UNIVERSITY of VIRGINIA

YaRN: Efficient Context Window Extension of Large Language Models

Presented by:
Anisha Patrikar (gjq2yf) and Yagnik Panguluri (yye7pm)

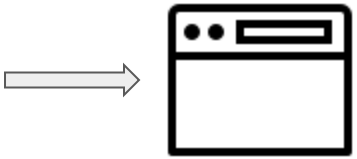
Yagnik Panguluri (yye7pm)

Presentation Outline

- Introduction and Motivation
- Background: Position Embeddings in Transformer
- Prior Solutions and Challenges
- YaRN Overview
- Technical Insight: NTK-aware and NTK-by-parts
- Dynamic Scaling and Attention Temperature
- Training Setup
- Evaluation Criteria
- Key Results: Perplexity and Long Sequence Modeling
- Conclusion and Takeaways

Motivation and Introduction

Context Window Limitations



Extend?

Challenges

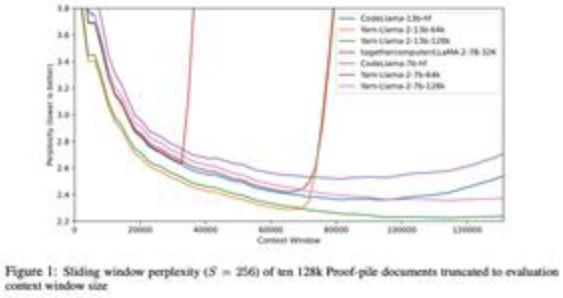


Figure 1: Sliding window perplexity ($S = 256$) of ten 128k Proof-pile documents truncated to evaluation context window size

YaRN

Motivation and Introduction

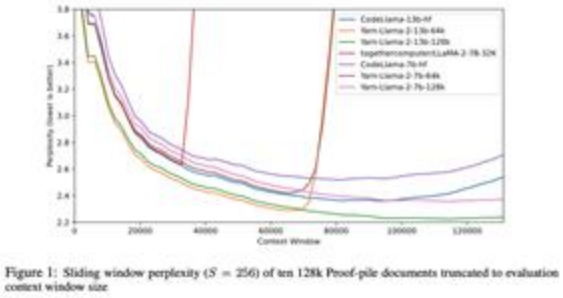


Figure 1: Sliding window perplexity ($S = 256$) of ten 128k Proof-pile documents truncated to evaluation context window size

Context Window Limitations

Extend?

Challenges

YaRN

- Large Language Models (LLMs) like GPT, LLaMa are used for many NLP tasks
- Their performance heavily depends on context window size
- Pre-trained LLMs can only handle fixed, relatively short context lengths (2k - 4k tokens)

Motivation and Introduction

Context Window Limitations

Extend?

Challenges

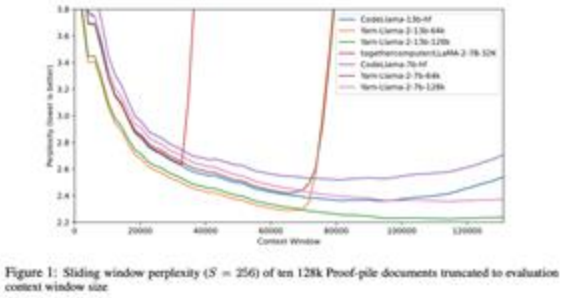


Figure 1: Sliding window perplexity ($S = 256$) of ten 128k Proof-pile documents truncated to evaluation context window size

YaRN

Longer contexts enable:

- Better long-term dependencies (e.g. long documents, conversations)
- Improved in-context learning abilities
- Enhanced tasks like summarization, retrieval, and reasoning

Motivation and Introduction

Context Window Limitations

Extend?

Challenges

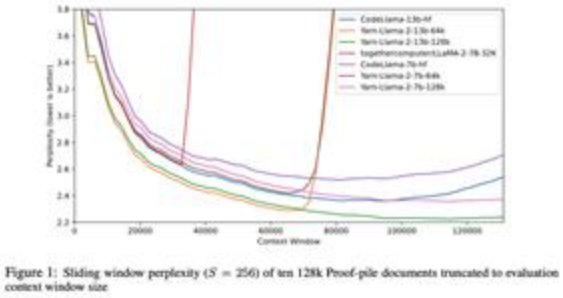


Figure 1: Sliding window perplexity ($S = 256$) of ten 128k Proof-pile documents truncated to evaluation context window size

YaRN

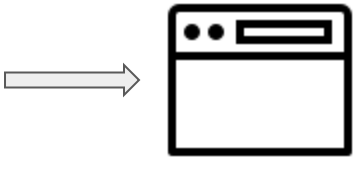
Position Encoding Bottleneck - Existing position encodings (like RoPE) don't generalize beyond training window

Existing Solutions:

- Significant compute (large-scale fine-tuning)
- Performance trade-offs at long lengths

Motivation and Introduction

Context Window Limitations



Extend?

Challenges

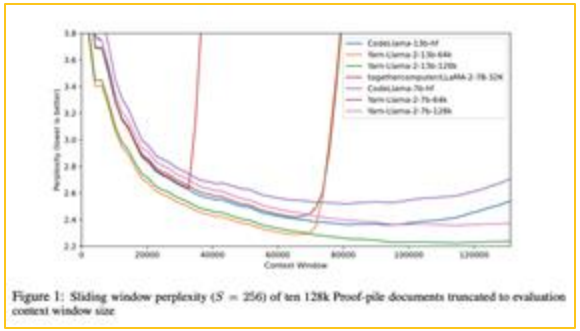


Figure 1: Sliding window perplexity ($S = 256$) of ten 128k Proof-pile documents truncated to evaluation context window size

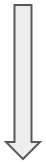
YaRN

- **Goal:** Efficiently extend the context window with minimal fine-tuning cost and no degradation
- Leverage and improve existing RoPE techniques to achieve:
  - Lower training steps
  - Generalizations to much longer contexts (up to 128k tokens)

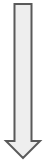
Background: Position Embeddings in Transformers

Transformers lack inherent sense of word order. Position embeddings encode token order information

Absolute Sinusoidal Encoding (Original Transformer)



Learnable Absolute Position Encoding



Relative Positional Encodings

Fixed, predefined sinusoidal patterns

Trainable vectors assigned to each position

Allow model to focus on relative distances between tokens

Popular methods:

- T5 Relative Bias
- ALiBi
- RoPE

Rotary Position Embeddings (RoPE)

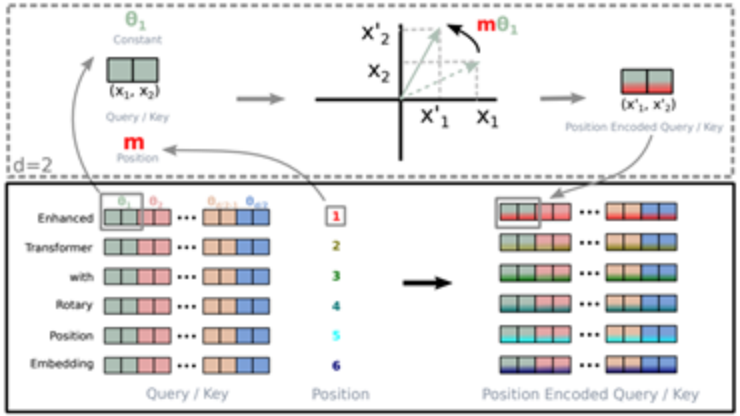


Figure 1: Implementation of Rotary Position Embedding(RoPE).

