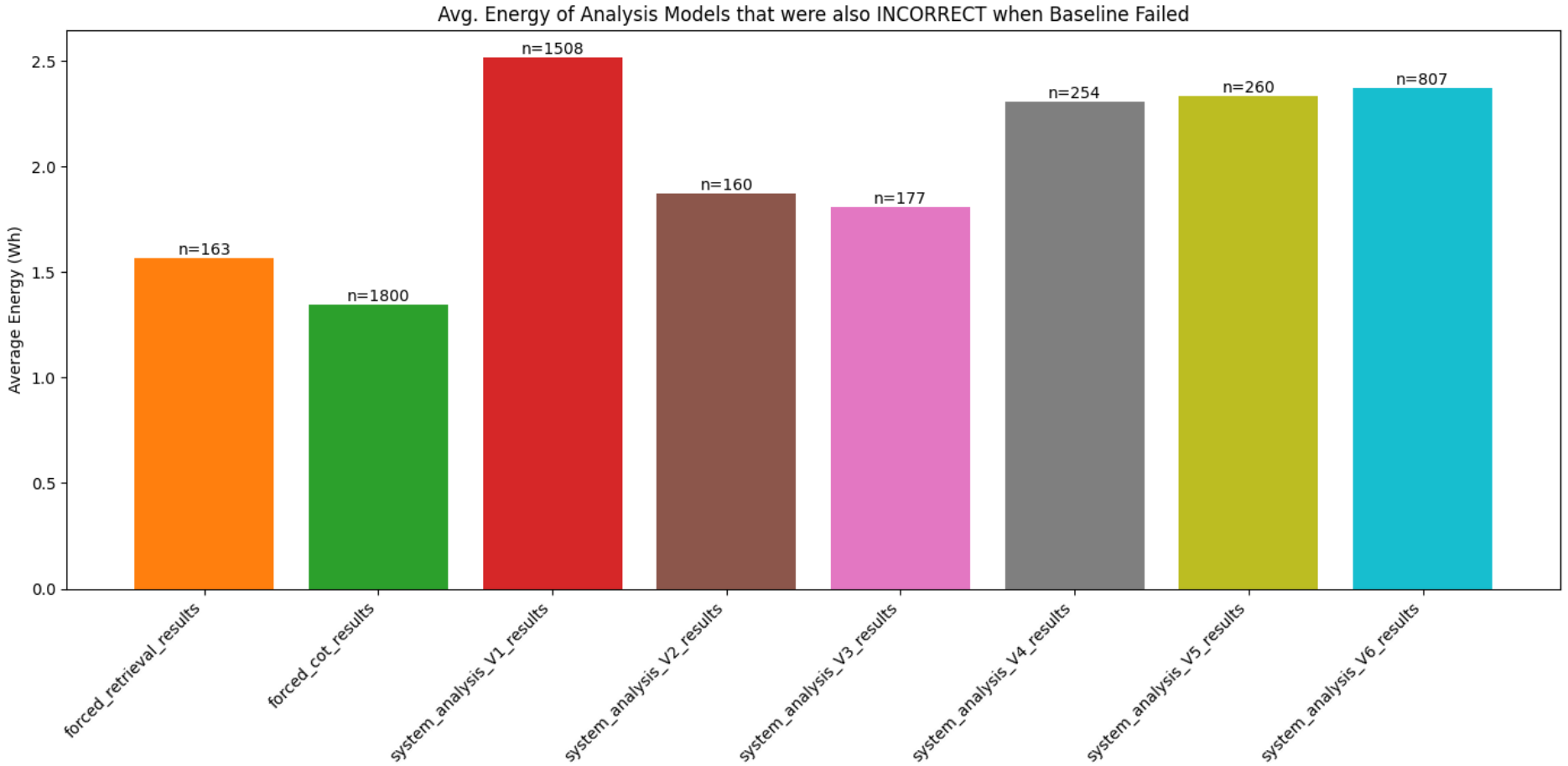



Figure 55: Avg. Energy of Analysis Instructions that were also INCORRECT when Baseline Failed.

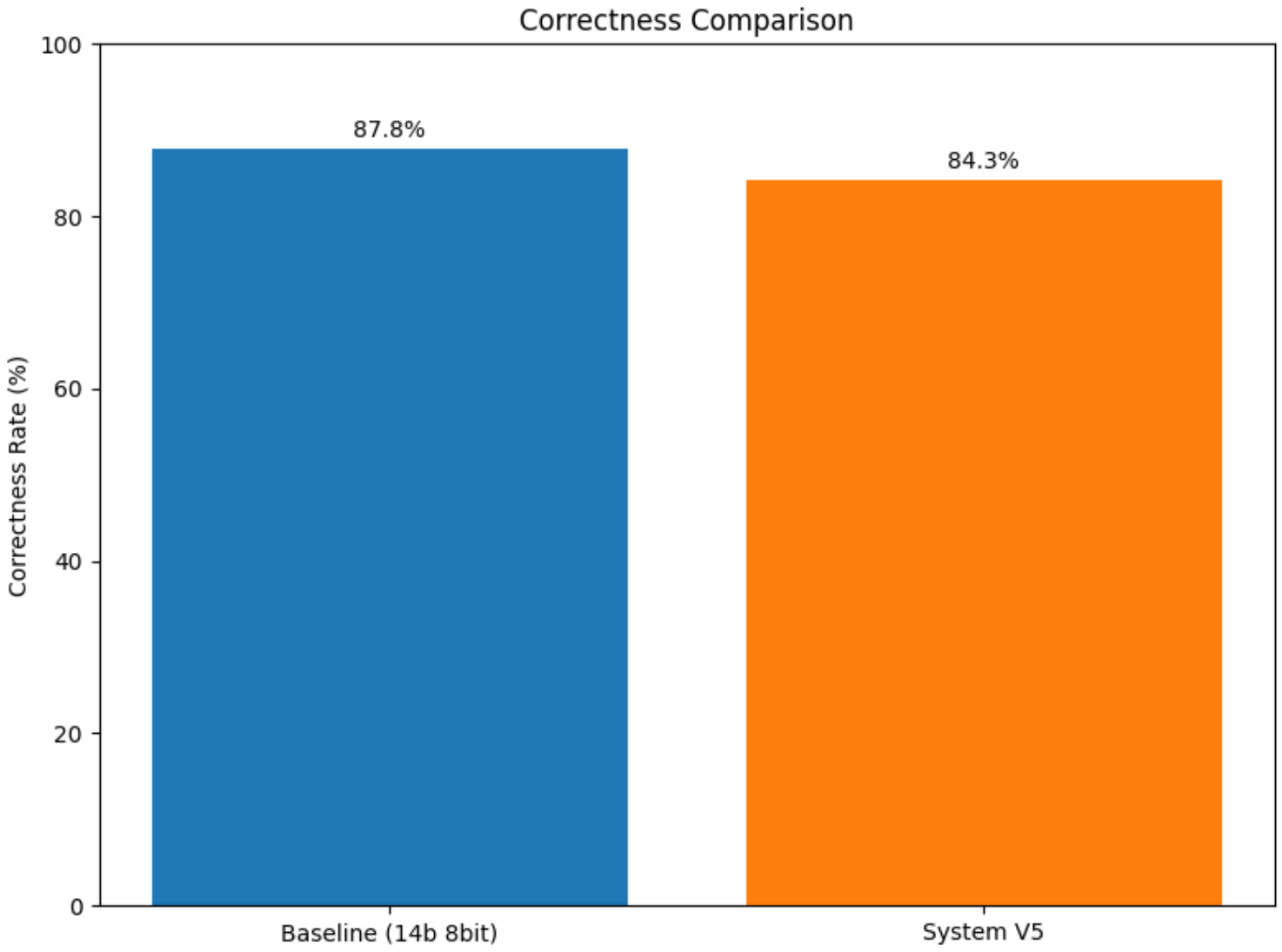



Figure 56: Correctness Comparison between system versus a 14B Model.

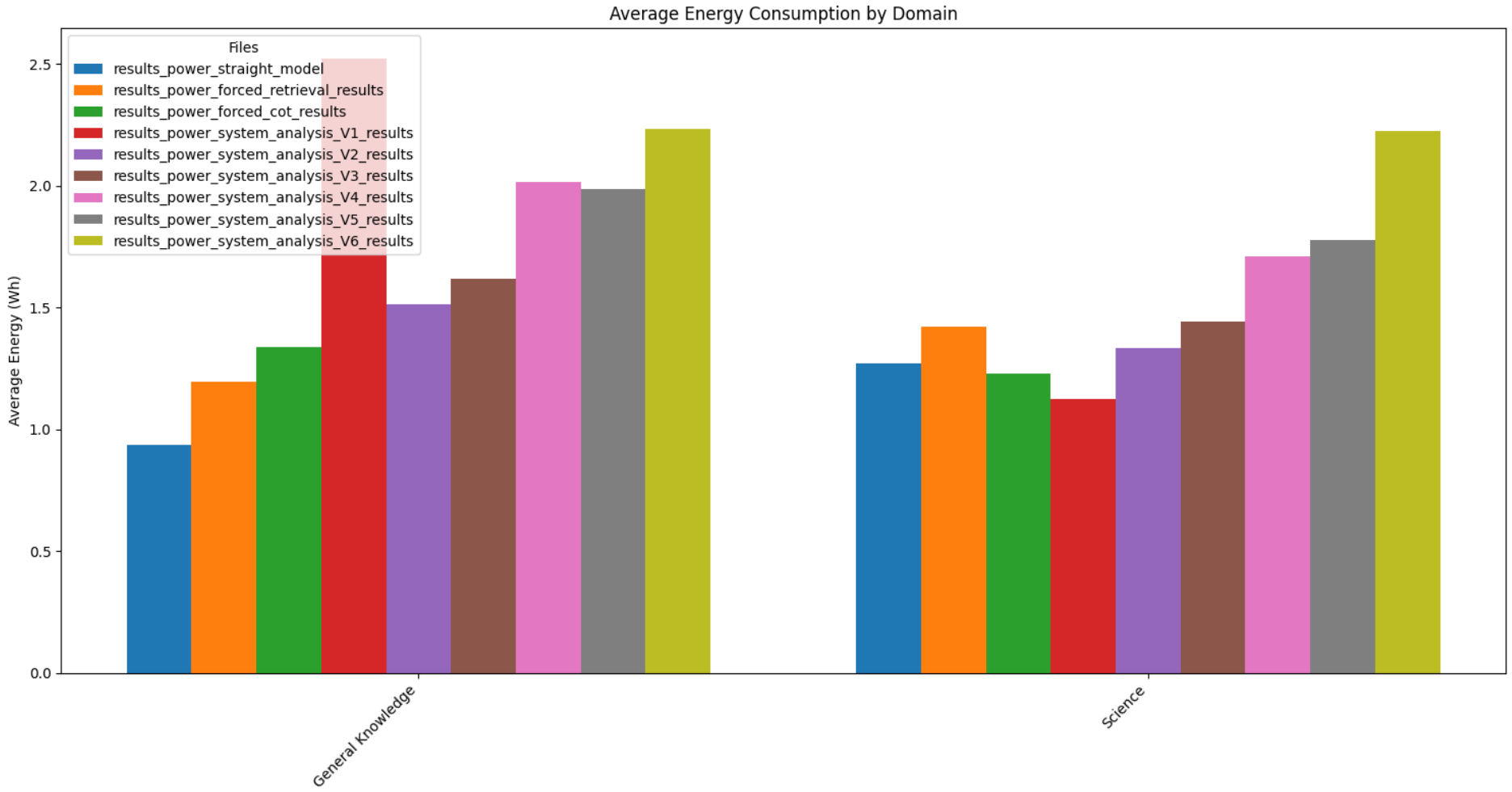



Figure 57: Average Energy Consumption by Domain.

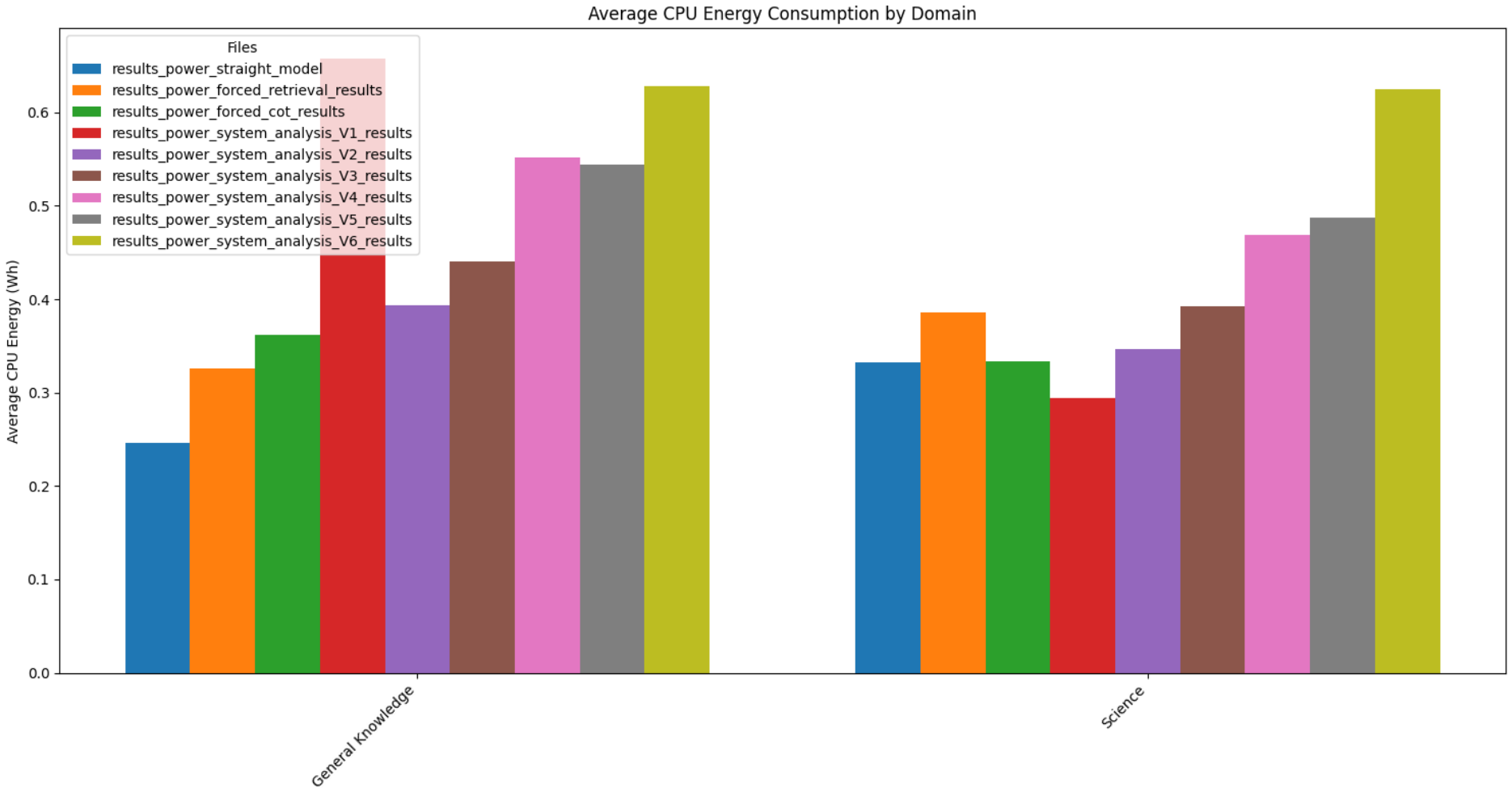



Figure 58: Average CPU Energy Consumption by Domain.

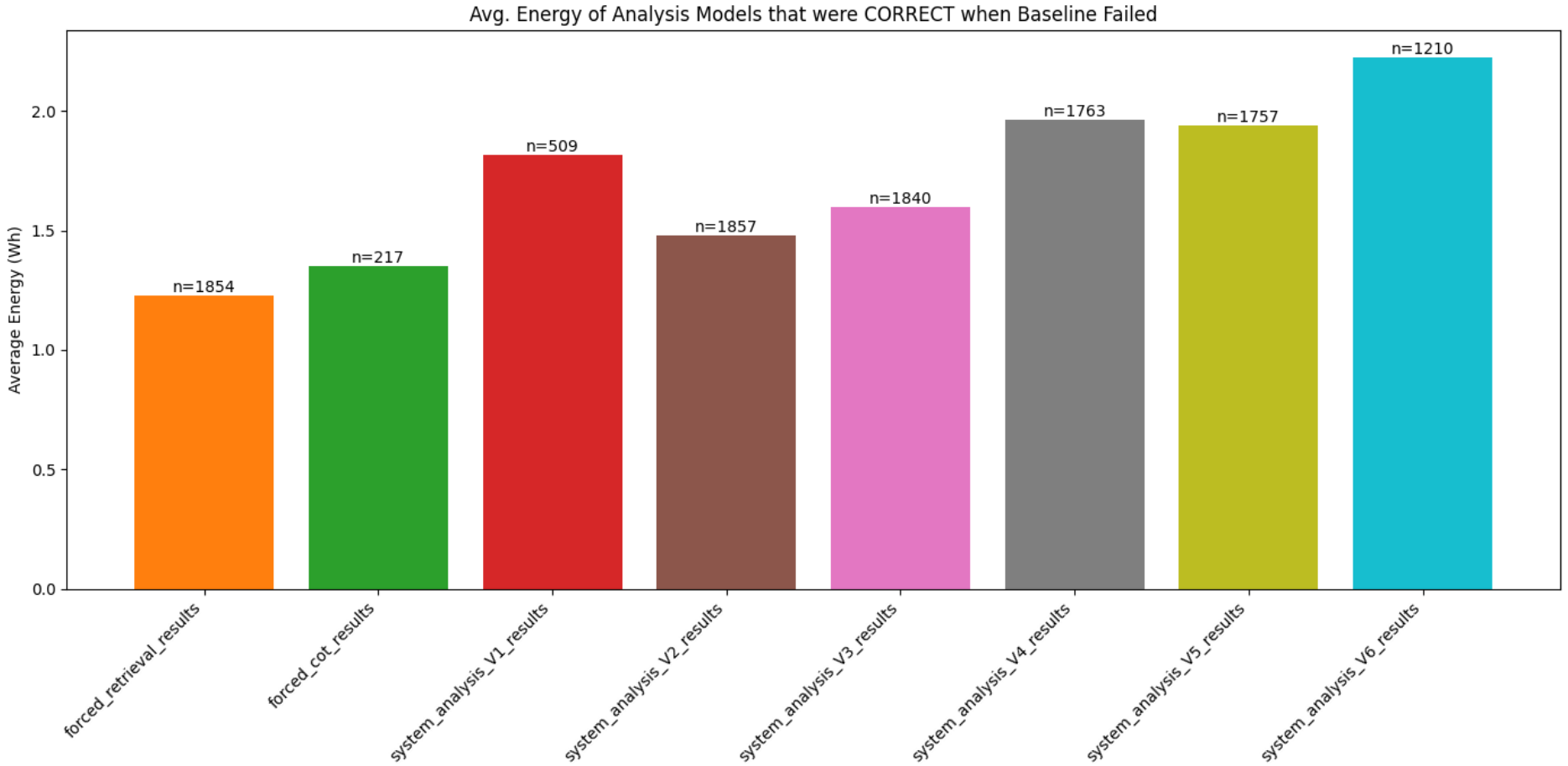



Figure 59: Avg. Energy of Analysis Models that were CORRECT when Baseline Failed.