Rotary Position Embeddings

- Encodes relative positional information via complex rotations
- Introduced in RoFormer, widely used in models like LLaMA, GPT-NeoX, and PaLM

Methodology

- Converts token embeddings into complex space
- Applies a rotation based on the token's position

To convert embeddings $\mathbf{x}_m, \mathbf{x}_n$ into query and key vectors, we are first given $\mathbb{R}$-linear operators

$$\mathbf{W}_q, \mathbf{W}_k : \mathbb{R}^{|D|} \to \mathbb{R}^{|D|}.$$

In complex coordinates, the functions f_q, f_k are given by

$$f_q(\mathbf{x}_m, m) = e^{im\theta}\mathbf{W}_q\mathbf{x}_m, \ f_k(\mathbf{x}_n, n) = e^{in\theta}\mathbf{W}_k\mathbf{x}_n, \tag{5}$$

where $\theta = \mathrm{diag}(\theta_1, \cdots, \theta_{|D|/2})$ is the diagonal matrix with $\theta_d = b^{-2d/|D|}$ and $b = 10000$. This way, RoPE associates each (complex-valued) hidden neuron with a separate frequency θ_d. The benefit of doing so is that the dot product between the query vector and the key vector only depends on the relative distance $m - n$ as follows

$$\begin{aligned}
&\langle f_q(\mathbf{x}_m, m), f_k(\mathbf{x}_n, n)\rangle_{\mathbb{R}} && (6)\\
=&\mathrm{Re}(\langle f_q(\mathbf{x}_m, m), f_k(\mathbf{x}_n, n)\rangle_{\mathbb{C}}) && (7)\\
=&\mathrm{Re}(\mathbf{x}_m^*\mathbf{W}_q^*\mathbf{W}_k\mathbf{x}_n e^{i\theta(m-n)}) && (8)\\
=&g(\mathbf{x}_m, \mathbf{x}_n, m-n). && (9)
\end{aligned}$$

In real coordinates, the RoPE can be written using the following function

$$f_{\mathbf{W}}(\mathbf{x}_m, m, \theta_d) = \begin{pmatrix} \cos m\theta_1 & -\sin m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ \sin m\theta_1 & \cos m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cos m\theta_2 & -\sin m\theta_2 & \cdots & 0 & 0 \\ 0 & 0 & \sin m\theta_2 & \cos m\theta_2 & \cdots & 0 & 0 \end{pmatrix} \mathbf{W}\mathbf{x}_m,$$

Prior Solutions and Challenges

RoPE struggles to generalize beyond its trained context length. Extending the context requires rethinking how positional information is encoded.

Position Interpolation (PI)

NTK-aware Interpolation

Dynamic NTK

ReRoPE, LM-Infinite

Core challenge: high compute cost, complex fine-tuning, and may degrade performance on short or unseen sequences

extrapolation does not perform well on sequences $w_1, \cdots, w_L$ with L larger than the pre-trained limit, they discovered that interpolating the position indicies within the pre-trained limit works well with the help of a small amount of fine-tuning. Specifically, given a pre-trained language model with RoPE, they modify the RoPE by

$$f'_{\mathbf{W}}\left(\mathbf{x}_m, m, \theta_d\right) = f_{\mathbf{W}}\left(\mathbf{x}_m, \frac{mL}{L'}, \theta_d\right), \tag{10}$$

where $L' > L$ is a new context window beyond the pre-trained limit. With the original pre-trained model plus the modified RoPE formula, they fine-tuned the language model further on several orders of magnitude fewer tokens (a few billion in Chen et al. [9]) and successfully acheived context window extension.

YaRN Overview

YaRN = Yet another RoPE extensioN method

	Prior Methods	YaRN
Tokens Needed	High (billions)	Low (~0.1% of pretrain data
Training Steps	High	Reduced (2.5x fewer)
Max Context	Limited (32k-100k)	Up to 128k

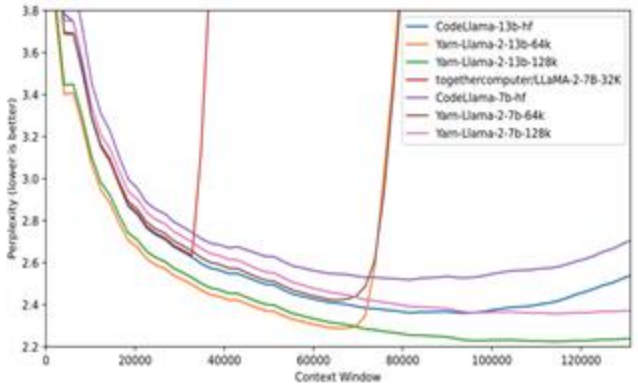


Figure 1: Sliding window perplexity ($S = 256$) of ten 128k Proof-pile documents truncated to evaluation context window size

Improved Interpolation Techniques

Dynamic Scaling

Attention Temperature Adjustment

Technical Insight: NTK-aware and NTK-by-parts

NTK-aware

- Inspired by Neural Tangent Kernel (NTK) theory
- Adjusts scaling non-uniformly
  - High frequency dimensions scaled less
  - Preserves fine-grained token relationships

NTK-by-parts

- Observation: wavelength of RoPE dimension differs
- Idea:
  - High Frequency -> no interpolation
  - Low Frequency -> Interpolated proportionally
  - Mid-range Frequency -> Smooth ramp function between the two

Anisha Patrikar (gjq2yf)

Presentation Outline

- Introduction and Motivation
- Background: Position Embeddings in Transformer
- Prior Solutions and Challenges
- YaRN Overview
- Technical Insight: NTK-aware and NTK-by-parts
- Dynamic Scaling and Attention Temperature
- Training Setup
- Evaluation Criteria
- Key Results: Perplexity and Long Sequence Modeling
- Conclusion and Takeaways

Dynamic Scaling

Adapts interpolation dynamically based on sequence length

Prevents abrupt performance drop at longer contexts

Updates scaling factor dynamically at inference

Formula for Updating Scaling Factor:

$$s = \max(1, \frac{l'}{L})$$

Scaling factor for interpolation

Actual sequence length

Pretrained context limit

Attention Temperature

Prevents loss of attention weight sharpness

Adjusts softmax to avoid extreme probability shifts

$$\mathbf{q}_m = f_q(\mathbf{x}_m, m) \in \mathbb{R}^{|D|}, \ \mathbf{k}_n = f_k(\mathbf{x}_n, n) \in \mathbb{R}^{|D|}.$$

$$\text{softmax}\left(\frac{q_m^T k_n}{\sqrt{|D|}}\right) \Longrightarrow \text{softmax}\left(\frac{q_m^T k_n}{t\sqrt{|D|}}\right)$$

Training Setup

Fine-tuning Process:

- Used LLaMA 2 (7B & 13B) with YaRN Interpolation
- Training data: PG19 Dataset
- **10× fewer tokens, 2.5× fewer training steps than prior methods**

Training Efficiency:

- YaRN achieves context extension with minimal fine-tuning

Parameter	Value
Learning rate	2×10^{-5}
Batch size	64
Steps (s=16)	400
Steps (s=32)	200

Evaluating YaRN's Performance

Perplexity on Long Sequences

Passkey Retrieval Accuracy

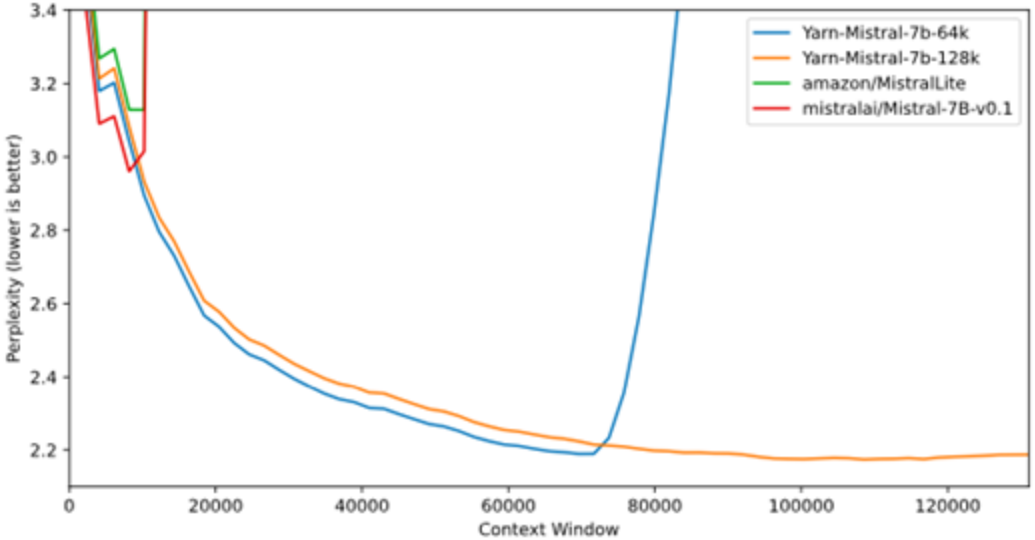
LLM Standard Benchmarks

Key Results: Passkey Accuracy

- YaRN 7B and 13B achieve accuracy of 99.4% at 128K tokens
- NTK-based models drop to 94.3% at 112K tokens

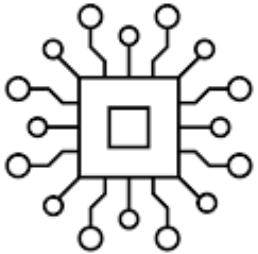
Model Size	Model Name	Scaling Factor (s)	Context Window	Training Data Context	Extension Method	Passkey Context	Passkey Accuracy
7B	Together	4	32k	32k	PI	32k	100%
7B	Code Llama	88.6	100k	16k	NTK	112k	94.3%
7B	YaRN	16	64k	64k	YaRN	64k	96.3%
7B	YaRN	32	128k	64k	YaRN	128k	99.4%
13B	Code Llama	88.6	100k	16k	NTK	128k	99.4%
13B	YaRN	16	64k	64k	YaRN	64k	97.5%
13B	YaRN	32	128k	64k	YaRN	128k	99.4%

Key Results: Perplexity



Model Size	Model Name	Context Window	Extension Method	Evaluation Context Window Size 8192	32768	65536	98304	131072
7B	Together	32k	PI	**3.50**	**2.64**	$> 10^2$	$> 10^3$	$> 10^4$
7B	Code Llama	100k	NTK	3.71	2.74	2.55	2.54	2.71
7B	YaRN ($s = 16$)	64k	YaRN	3.51	2.65	**2.42**	$> 10^1$	$> 10^1$
7B	YaRN ($s = 32$)	128k	YaRN	3.56	2.70	2.45	**2.36**	**2.37**
13B	Code Llama	100k	NTK	3.54	2.63	2.41	2.37	2.54
13B	YaRN ($s = 16$)	64k	YaRN	**3.25**	**2.50**	**2.29**	$> 10^1$	$> 10^1$
13B	YaRN ($s = 32$)	128k	YaRN	3.29	2.53	2.31	**2.23**	**2.24**

Conclusion and Key Takeaways



YaRN enables efficient context window extension

Requires significantly less training data

Compatible with existing transformer architectures

Long Context vs. RAG for LLMs: An Evaluation and Revisits

Presented by:
Aditya Kakkar (zjq5mr) and Aryan Sawhney (ryd2fx)

Aditya Kakkar (zjq5mr)

Presentation Outline

- Introduction (LC vs RAG)
- Background & Motivation
  - Related Work
- Question Filtering and Expansion
  - Evaluation Methodology