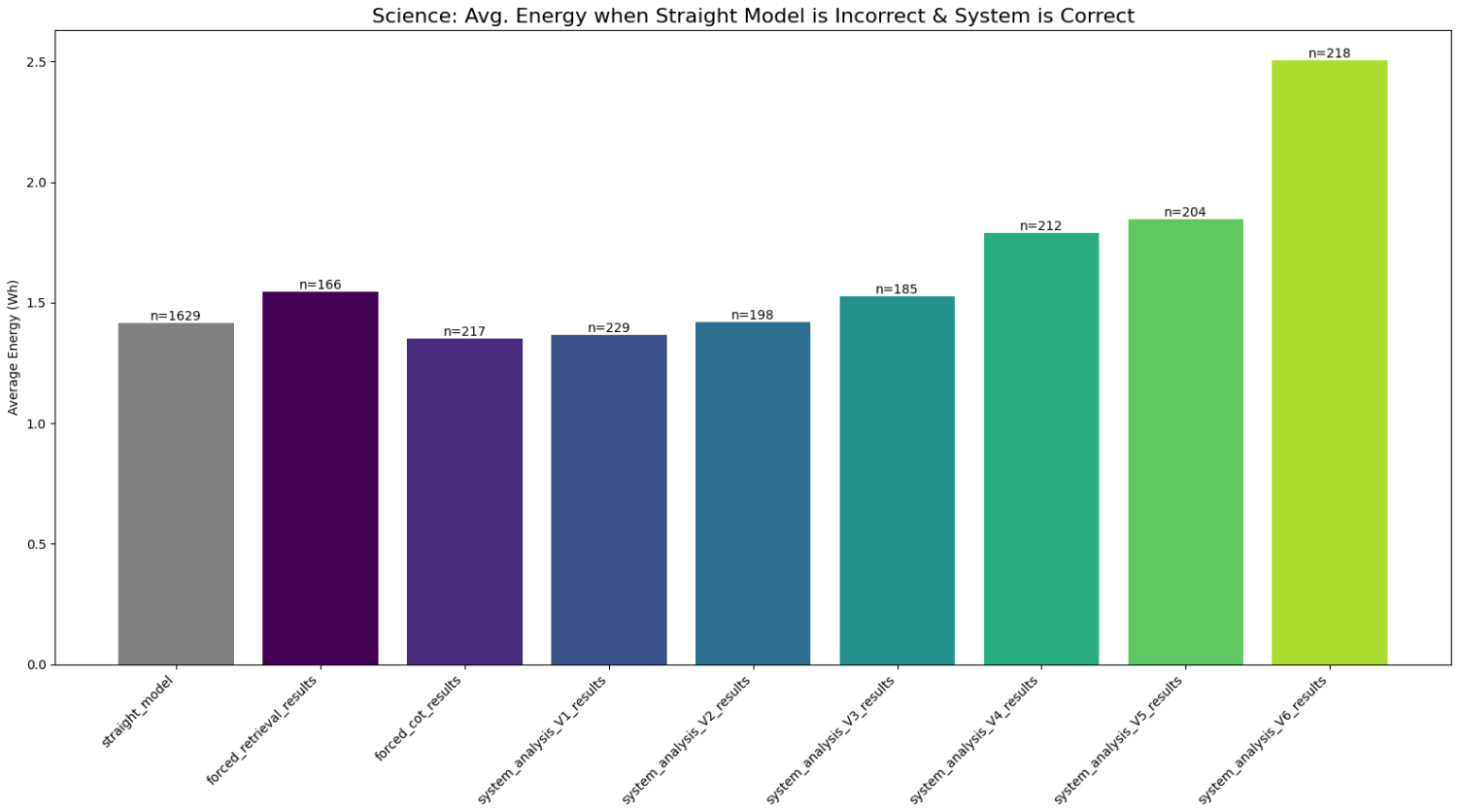



Figure 60: Science: Avg. Energy when Straight Model is Incorrect & System is Correct.

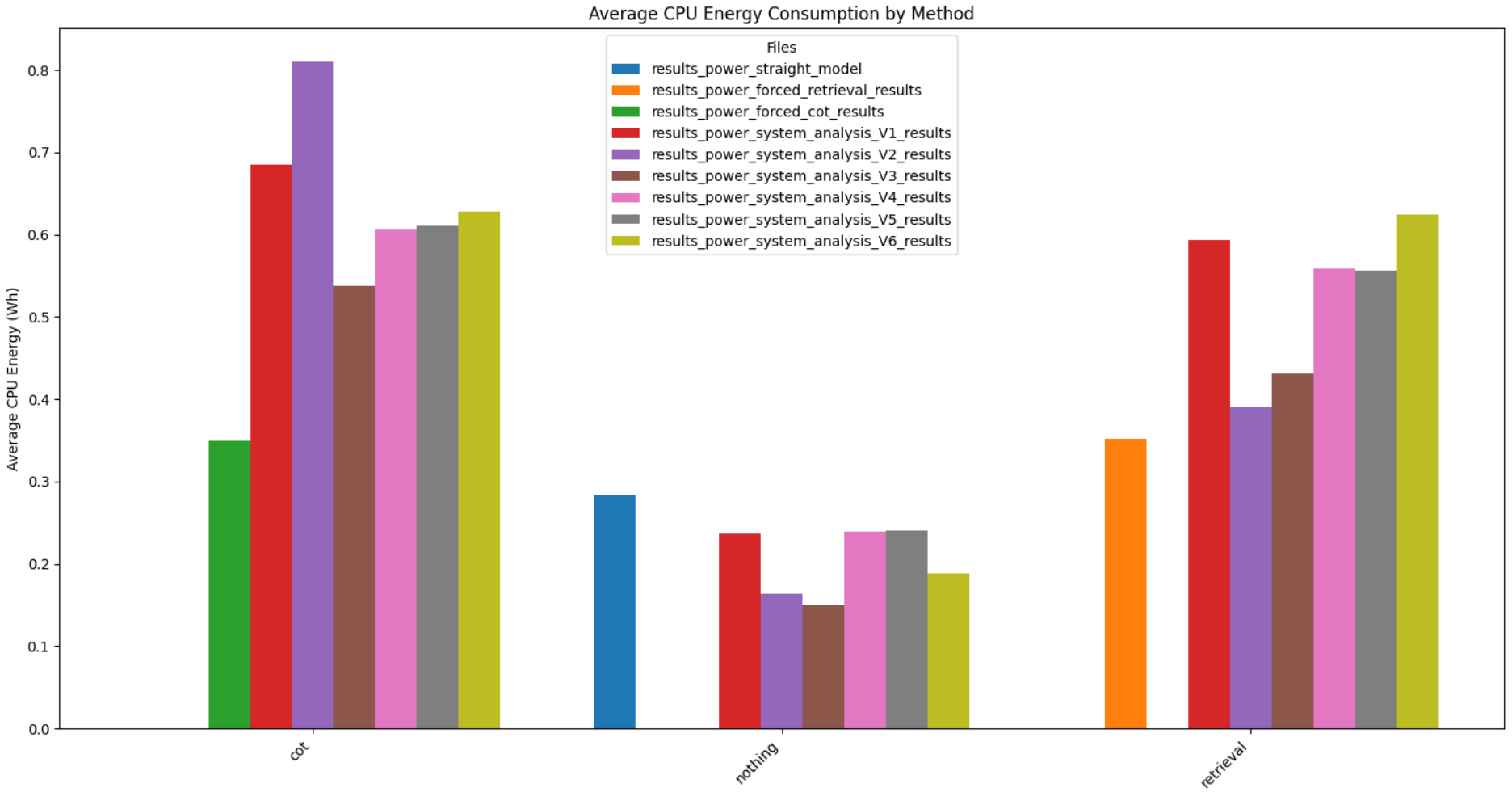



Figure 61: Average CPU Energy Consumption by Method.

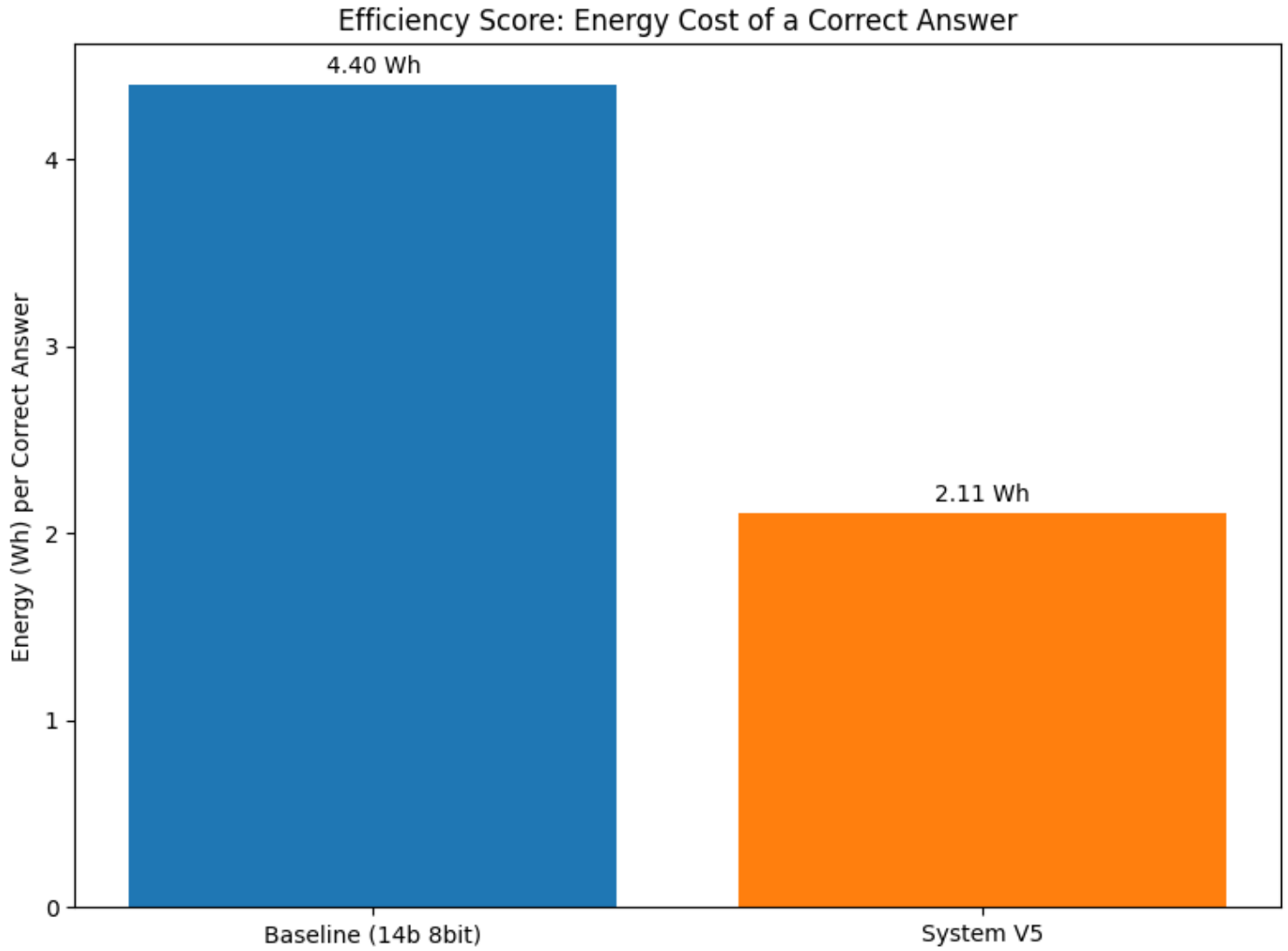



Figure 62: Efficiency Score: Energy Cost of a Correct Answer for the system versus the 14B Model at the science domain.