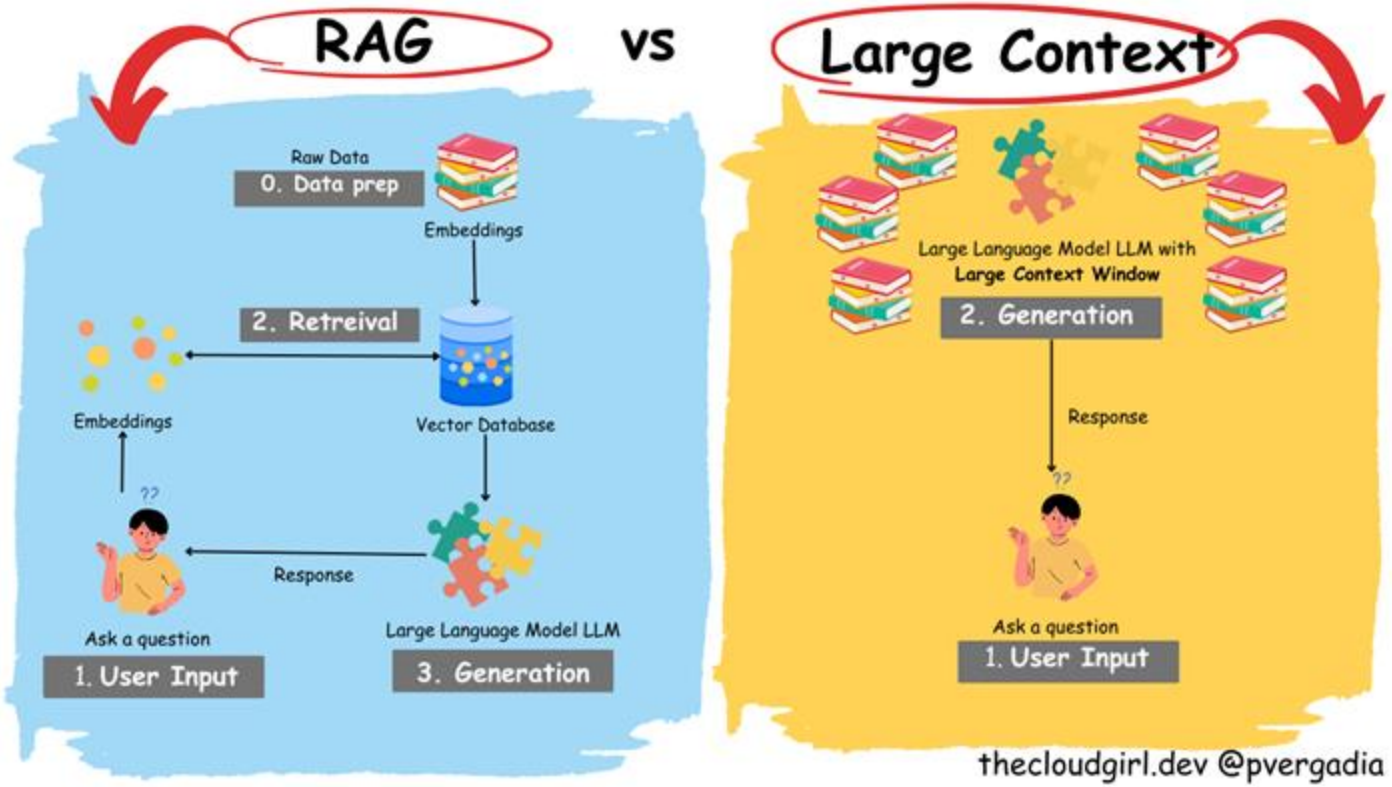
Presentation Outline

- Introduction (LC vs RAG)
- Background & Motivation
- Related Work
- Question Filtering and Expansion
- Evaluation Methodology
- Insights from Dataset and Evaluation

Introduction

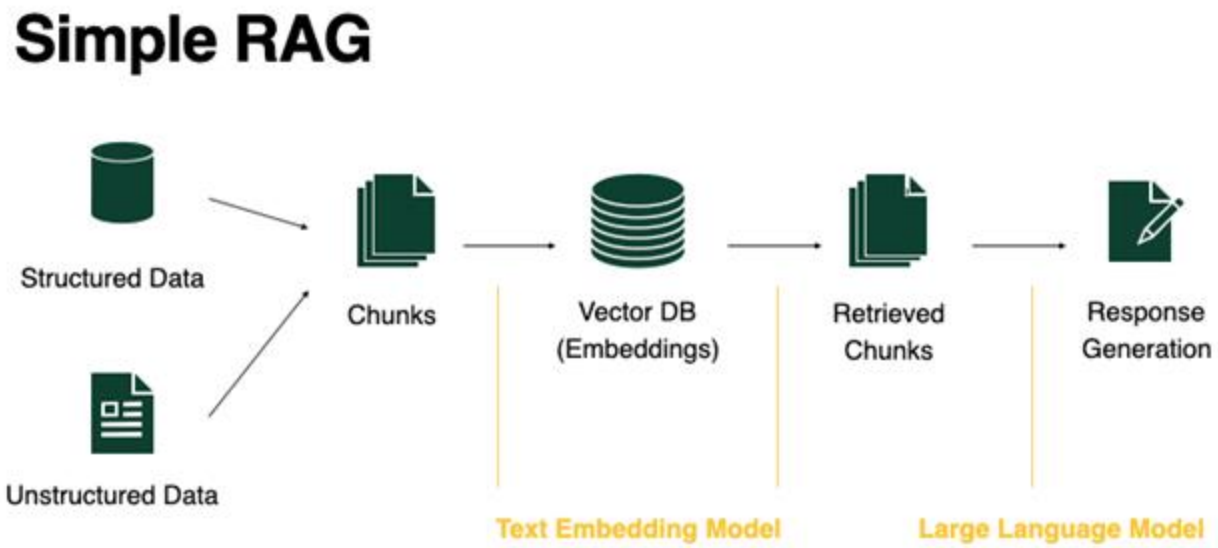
Long Context vs. RAG for LLMs

Background


RAG
vs
Large Context
Raw Data
0. Data prep
Embeddings
2. Retreival
Embeddings
Vector Database
??
Response
Ask a question
1. User Input
Large Language Model LLM
3. Generation
Large Language Model LLM with
Large Context Window
2. Generation
Response
??
Ask a question
1. User Input
thecloudgirl.dev @pvergadia

Motivation

RAG


Simple RAG
Structured Data
Unstructured Data
Chunks
Vector DB
(Embeddings)
Retrieved
Chunks
Response
Generation
Text Embedding Model
Large Language Model

RAG

Retrieval Type	How It Works	Best Use Case
Chunk-Based Retrieval	Splits document into smaller sections and retrieves relevant ones	Best for fact-based, structured text (e.g., research papers, news)
Index-Based Retrieval	Pre-processes documents into structured indices (e.g., tree or knowledge graphs) for efficient lookups	Best for large, structured databases or knowledge graphs
Summarization-Based Retrieval	Generates multi-level summaries instead of retrieving raw text chunks	Best for long documents, multi-step reasoning, and broad queries

Related Work

Paper	Type	Findings
LongBench (B) (Bai et al., 2024a)	•	Retrieval helps 4k model, but not 16k/32k models.
	+	Models benefit from continuous training on long contexts.
	+	Splitting context into shorter and more chunks is better.
Ret-LC LLM (R) (Xu et al., 2024b)	⋆	LC is better for multi-hop benchmarks than 4k RAG.
	○	RAG improves on 70B/43B models on all context lengths.
	+	For LC model, best results are obtained from top-5 or top-10.
LongRAG (L) (Jiang et al., 2024b)	○	Retrieval benefits from long retrieval units.
ChatQA2 (C) (Xu et al., 2024a)	✩	For sequence lengths up to 32K, RAG outperforms LC.
	○	From 3K to 24K, greater context window benefits RAG.
Self-ROUTE (S) (Li et al., 2024)	⋆	LC consistently outperforms RAG, but RAG has lower cost.
OP-RAG (O) (Yu et al., 2024)	✩	Efficient retrieval can outperform brute-force LC.
	+	Too many chunks in RAG harms performance.
	+	Preserving the original order is better than ordering by score.
LC LLM-RAG (M) (Jin et al., 2024)	•	Retrieve more passages first improves performance then drops.
	+	Ordering higher score information to front and back helps.
LC RAG Performance (P) (Leng et al., 2024)	○	Most close models' RAG improves up to 100k tokens.
	•	Most open models' RAG peak at 16k-32k then performance drops.
LongBench v2 (V) (Bai et al., 2024b)	✩	GPT-4o performs better at 128k without RAG.
	○	GPT-4o performance keeps increasing to 128k RAG context.
	•	Qwen2.5 & GLM-4-Plus drop with >32k RAG contexts.

- **LC vs RAG**: RAG improves performance for models like GPT-4o with long context windows (up to 128K tokens), but LC outperforms RAG in multi-hop benchmarks, depending on model size and retrieval strategy (Xu et al., 2024b, Xu et al., 2024a).

- **Efficiency**: Combining LC and RAG can be beneficial with efficient retrieval strategies, as demonstrated by Self-ROUTE and OP-RAG, which also consider cost reduction (Li et al., 2024, Yu et al., 2024).

Question Filtering and Expansion

Dataset	# Questions	# Kept Q	% Kept Q
Coursera	172	54	32
NQ	1,109	373	34
NovelQA	2,283	869	38
2WikiMHQA	2,300	1,036	45
HotpotQA	2,200	1,113	51
MuSiQue	2,200	1,663	78
MultiFieldQA	150	121	81
NarrativeQA	2,211	1,880	85
QASPER	2,718	2,674	98
QuALTY	2,725	2,725	100
TOEFL-QA	962	962	100
MultiDoc2Dial	158	158	100
Total	**19,188**	**13,628**	**71**

- Curates dataset from 12 QA sources to compare LC and RAG.
- Filters questions to need external context, using GPT-4o.
- Expands dataset to 20,000 questions for better analysis.

Evaluation Methodology

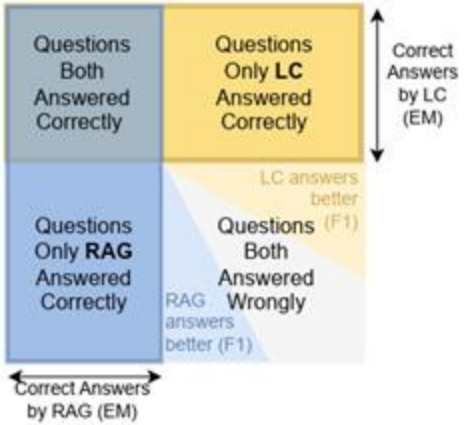


Figure 2: Evaluation Matrix for In-depth Analysis.

- **Evaluation Phases:** Compares RAG and LC in three phases
  - Tests **five retrievers** and picks the best(RAPTOR)
  - Compares RAG and LC on a full question set with the **same LLM**
  - Analyzes subsets where each excels.

- **Evaluation Metrics:** Uses win-lose rate with Exact Match and F1 scores; loose evaluation includes EM wins and higher F1 scores for open-ended answers.

Aryan Sawhney (ryd2fx)

Experiment Setup

- **To obtain answers from question dataset the following prompt was used:**
  “From the context: [context], answer the questions briefly with no explanation.” for both retrieval and long context settings.

- **For MCQ questions, the following sentence was added to the prompt**
  “Answer the question with the letters of the correct options (e.g. A, BC, C, ACD, etc.) with- out including text”

Phase 1: Retrievers

Type	Retriever	Correct (%)	RAG Only	RAG Better
Chunk	BM25	319 (20.4)	50	141
	Contriever	315 (20.1)	43	143
	Text-emb-3-small	338 (21.6)	47	151
Index	Tree Index	470 (30.1)	82	234
	Window Parsing	555 (35.5)	91	237
Summarization	RAPTOR	**602 (38.5)**	97	258

Table 5: Comparison of different retrieval methods

- Evaluated retrievers include chunk-based, index-based, and summarization-based methods
- RAPTOR demonstrates superior overall document understanding, particularly for research papers achieving the highest correct answer rate at 38.5%
- Index-based retrievers outperform chunk-based methods
- Chunk-based methods struggle with information dispersed across multiple chunks.
- Index-based retrievers, while not as strong as RAPTOR in overall comprehension, are effective in interpreting dialogues.

Phase 2: Comparing LC and RAG

Dataset	# Questions	LC Correct	RAG Correct	LC Only	RAG Only	LC Better	RAG Better
Coursera	54	26	20	10	4	10	4
2WikiMHQA	1,036	594	431	242	79	265	107
HotpotQA	1,113	876	723	212	59	231	67
MultiFieldQA	121	63	60	14	11	44	21
NQ	373	189	138	75	24	104	35
NarrativeQA	1,880	558	405	276	123	685	281
QASPER	2,674	884	863	517	496	1,011	762
QuALITY	2,725	2,290	2,050	402	162	402	162
TOEFL-QA	962	895	884	26	15	26	15
MuiQue	1,663	821	663	344	186	426	225
MultiDoc2Dial	158	14	38	5	29	65	58
NovelQA	869	466	408	164	106	164	106
Overall	**13,628**	**7676**	**6,683**	**2,287**	**1,294**	**3,433**	**1,843**

Table 4: Performance of LC and RAG across different datasets. We report the number of questions answered correctly by each method, as well as the breakdown of questions where: only LC answers correctly (LC Only), only RAG answers correctly (RAG Only), LC outperforms RAG (LC Better), and RAG outperforms LC (RAG Better).

- Compared LC and RAG on the filtered, full question set across 12 datasets
- Overall, LC answers 56.3% of questions correctly, while RAG answers 49.0% correctly.
- Although LC shows better overall results, nearly 10% of the 13,628 questions can only be answered correctly by RAG, indicating retrievers cannot be simply replaced by long-context LLMs.