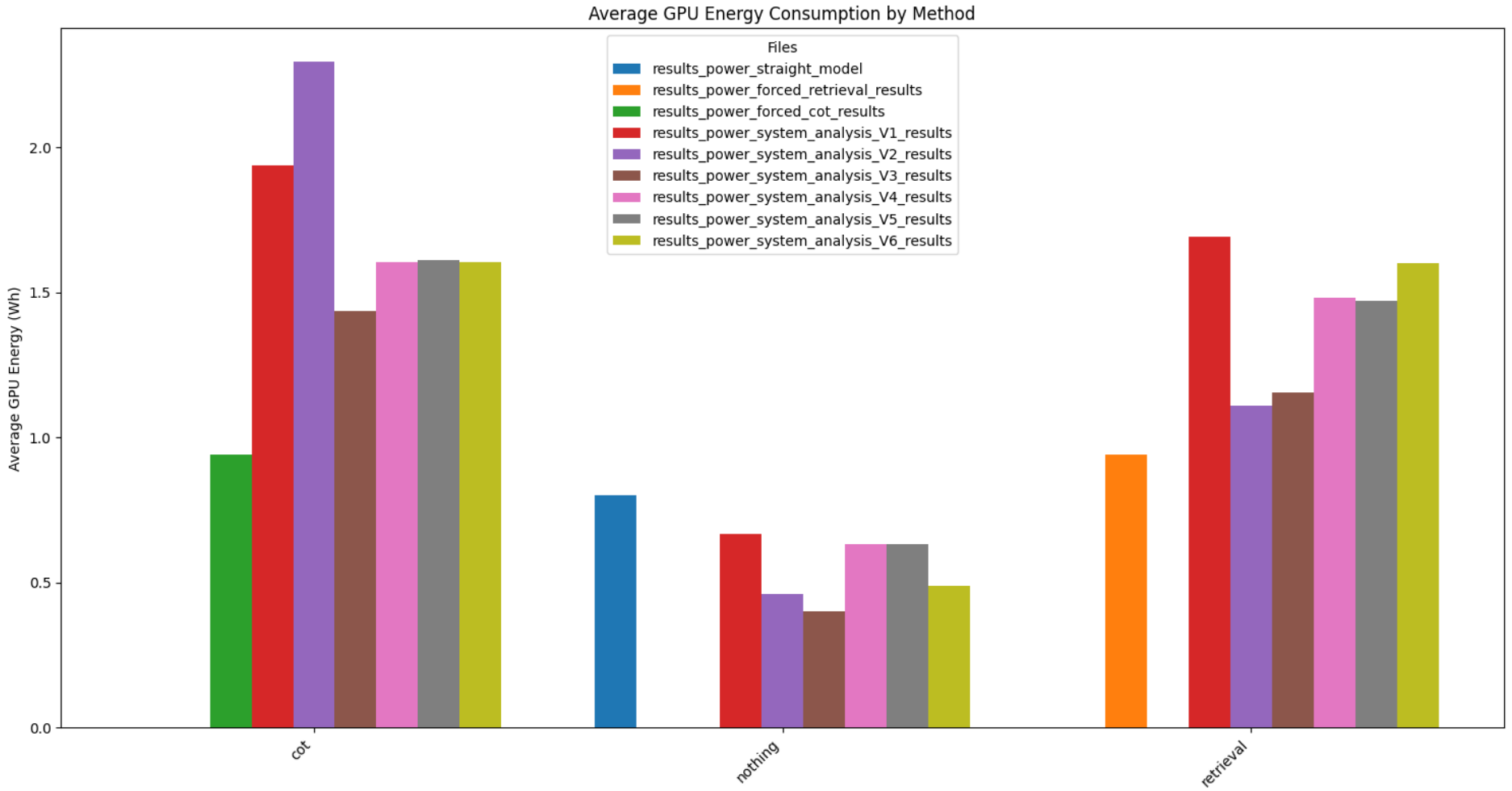



Figure 63: Average GPU Energy Consumption by Method.

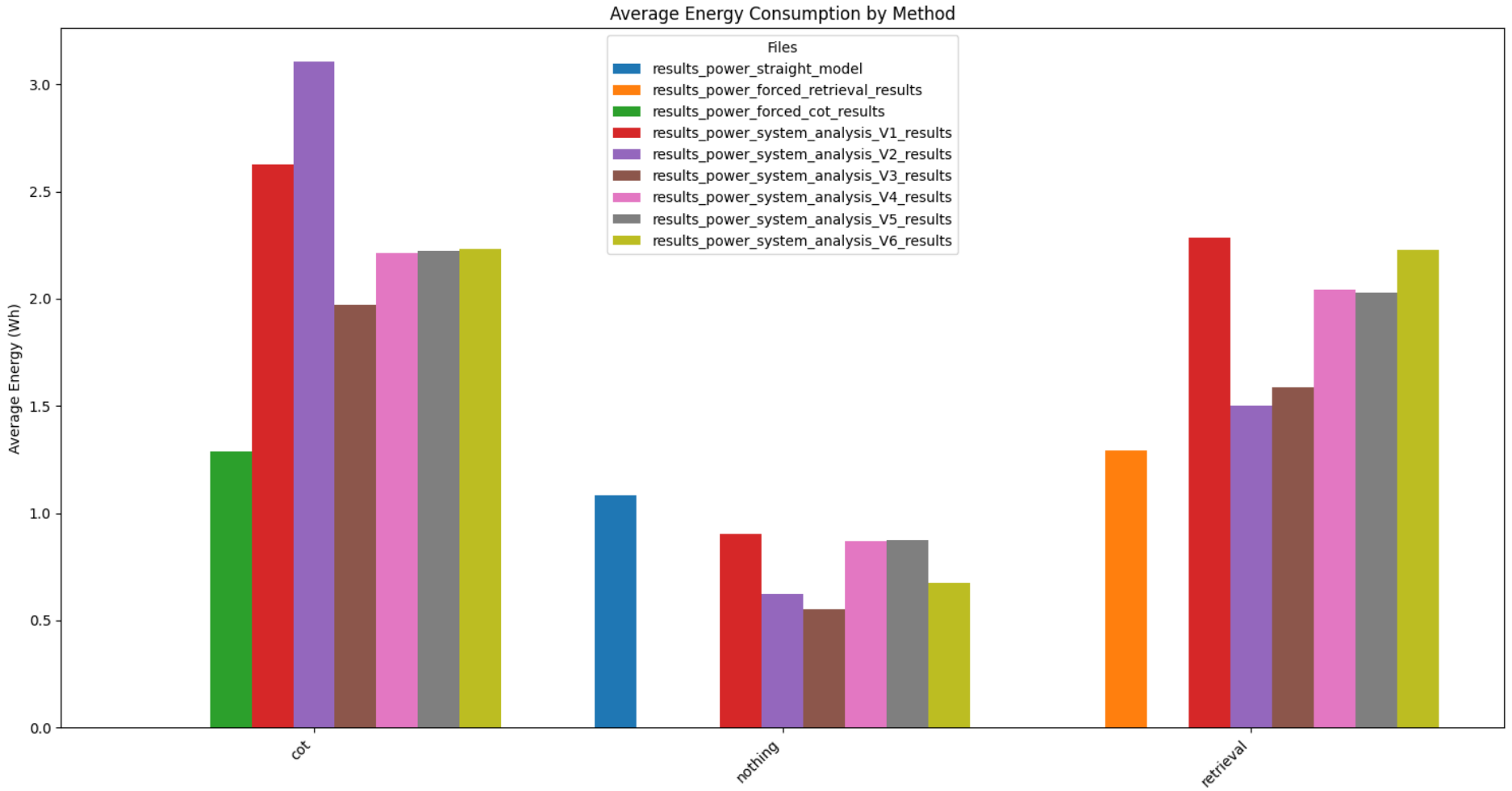



Figure 64: Average Energy Consumption by Method.

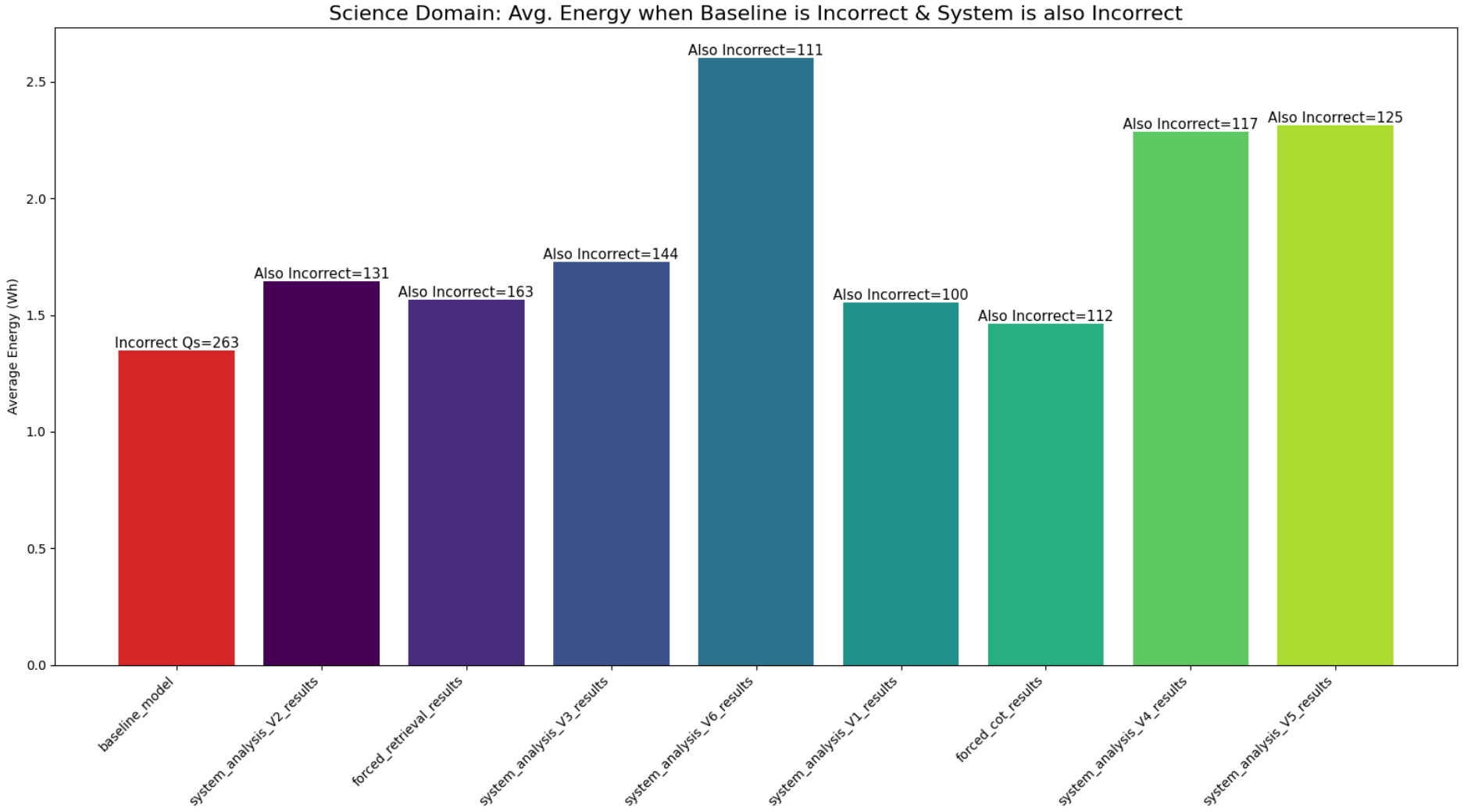



Figure 65: Science Domain: Avg. Energy when Baseline is Incorrect & System is also Incorrect.