Phase 3: In-Depth Analysis

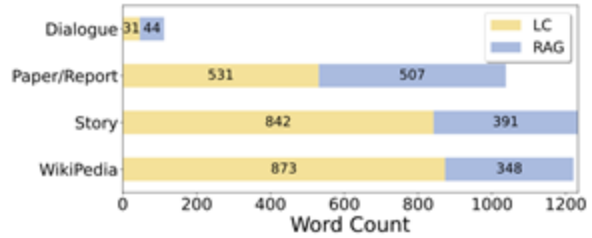


Figure 3: Performance breakdown by knowledge source for LC Only and RAG Only.

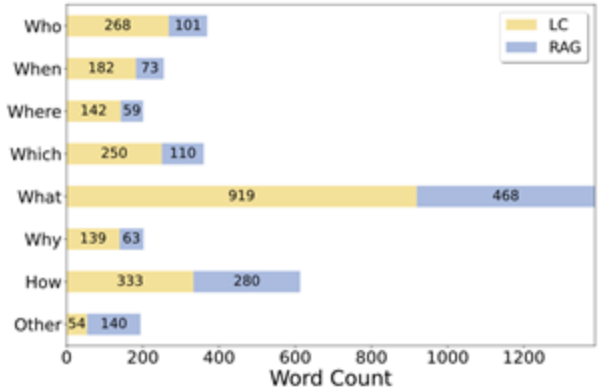


Figure 4: Performance breakdown by question type for LC Only and RAG Only.

- LC and RAG performance varies by knowledge source and question type
- LC excels with noisy, long contexts (e.g., Wikipedia and stories), showing robustness to irrelevant information
- RAG performs better with naturally segmented sources like dialogues, papers, and reports
- LC leads on fact-based questions (e.g., “Who,” “Where,” “Which”), while RAG is comparable on open-ended (“How”) and general yes/no questions
- These findings highlight that each method has unique strengths depending on the scenario.

Word Frequency Visualization

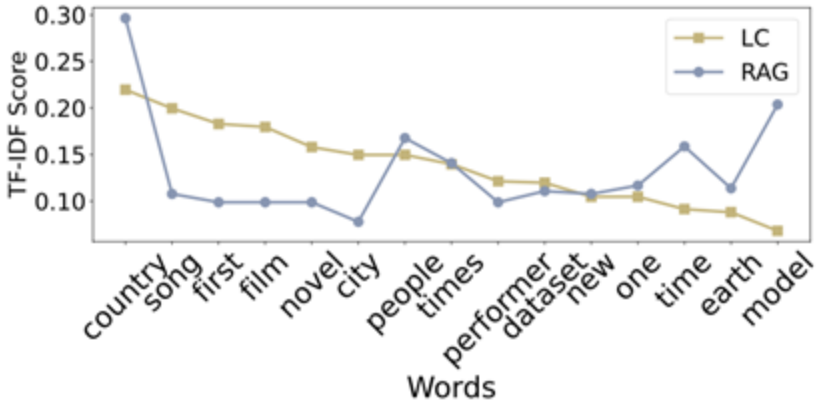


Figure 5: Top 15 Words based on TF-IDF Score for LC Only vs. RAG Only.

- TF-IDF analysis was conducted on questions where LC or RAG exclusively produced correct answers
- The analysis treated all questions from each dataset as a single document, focusing on term frequency after removing stopwords
- LC's top terms include "song," "film," and "novel," indicating strength in narrative topics
- RAG's top terms include "country," "dataset," and "model," suggesting an edge in technical or data-oriented topics

Impact of Generation Model in RAG

Retriever	Model	Correct (%)	RAG Only	RAG Better
BM25	GPT-4o	319 (20.4)	50	141
	GPT-4-Turbo	310 (19.8)	51	152
Tree-Index	GPT-4o	470 (30.1)	82	234
	GPT-4-Turbo	458 (29.3)	81	229
RAPTOR	GPT-4o	**602 (38.5)**	97	258
	GPT-4-Turbo	**589 (37.7)**	99	295

Table 6: Results of using different generation models

- Evaluated the impact of GPT-4o and GPT-4-Turbo on RAG's performance with three retrievers (BM25, Tree Index, RAPTOR)
- Performance is largely consistent across generation models regardless of the retriever
- RAPTOR outperforms the others, with a slight decrease when using GPT-4-Turbo versus GPT-4o
- Differences between GPT-4o and GPT-4-Turbo are marginal, suggesting the choice depends on factors like efficiency or resource availability
- The retrieval method has a larger influence on overall performance than the specific generation model

Case Study

Question: What is the debt-to-GDP ratio of the country where Anthony Upko was formerly involved in the government? **Wrong Answer:** The context does not provide the debt-to-GDP ratio for Nigeria. **Gold:** 11 percent **Relevant Sents:** 1. Nigeria is the world's 20th largest economy ... the debt-to-GDP ratio is only 11 percent. 2. Anthony Ukpo was Minister of Information and Culture, and then Governor of Rivers State, Nigeria.
Question: When is the performer of song Swing Down Sweet Chariot 's birthday? **Wrong Answer:** May 8, 1940 **Gold:** January 8, 1935 **Relevant Sents:** 1. Swing Down Sweet Chariot is a traditional song ... recorded by Elvis Presley. 2. Elvis Aaron Presley (January 8, 1935 – August 16, 1977), also known as ...

Table 7: Examples cases where RAG made mistakes

Question: Do the tweets come from a specific region? **Wrong Answer:** Yes, the tweets come from 16 different countries. **Gold:** No **Relevant Sents:** This helped us narrow down our query space to 16 countries.
Question: Where did Valancourt lose his wealth? **Wrong Answer:** In Gambling. **Gold:** Paris **Relevant Sents:** Returning to her aunt's estate, Emily learns that Valancourt has gone to Paris and lost his wealth.

Table 8: Examples representing common cases where only RAG answers correctly

- LC rarely reports context absence, but its mistakes stem from misinterpreting questions (e.g., confusing "a specific region" with multiple countries or answering "how" instead of "where")
- Both models can locate related text, but their reasoning is affected by how they handle context and question interpretation

- A case study examined frequent errors from LC and RAG by manually reviewing questions each method got wrong
- RAG's most common error is failing to retrieve relevant context, often due to missed sentences or split chunks, leading to refusal or incomplete answers
- RAG also misinterprets partial context, as seen when it linked an incorrect birthday due to overly long, ambiguous text spans

What is Long Context?

- **Definition Variability:** Studies define "long" differently—ChatQA2 uses >32k tokens, LongBench v2 uses >8k, with others using 8k, 16k, or even 128k; no universal standard exists
- **Relative Nature of "Long":** The term is context-dependent; 4k tokens may be long for some models (e.g., BERT-base) but not for others with larger context windows
- **Context Relevance:** "Context" means the situation explaining a question, but long-context datasets may not always prioritize high relevance
- **Dataset Types:**
  - Realistic long texts (novels, research papers) challenge models with cohesive, dense information
  - Synthetic long texts (concatenated segments from sources like Wikipedia) simulate retrieval tasks and may include noise
- **Task Alignment:** Realistic texts suit reading comprehension, while synthetic texts assess factual reasoning and effects like the lost-in-the-middle phenomenon

How to Compare or Combine LC & RAG?