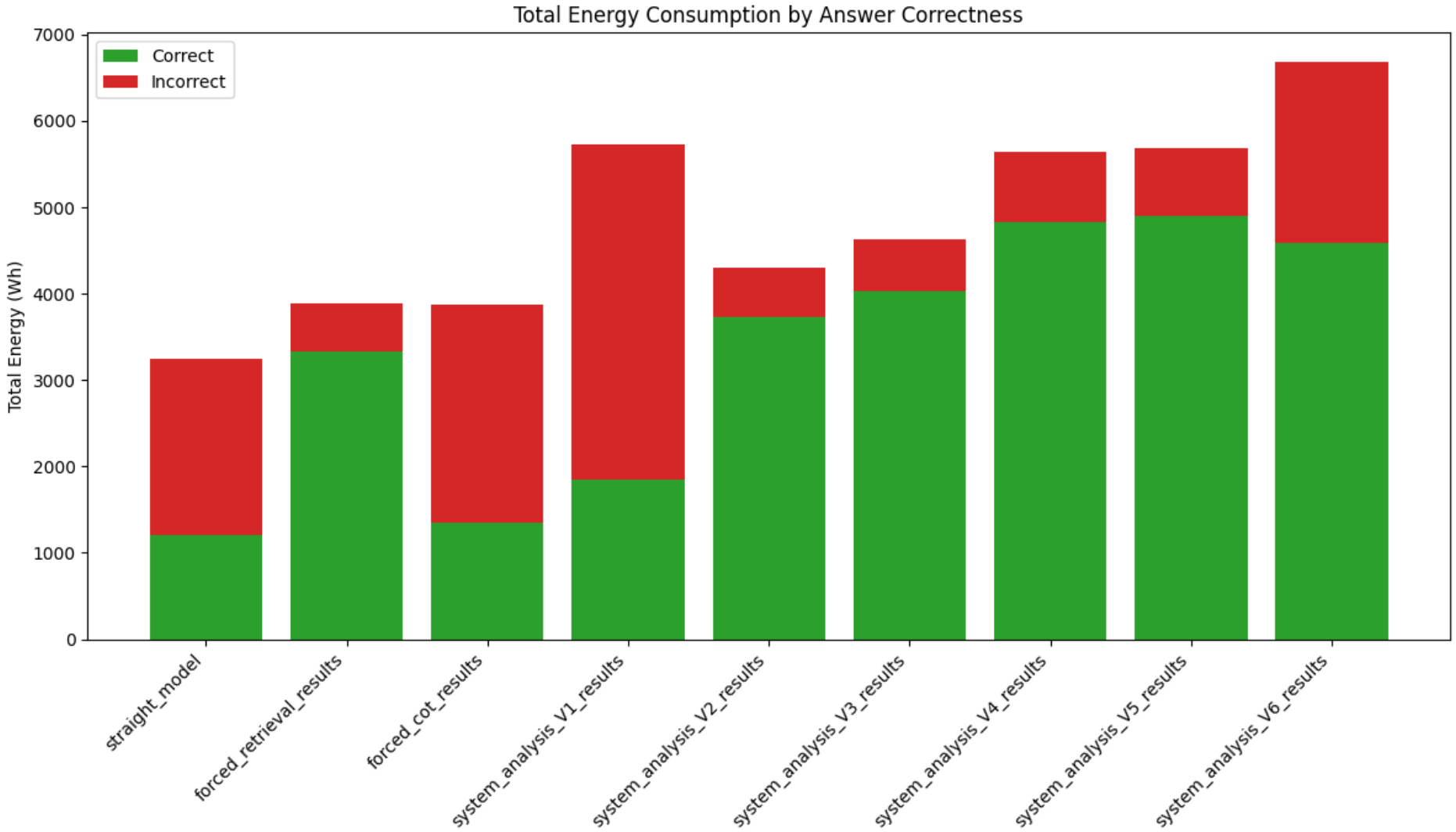



Figure 66: Total Energy Consumption by Answer.

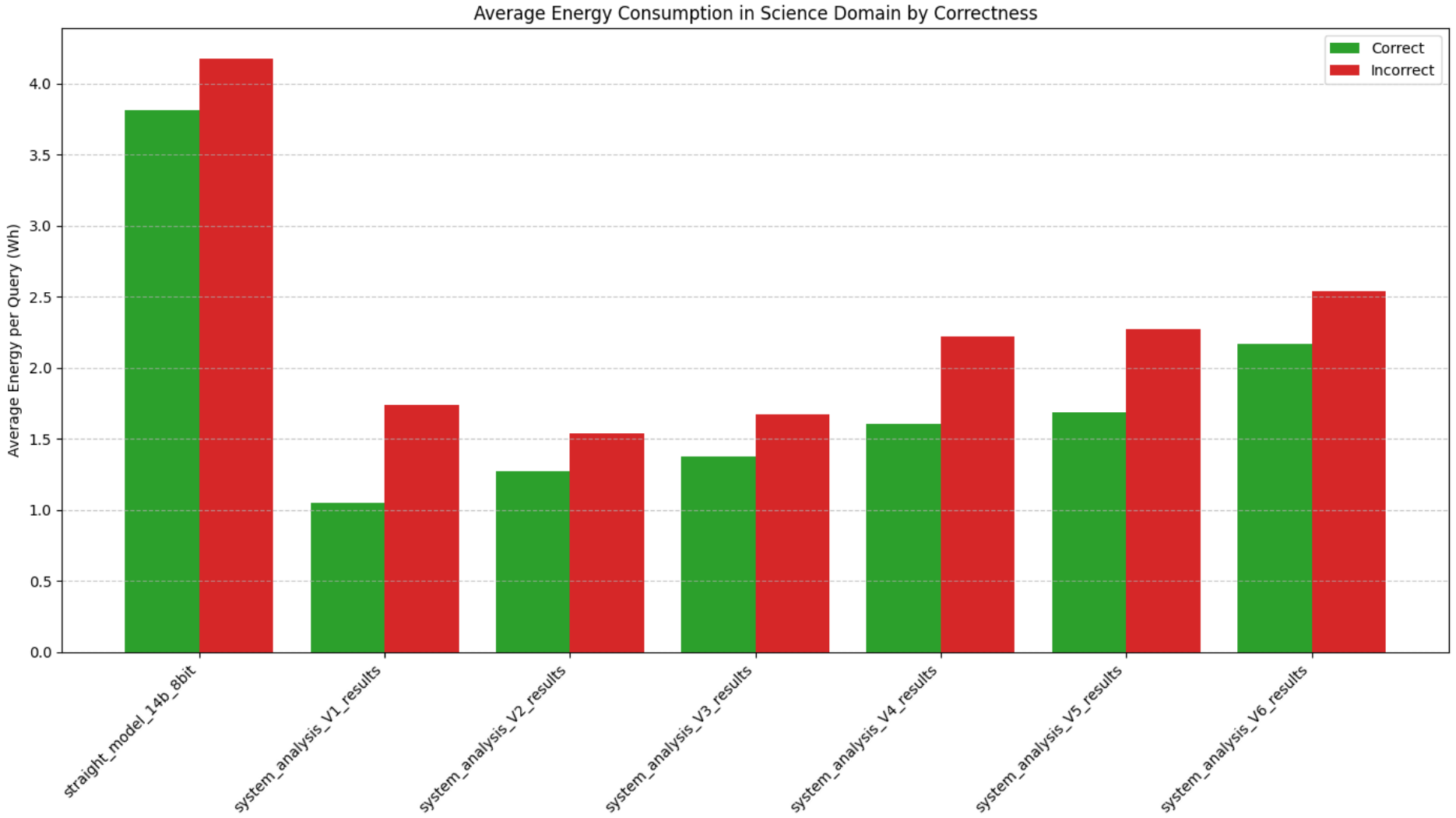



Figure 67: Average Energy Consumption in Science Domain by Correctness.

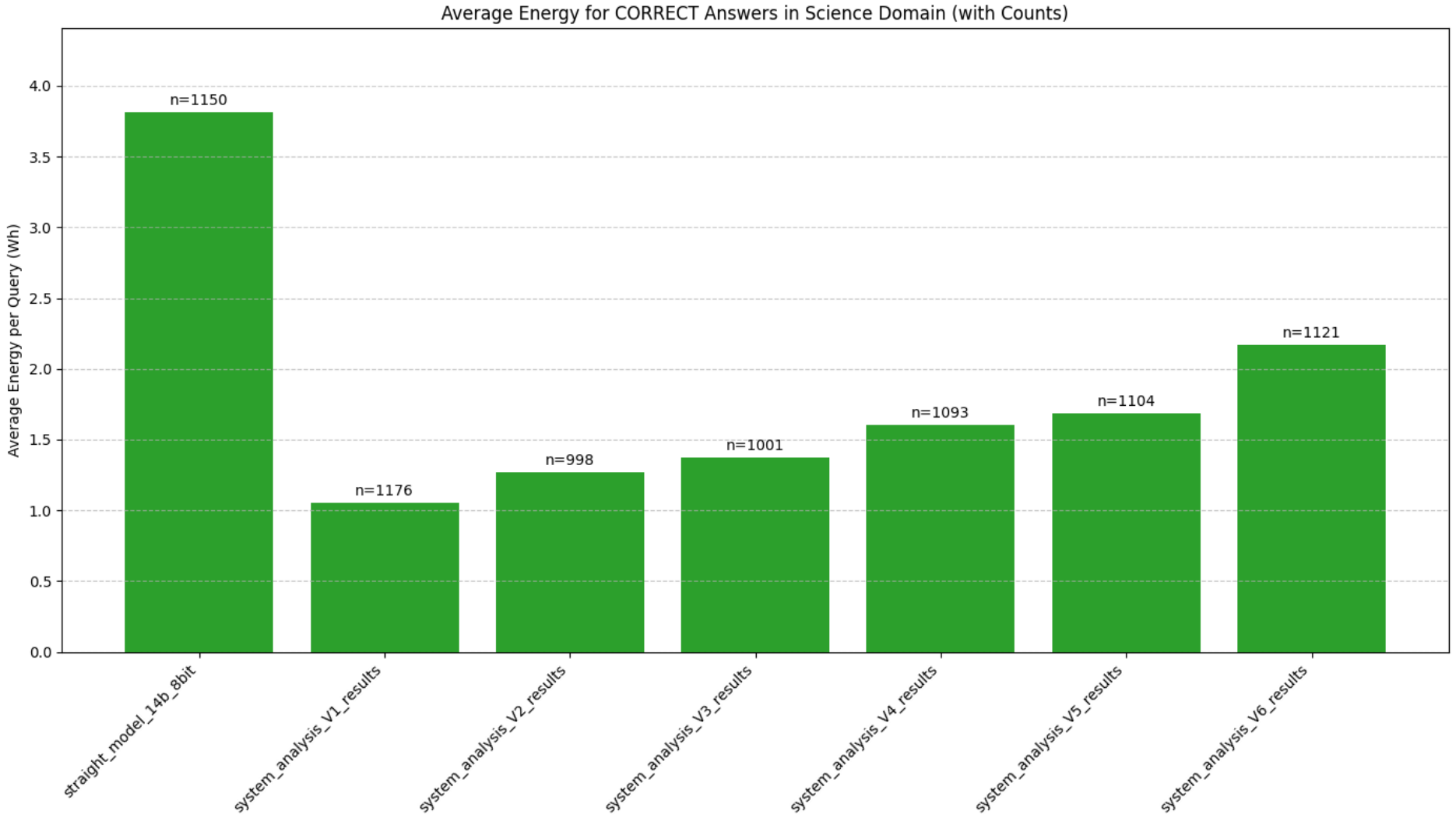



Figure 68: Average Energy for CORRECT Answers in Science Domain (with Counts).

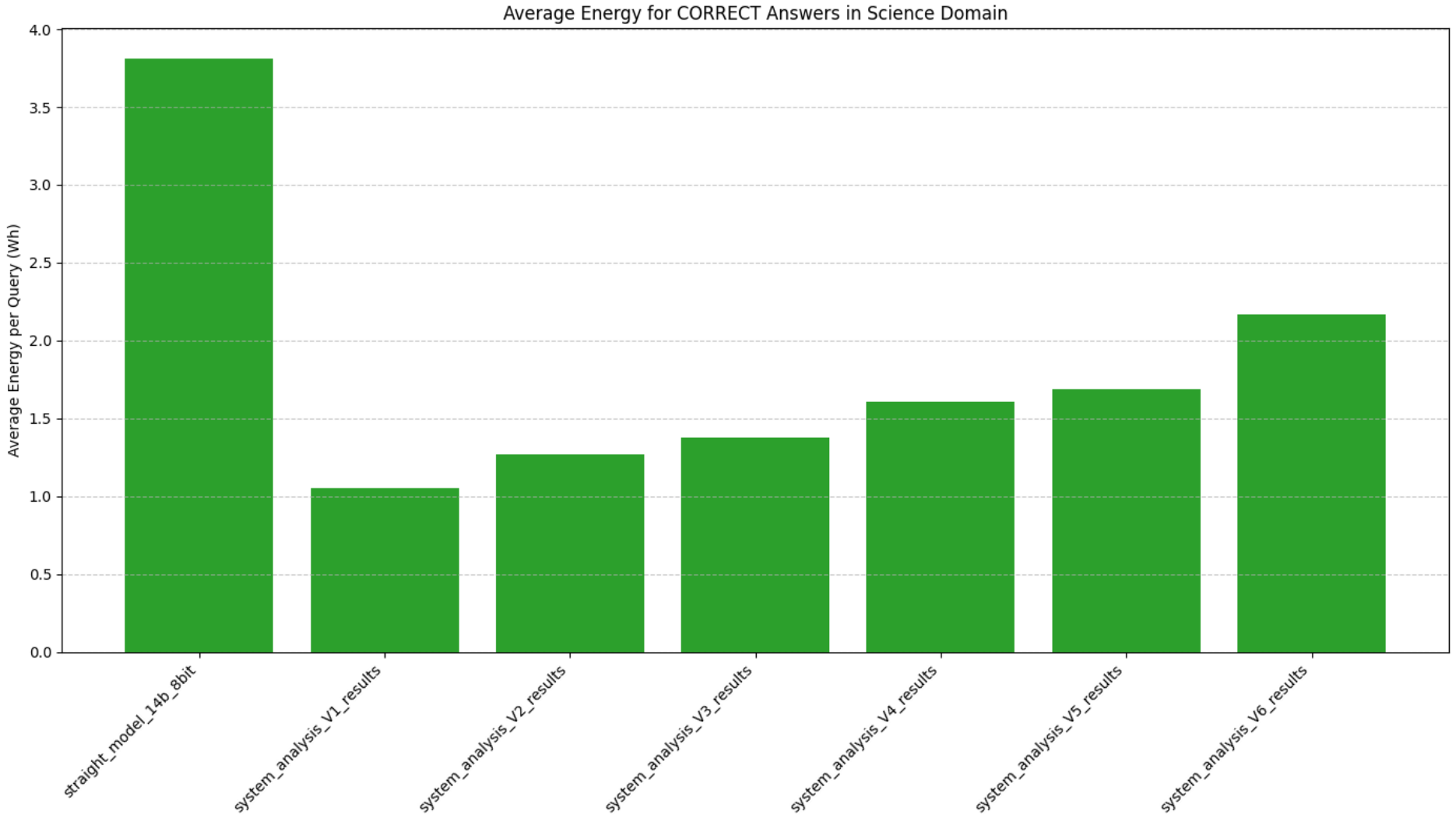



Figure 69: Average Energy for CORRECT Answers in Science Domain.

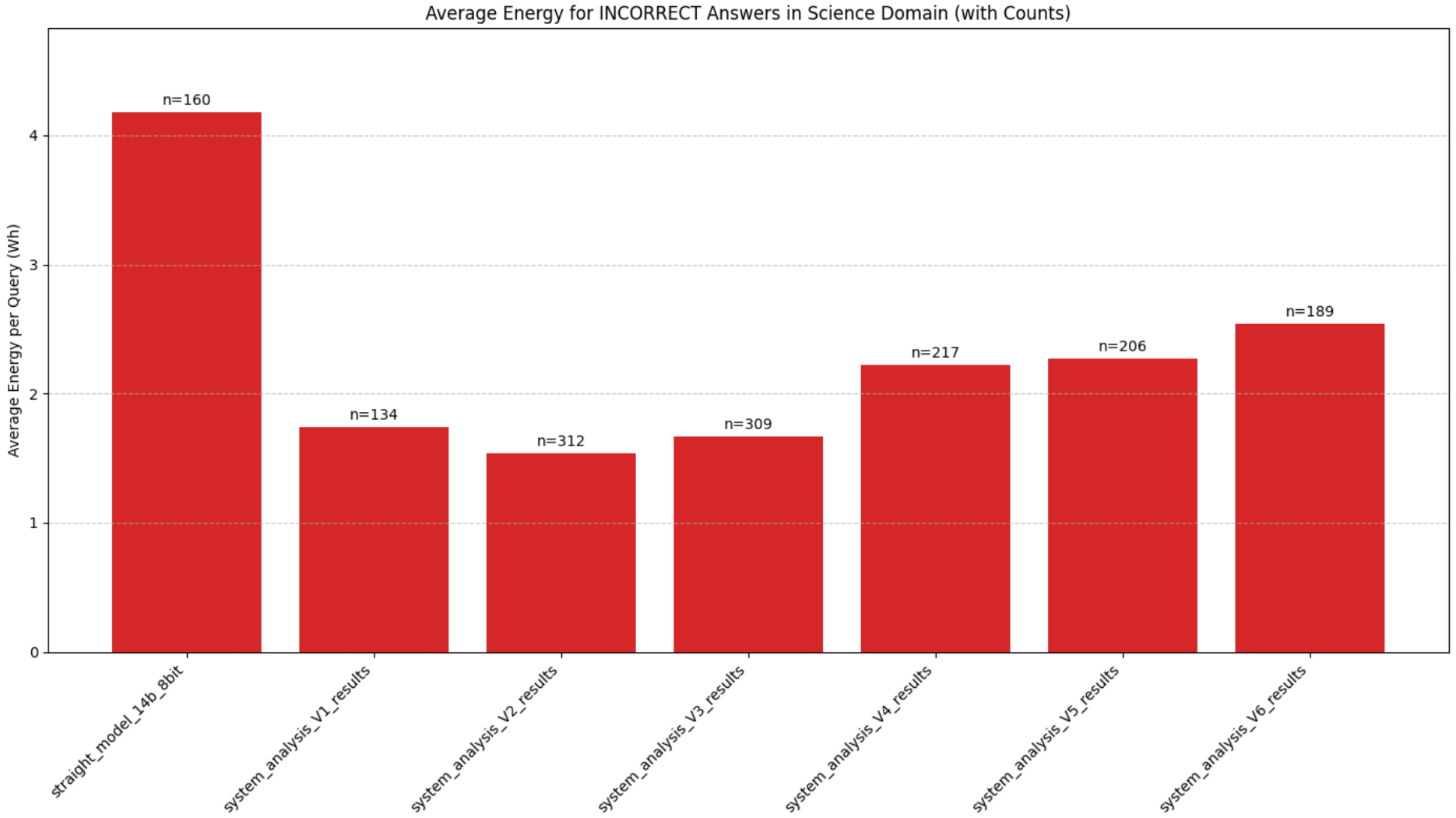



Figure 70: Average Energy for INCORRECT Answers in Science Domain (with Counts).

# References


[1] Wenpeng Yin et al. *Comparative Study of CNN and RNN for Natural Language Processing*. arXiv:1702.01923 [cs]. Feb. 2017. DOI: 10.48550/arXiv.1702.01923. URL: http://arxiv.org/abs/1702.01923 (visited on 01/09/2025).

[2] Jeffrey L. Elman. *Finding Structure in Time*. 1990. DOI: https://doi.org/10.1207/s15516709cog1402_1. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1207/s15516709cog1402_1. URL: https://onlinelibrary.wiley.com/doi/abs/10.1207/s15516709cog1402_1.

[3] Neural Computation. "Long short-term memory". In: *Neural Comput* 9 (2016), pp. 1735–1780. URL: https://interactiveaudiolab.github.io/teaching/casa/HorchreiterSchmidhuber_LSTM.pdf (visited on 01/09/2025).

[4] Kyunghyun Cho et al. *On the Properties of Neural Machine Translation: Encoder-Decoder Approaches*. arXiv:1409.1259 [cs]. Oct. 2014. DOI: 10.48550/arXiv.1409.1259. URL: http://arxiv.org/abs/1409.1259 (visited on 01/09/2025).