- A clear framework is needed to compare LC and RAG, focusing on three perspectives: context length, context relevance, and experiment design
- **Context Length:** Refers to the maximum tokens a model can process versus the amount of text provided in a dataset; synthetic datasets trade off between increased length and decreased relevance, making it essential to specify if 'long' comes from the model's capacity, dataset design, or both
- **Context Relevance:** Distinguishes between realistic long contexts (highly relevant, cohesive texts) and synthetic ones (often low relevance and resembling RAG pipelines), with biases potentially introduced by chunking that disrupts information continuity
- **Experiment Settings:** Evaluation can compare short-context RAG vs. long-context single input, long-context RAG vs. long-context single input, and explore how RAG performance scales with increasing context length, highlighting trade-offs in performance and computational cost
- These insights suggest that LC and RAG can complement each other in real-world scenarios depending on the data characteristics and question types.

Limitations

- The study is limited to text-based long contexts and does not consider audio, video, or multi-modal contexts
- It focuses on papers that compare RAG with LC, rather than surveying all available retrievers and models
- The experiments use current LC and RAG implementations, and future advancements may alter the comparative outcomes

UNIVERSITY of VIRGINIA

Importing Phantoms: Measuring LLM Package Hallucination Vulnerabilities

Presented by:
Nina Chinnam (fhs9af), Mihika Rao (xsw5kn)

Presentation Outline

- Background & Motivation
- Methodology
- Analysis
- Related Work
- Conclusion

Presentation Outline

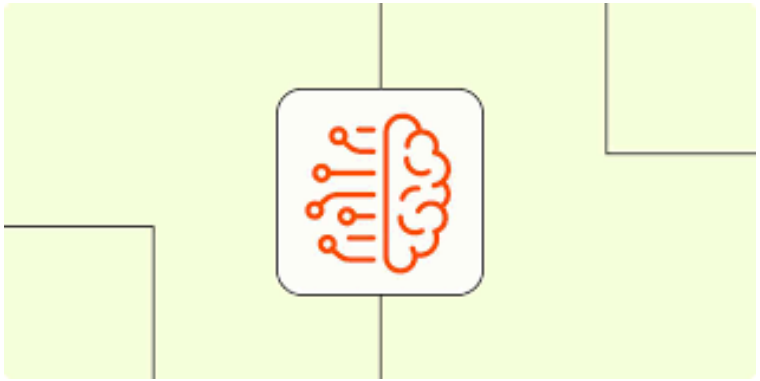
- Background & Motivation
- Methodology
- Analysis
- Related Work
- Conclusion

Mihika Rao (xsw5kn)

Background & Motivation

LLMs & Hallucinations in Code

- **LLMs** can sometimes produce unfounded or fabricated information
- Caused by statistical rather than factual reasoning
- Can happen in any domain (text, code, etc.)



Example of Hallucinated Package

- **Securehashlib** looks credible but does not exist in PyPI
- Developers might trust if they don’t double check
- A malicious actor can register the name for an attack

```
import securehashlib

def hash_password(password):
    return securehashlib.secure_hash(password, rounds=10000)
```

How Attackers Exploit Hallucinations

- **Monitor** LLM outputs (social media, dev forums, code snippets)
- **Identify** nonexistent package names (e.g., securehashlib)
- **Register** the package on PyPI/NPM/crates.io with malicious code
- **Developers** unknowingly install the now-real malicious code

Security Impact & Software Supply Chain

- **Open Repositories:** PyPI, NPM, crates.io allow anyone to register a package
- **Supply Chain Risk**: One bad dependency can compromise hundreds of projects
- **Typosquatting Parallel:** Attackers already use similar looking names to deceive developers (ex: reqeusts vs. requests).

Methodology

Models Evaluated in the Study

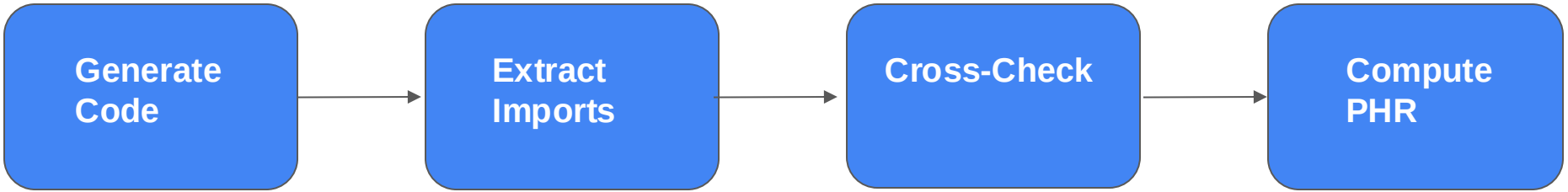
- Which LLMs were tested?
  - 10 different models from various providers
  - Includes both coding-specialized and general-purpose models

Importing Phantoms: Measuring LLM Package Hallucination Vulnerabilities

Label	Params	Code model?	Open weights?	Provider	Full name; reference
CodeGemma	7B	y	y	Google	CodeGemma 7B; CodeGemma Team (2024)
Dracarys	70B	y	y	Abacus.AI	Dracarys-Llama-3.1-70B-Instruct
GPT-4o	200B*	n	n	OpenAI	gpt-4o-2024-08-06; OpenAI (2024)
Granite-3.0	8B	y	y	IBM	Granite Team (2024)
Llama-3.1-8B	8B	n	y	Meta	Llama 3.1 8B
Llama-3.1-70B	70B	n	y	Meta	Llama 3.1 70B
Mamba-Codestral	7B	y	y	Mistral AI	Mamba Codestral 7B v0.1; Mistral AI team (2024)
Minitron-Mistral	8B	n	y	NVIDIA	Mistral-NeMo-Minitron-8B-Instruct; Adler et al. (2024)
Nemotron-Llama-3.1	70B	n	y	NVIDIA	Llama-3.1-Nemotron-70B-Instruct; Adler et al. (2024)
Qwen2.5-Coder	7B	y	y	Alibaba/Qwen	Qwen2.5-Coder 7B; Hui et al. (2024)
StarCoder2	15B	y	y	BigCode	StarCoder2-15B; Lozhkov et al. (2024)

Table 1. Models selected for assessment. *:GPT-4o parameter count is an approximation, from Abacha et al. (2024)