[5] Ashish Vaswani et al. "Attention is All you Need". In: *Advances in Neural Information Processing Systems*. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845
Abstract.html (visited on 01/08/2025).

[6] Sofia Serrano and Noah A. Smith. *Is Attention Interpretable?* arXiv:1906.03731 [cs]. June 2019. DOI: 10.48550/arXiv.1906.03731. URL: http://arxiv.org/abs/1906.03731 (visited on 01/09/2025).

[7] *[2006.16362] Multi-Head Attention: Collaborate Instead of Concatenate*. URL: https://arxiv.org/abs/2006.16362 (visited on 01/10/2025).

[8] Katikapalli Subramanyam Kalyan, Ajit Rajasekharan, and Sivanesan Sangeetha. *AMMUS : A Survey of Transformer-based Pretrained Models in Natural Language Processing*. arXiv:2108.05542 [cs]. Aug. 2021. DOI: 10.48550/arXiv.2108.05542. URL: http://arxiv.org/abs/2108.05542 (visited on 01/11/2025).

[9] Tom B. Brown et al. *Language Models are Few-Shot Learners*. arXiv:2005.14165 [cs]. July 2020. DOI: 10.48550/arXiv.2005.14165. URL: http://arxiv.org/abs/2005.14165 (visited on 01/11/2025).

[10] Wei Zeng et al. *PanGu-$\alpha$: Large-scale Autoregressive Pretrained Chinese Language Models with Auto-parallel Computation*. arXiv:2104.12369 [cs]. Apr. 2021. DOI: 10.48550/arXiv.2104.12369. URL: http://arxiv.org/abs/2104.12369 (visited on 01/11/2025).

[11] Dmitry Lepikhin et al. *GShard: Scaling Giant Models with Conditional Computation and Automatic Sharding*. arXiv:2006.16668 [cs]. June 2020. DOI: 10.48550/arXiv.2006.16668. URL: http://arxiv.org/abs/2006.16668 (visited on 01/11/2025).

[12] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. arXiv:1810.04805 [cs]. May 2019. DOI: 10.48550/arXiv.1810.04805. URL: http://arxiv.org/abs/1810.04805 (visited on 01/08/2025).

[13] Microsoft. *microsoft/Phi-3-mini-4k-instruct at main*. Sept. 2024. URL: https://huggingface.co/microsoft/Phi-3-mini-4k-instruct/tree/main (visited on 01/13/2025).

[14] Yanshu Wang et al. *Art and Science of Quantizing Large-Scale Models: A Comprehensive Overview*. arXiv:2409.11650 [cs]. Sept. 2024. DOI: 10.48550/arXiv.2409.11650. URL: http://arxiv.org/abs/2409.11650 (visited on 01/08/2025).

[15] Zhewei Yao et al. *ZeroQuant-V2: Exploring Post-training Quantization in LLMs from Comprehensive Study to Low Rank Compensation*. arXiv:2303.08302 [cs]. May 2023. DOI: 10.48550/arXiv.2303.08302. URL: http://arxiv.org/abs/2303.08302 (visited on 01/08/2025).

[16] Wenxiao Wang et al. *Model Compression and Efficient Inference for Large Language Models: A Survey*. arXiv:2402.09748 [cs]. Feb. 2024. DOI: 10.48550/arXiv.2402.09748. URL: http://arxiv.org/abs/2402.09748 (visited on 01/08/2025).

[17] Gunho Park et al. *LUT-GEMM: Quantized Matrix Multiplication based on LUTs for Efficient Inference in Large-Scale Generative Language Models*. arXiv:2206.09557 [cs]. Apr. 2024. DOI: 10.48550/arXiv.2206.09557. URL: http://arxiv.org/abs/2206.09557 (visited on 01/08/2025).

[18] Sehoon Kim et al. *SqueezeLLM: Dense-and-Sparse Quantization*. arXiv:2306.07629 [cs]. June 2024. DOI: 10.48550/arXiv.2306.07629. URL: http://arxiv.org/abs/2306.07629 (visited on 01/08/2025).

[19] Elias Frantar et al. *GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers*. arXiv:2210.17323 [cs]. Mar. 2023. DOI: 10.48550/arXiv.2210.17323. URL: http://arxiv.org/abs/2210.17323 (visited on 01/08/2025).

[20] Elias Frantar et al. "OPTQ: Accurate Quantization for Generative Pre-trained Transformers". en. In: URL: https://openreview.net/forum?id=tcbBPnfwxS (visited on 01/24/2025).

[21] Elias Frantar and Dan Alistarh. “Optimal Brain Compression: A Framework for Accurate Post-Training Quantization and Pruning”. en. In: (). URL: https://proceedings.neurips.cc/paper_files/paper/2022/hash/1caf09c9f4e6b0150b06a07e77f271Abstract-Conference.html (visited on 01/08/2025).

[22] Hanlin Tang et al. *EasyQuant: An Efficient Data-free Quantization Algorithm for LLMs*. arXiv:2403.02775 [cs]. Mar. 2024. DOI: 10.48550/arXiv.2403.02775. URL: http://arxiv.org/abs/2403.02775 (visited on 01/08/2025).

[23] Zhewei Yao et al. “ZeroQuant: Efficient and Affordable Post-Training Quantization for Large-Scale Transformers”. en. In: (). URL: https://proceedings.neurips.cc/paper_files/paper/2022/hash/adf7fa39d65e2983d724ff7da57f00ac-Abstract-Conference.html (visited on 01/08/2025).

[24] Wei Huang et al. *BiLLM: Pushing the Limit of Post-Training Quantization for LLMs*. arXiv:2402.04291 [cs]. May 2024. DOI: 10.48550/arXiv.2402.04291. URL: http://arxiv.org/abs/2402.04291 (visited on 01/08/2025).

[25] Chee-Yong Chan and Yannis E. Ioannidis. *Bitmap index design and evaluation*. Seattle, Washington, USA, 1998. DOI: 10.1145/276304.276336. URL: https://doi.org/10.1145/276304.276336.

[26] Zefan Li et al. “ICCV 2017 Open Access Repository”. In: URL: https://openaccess.thecvf.com/content_iccv_2017/html/Li_Performance_Guaranteed_Network_ICCV_2017_paper.html (visited on 01/08/2025).

[27] Wei Huang et al. *An empirical study of LLaMA3 quantization: from LLMs to MLLMs*. Dec. 2024. DOI: 10.1007/s44267-024-00070-x. URL: https://doi.org/10.1007/s44267-024-00070-x.

[28] Saleh Ashkboos et al. *QuaRot: Outlier-Free 4-Bit Inference in Rotated LLMs*. arXiv:2404.00456 [cs]. Oct. 2024. DOI: 10.48550/arXiv.2404.00456. URL: http://arxiv.org/abs/2404.00456 (visited on 01/08/2025).

[29] Guangxuan Xiao et al. *SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models*. Ed. by Andreas Krause et al. July 2023. URL: https://proceedings.mlr.press/v202/xiao23c.html.

[30] Tim Dettmers et al. *LLM.int8(): 8-bit Matrix Multiplication for Transformers at Scale*. arXiv:2208.07339 [cs]. Nov. 2022. DOI: 10.48550/arXiv.2208.07339. URL: http://arxiv.org/abs/2208.07339 (visited on 01/14/2025).

[31] Aohan Zeng et al. *GLM-130B: An Open Bilingual Pre-trained Model*. arXiv:2210.02414 [cs]. Oct. 2023. DOI: 10.48550/arXiv.2210.02414. URL: http://arxiv.org/abs/2210.02414 (visited on 01/08/2025).

[32] Susan Zhang et al. *OPT: Open Pre-trained Transformer Language Models*. arXiv:2205.01068 [cs]. June 2022. DOI: 10.48550/arXiv.2205.01068. URL: http://arxiv.org/abs/2205.01068 (visited on 01/08/2025).

[33] Hugo Touvron et al. *LLaMA: Open and Efficient Foundation Language Models*. arXiv:2302.13971 [cs]. Feb. 2023. DOI: 10.48550/arXiv.2302.13971. URL: http://arxiv.org/abs/2302.13971 (visited on 01/08/2025).

[34] Fabio Petroni et al. *Language Models as Knowledge Bases?* arXiv:1909.01066 [cs]. Sept. 2019. DOI: 10.48550/arXiv.1909.01066. URL: http://arxiv.org/abs/1909.01066 (visited on 01/08/2025).

[35] Patrick Lewis et al. "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks". In: *Advances in Neural Information Processing Systems*. Vol. 33. Curran Associates, Inc., 2020, pp. 9459–9474. URL: https://proceedings.neurips.cc/paper/2020/hash/6b493230205f780e1bc26945df7481e5-Abstract.html (visited on 01/08/2025).

[36] Vladimir Karpukhin et al. *Dense Passage Retrieval for Open-Domain Question Answering*. arXiv:2004.04906 [cs]. Sept. 2020. DOI: 10.48550/arXiv.2004.04906. URL: http://arxiv.org/abs/2004.04906 (visited on 01/08/2025).

[37] Mike Lewis et al. *BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension*. arXiv:1910.13461 [cs] version: 1. Oct. 2019. DOI: 10.48550/arXiv.1910.13461. URL: http://arxiv.org/abs/1910.13461 (visited on 01/08/2025).

[38] Fabing Duan, François Chapeau-Blondeau, and Derek Abbott. "Optimized injection of noise in activation functions to improve generalization of neural networks". In: *Chaos, Solitons & Fractals* 178 (Jan. 2024), p. 114363. ISSN: 0960-0779. DOI: 10.1016/j.chaos.2023.114363. URL: https://www.sciencedirect.com/science/article/pii/S0960077923012651 (visited on 01/08/2025).

[39] Patrick Lewis et al. "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks". In: *Advances in Neural Information Processing Systems*. Vol. 33. Curran Associates, Inc., 2020, pp. 9459–9474. URL: https://proceedings.neurips.cc/paper/2020/hash/6b493230205f780e1bc26945df7481e5-Abstract.html (visited on 01/08/2025).

[40] Luyu Gao et al. *Precise Zero-Shot Dense Retrieval without Relevance Labels*. arXiv:2212.10496 [cs]. Dec. 2022. DOI: 10.48550/arXiv.2212.10496. URL: http://arxiv.org/abs/2212.10496 (visited on 01/08/2025).

[41] Luyu Gao and Jamie Callan. *Unsupervised Corpus Aware Language Model Pre-training for Dense Passage Retrieval*. arXiv:2108.05540 [cs]. Aug. 2021. DOI: 10.48550/arXiv.2108.05540. URL: http://arxiv.org/abs/2108.05540 (visited on 01/08/2025).

[42] Long Ouyang et al. "Training language models to follow instructions with human feedback". en. In: (). URL: https://proceedings.neurips.cc/paper_files/paper/2022/hash/b1efde53be364a73914f58805a001731-Abstract-Conference.html (visited on 01/08/2025).

[43] Brian J. Chan et al. *Don't Do RAG: When Cache-Augmented Generation is All You Need for Knowledge Tasks*. arXiv:2412.15605 [cs]. Dec. 2024. DOI: 10.48550/arXiv.2412.15605. URL: http://arxiv.org/abs/2412.15605 (visited on 01/15/2025).

[44] Chao Jin et al. *RAGCache: Efficient Knowledge Caching for Retrieval-Augmented Generation*. arXiv:2404.12457 [cs]. Apr. 2024. DOI: 10.48550/arXiv.2404.12457. URL: http://arxiv.org/abs/2404.12457 (visited on 01/15/2025).

[45] Zhuowan Li et al. "Retrieval Augmented Generation or Long-Context LLMs? A Comprehensive Study and Hybrid Approach". In: *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: Industry Track*. Ed. by Franck Dernoncourt, Daniel Preoţiuc-Pietro, and Anastasia Shimorina. Miami, Florida, US: Association for Computational Linguistics, Nov. 2024, pp. 881–893. DOI: 10.18653/v1/2024.emnlp-industry.66. URL: https://aclanthology.org/2024.emnlp-industry.66/ (visited on 01/15/2025).

[46] *Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context*. arXiv:2403.05530 [cs]. Dec. 2024. DOI: 10.48550/arXiv.2403.05530. URL: http://arxiv.org/abs/2403.05530.

[47] Woosuk Kwon et al. "Efficient Memory Management for Large Language Model Serving with PagedAttention". In: *Proceedings of the 29th Symposium on Operating Systems Principles*. SOSP '23. New York, NY, USA: Association for Computing Machinery, Oct. 2023, pp. 611–626. ISBN: 979-8-4007-0229-7. DOI: 10.1145/3600006.3613165. URL: https://dl.acm.org/doi/10.1145/3600006.3613165 (visited on 01/16/2025).

[48] Conglong Li et al. "Improving Approximate Nearest Neighbor Search through Learned Adaptive Early Termination". In: *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. SIGMOD '20. New York, NY, USA: Association for Computing Machinery, May 2020, pp. 2539–2554. ISBN: 978-1-4503-6735-6. DOI: 10.1145/3318464.3380600. URL: https://dl.acm.org/doi/10.1145/3318464.3380600 (visited on 01/19/2025).

[49] Gautier Izacard et al. *Unsupervised Dense Information Retrieval with Contrastive Learning*. arXiv:2112.09118 [cs]. Aug. 2022. DOI: 10.48550/arXiv.2112.09118. URL: http://arxiv.org/abs/2112.09118 (visited on 01/20/2025).

[50] Kenton Lee, Ming-Wei Chang, and Kristina Toutanova. *Latent Retrieval for Weakly Supervised Open Domain Question Answering*. arXiv:1906.00300 [cs]. June 2019. DOI: 10.48550/arXiv.1906.00300. URL: http://arxiv.org/abs/1906.00300 (visited on 01/20/2025).

[51] Kaiming He et al. *Momentum Contrast for Unsupervised Visual Representation Learning*. arXiv:1911.05722 [cs]. Mar. 2020. DOI: 10.48550/arXiv.1911.05722. URL: http://arxiv.org/abs/1911.05722 (visited on 01/21/2025).

[52] Yile Wang et al. "Self-Knowledge Guided Retrieval Augmentation for Large Language Models". In: *Findings of the Association for Computational Linguistics: EMNLP 2023*. Ed. by Houda Bouamor, Juan Pino, and Kalika Bali. Singapore: Association for Computational Linguistics, Dec. 2023, pp. 10303–10315. DOI: 10.18653/v1/2023.findings-emnlp.691. URL: https://aclanthology.org/2023.findings-emnlp.691/ (visited on 05/05/2025).

[53] Evelyn Fix and J. L. Hodges. "Discriminatory Analysis. Nonparametric Discrimination: Consistency Properties". In: *International Statistical Review / Revue Internationale de Statistique* 57.3 (1989), pp. 238–247. ISSN: 03067734, 17515823. URL: http://www.jstor.org/stable/1403797 (visited on 06/17/2025).

[54] LLM Explorer. *LLM Explorer: A Curated Large Language Model Directory. LLM List. 41870 Open-Source Language Models.* en. URL: https://llm.extractum.io (visited on 01/24/2025).

[55] Dan Hendrycks et al. *Measuring Massive Multitask Language Understanding*. arXiv:2009.03300 [cs]. Jan. 2021. DOI: 10.48550/arXiv.2009.03300. URL: http://arxiv.org/abs/2009.03300 (visited on 01/08/2025).

[56] Yubo Wang et al. *MMLU-Pro: A More Robust and Challenging Multi-Task Language Understanding Benchmark*. arXiv:2406.01574 [cs]. Nov. 2024. DOI: 10.48550/arXiv.2406.01574. URL: http://arxiv.org/abs/2406.01574 (visited on 01/08/2025).

[57] David Rein et al. *GPQA: A Graduate-Level Google-Proof Q&A Benchmark*. arXiv:2311.12022 [cs]. Nov. 2023. DOI: 10.48550/arXiv.2311.12022. URL: http://arxiv.org/abs/2311.12022 (visited on 01/08/2025).

[58] Zayne Sprague et al. *MuSR: Testing the Limits of Chain-of-thought with Multistep Soft Reasoning*. arXiv:2310.16049 [cs]. Mar. 2024. DOI: 10.48550/arXiv.2310.16049. URL: http://arxiv.org/abs/2310.16049 (visited on 01/08/2025).

[59] Jeffrey Zhou et al. *Instruction-Following Evaluation for Large Language Models*. arXiv:2311.07911 [cs]. Nov. 2023. DOI: 10.48550/arXiv.2311.07911. URL: http://arxiv.org/abs/2311.07911 (visited on 01/08/2025).

[60] Peter Clark et al. *Think you have Solved Question Answering? Try ARC, the AI2 Reasoning Challenge*. arXiv:1803.05457 [cs]. Mar. 2018. DOI: 10.48550/arXiv.1803.05457. URL: http://arxiv.org/abs/1803.05457 (visited on 01/24/2025).

[61] Rowan Zellers et al. *HellaSwag: Can a Machine Really Finish Your Sentence?* arXiv:1905.07830 [cs]. May 2019. DOI: 10.48550/arXiv.1905.07830. URL: http://arxiv.org/abs/1905.07830 (visited on 01/08/2025).

[62] Stephanie Lin, Jacob Hilton, and Owain Evans. *TruthfulQA: Measuring How Models Mimic Human Falsehoods*. arXiv:2109.07958 [cs]. May 2022. DOI: 10.48550/arXiv.2109.07958. URL: http://arxiv.org/abs/2109.07958 (visited on 01/08/2025).

[63] Karl Cobbe et al. *Training Verifiers to Solve Math Word Problems*. arXiv:2110.14168 [cs]. Nov. 2021. DOI: 10.48550/arXiv.2110.14168. URL: http://arxiv.org/abs/2110.14168 (visited on 01/08/2025).

[64] Dan Hendrycks et al. *Measuring Mathematical Problem Solving With the MATH Dataset*. arXiv:2103.03874 [cs]. Nov. 2021. DOI: 10.48550/arXiv.2103.03874. URL: http://arxiv.org/abs/2103.03874 (visited on 01/08/2025).

[65] Kunlun Zhu et al. *RAGEval: Scenario Specific RAG Evaluation Dataset Generation Framework*. arXiv:2408.01262 [cs]. Mar. 2025. DOI: 10.48550/arXiv.2408.01262. URL: http://arxiv.org/abs/2408.01262 (visited on 03/28/2025).

[66] Zhilin Yang et al. *HotpotQA: A Dataset for Diverse, Explainable Multi-hop Question Answering*. arXiv:1809.09600 [cs]. Sept. 2018. DOI: 10.48550/arXiv.1809.09600. URL: http://arxiv.org/abs/1809.09600 (visited on 07/29/2025).

[67] Rolf Jagerman et al. *Query Expansion by Prompting Large Language Models*. arXiv:2305.03653 [cs]. May 2023. DOI: 10.48550/arXiv.2305.03653. URL: http://arxiv.org/abs/2305.03653 (visited on 01/22/2025).

[68] Lihu Chen and Gaël Varoquaux. *What is the Role of Small Models in the LLM Era: A Survey*. arXiv:2409.06857 [cs]. Dec. 2024. DOI: 10.48550/arXiv.2409.06857. URL: http://arxiv.org/abs/2409.06857 (visited on 01/08/2025).

[69] DeepSeek-AI et al. *DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning*. en. arXiv:2501.12948 [cs]. Jan. 2025. DOI: 10.48550/arXiv.2501.12948. URL: http://arxiv.org/abs/2501.12948 (visited on 06/20/2025).

[70] Mirac Suzgun et al. *Challenging BIG-Bench Tasks and Whether Chain-of-Thought Can Solve Them*. arXiv:2210.09261 [cs]. Oct. 2022. DOI: 10.48550/arXiv.2210.09261. URL: http://arxiv.org/abs/2210.09261 (visited on 01/08/2025).

[71] Xinyi Wu et al. *On the Emergence of Position Bias in Transformers*. arXiv:2502.01951 [cs]. June 2025. DOI: 10.48550/arXiv.2502.01951. URL: http://arxiv.org/abs/2502.01951 (visited on 06/20/2025).

[72] *docs.nvidia.com/deploy/nvidia-smi/index.html*. URL: https://docs.nvidia.com/deploy/nvidia-smi/index.html (visited on 07/02/2025).

[73] *NVML Device Queries*. en-us. cppModule. URL: https://docs.nvidia.com/deploy/nvml-api/group__nvmlDeviceQueries.html#group__nvmlDeviceQuerie 1g7ef7dff0ff14238d08a19ad7fb23fc87 (visited on 07/02/2025).

[74] Alireza Salemi and Hamed Zamani. "Evaluating Retrieval Quality in Retrieval-Augmented Generation". In: *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '24. New York, NY, USA: Association for Computing Machinery, July 2024, pp. 2395–2400. ISBN: 979-8-4007-0431-4. DOI: 10.1145/3626772.3657957. URL: https://dl.acm.org/doi/10.1145/3626772.3657957 (visited on 01/23/2025).