How Hallucinations Were Measured



Language	Repository	Count
JavaScript	NPM	3,391,235
Python	PyPI	604,814
Rust	crates.io	169,823

Example of Hallucination Detection

- **Prompt**: “Write a Python script that stores a password securely.”

```
import securehashlib

def hash_password(password):
    return securehashlib.secure_hash(password, rounds=10000)
```

How Hallucination is Detected

- Extract Import -> securehashlib
- Check PyPI -> Not Found -> Marked as hallucination
- Counted in PHR calculation

Hallucination Rates Across Languages & Models

	PHR (%)		
Model	**JavaScript**	**Rust**	**Python**
Dracarys	20.44	15.38	2.42
Codegemma	23.74	42.20	33.85
StarCoder2	14.51	31.65	27.03
Granite-3.0	24.62	42.86	46.15
Llama-3.1-70B	24.40	18.02	25.93
Llama-3.1-8B	11.43	28.79	5.49
Mamba-Codestral	14.95	14.29	33.85
Nemotron-Llama-3.1	0.22	0.22	4.84
Minitron-Mistral	10.77	24.62	33.41
GPT-4o	1.76	10.99	3.52
Qwen2.5-Coder	15.16	43.08	38.02

Table 3. Overall package hallucination rate for JavaScript, Rust, and Python

Nina Chinnam (fhs9af)

Presentation Outline

- Background & Motivation
- Methodology
- Analysis
- Related Work
- Conclusion

Analysis

Goals of Analysis

Importing Phantoms: Measuring LLM Package Hallucination Vulnerabilities

Label	Params	Code model?	Open weights?	Provider	Full name; reference
CodeGemma	7B	y	y	Google	CodeGemma 7B; CodeGemma Team (2024)
Dracarys	70B	y	y	Abacus.AI	Dracarys-Llama-3.1-70B-Instruct
GPT-4o	200B*	n	n	OpenAI	gpt-4o-2024-08-06; OpenAI (2024)
Granite-3.0	8B	y	y	IBM	Granite Team (2024)
Llama-3.1-8B	8B	n	y	Meta	Llama 3.1 8B
Llama-3.1-70B	70B	n	y	Meta	Llama 3.1 70B
Mamba-Codestral	7B	y	y	Mistral AI	Mamba Codestral 7B v0.1; Mistral AI team (2024)
Minitron-Mistral	8B	n	y	NVIDIA	Mistral-NeMo-Minitron-8B-Instruct; Adler et al. (2024)
Nemotron-Llama-3.1	70B	n	y	NVIDIA	Llama-3.1-Nemotron-70B-Instruct; Adler et al. (2024)
Qwen2.5-Coder	7B	y	y	Alibaba/Qwen	Qwen2.5-Coder 7B; Hui et al. (2024)
StarCoder2	15B	y	y	BigCode	StarCoder2-15B; Lozhkov et al. (2024)

Table 1. Models selected for assessment. *:GPT-4o parameter count is an approximation, from Abacha et al. (2024)

Programming Language

Model Type

Size

Coding Benchmarks

Programming Languages

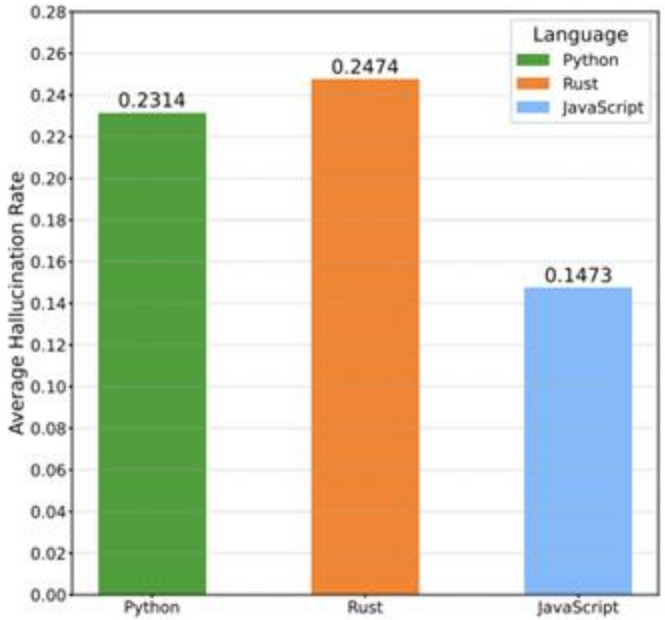


Figure 1. Package Hallucination Rate by language, averaged across all models

	PHR (%)		
Model	**JavaScript**	**Rust**	**Python**
Dracarys	20.44	15.38	2.42
Codegemma	23.74	42.20	33.85
StarCoder2	14.51	31.65	27.03
Granite-3.0	24.62	42.86	46.15
Llama-3.1-70B	24.40	18.02	25.93
Llama-3.1-8B	11.43	28.79	5.49
Mamba-Codestral	14.95	14.29	33.85
Nemotron-Llama-3.1	0.22	0.22	4.84
Minitron-Mistral	10.77	24.62	33.41
GPT-4o	1.76	10.99	3.52
Qwen2.5-Coder	15.16	43.08	38.02

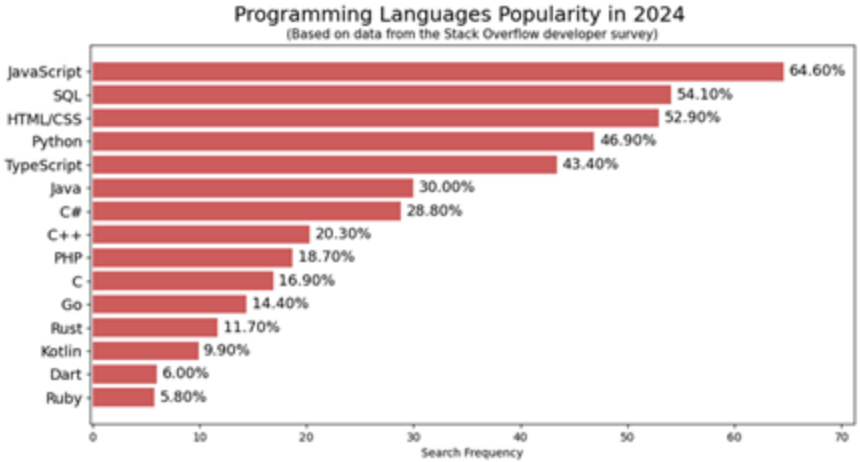
Table 3. Overall package hallucination rate for JavaScript, Rust, and Python

Programming Languages: Reasons for Variance

Ecosystem Size Matters

- JavaScript - 3.4 million
- PyPI - 604,814
- crates.io - 169,823

Training Data Composition



Coding-Specific vs. General-Purpose Models

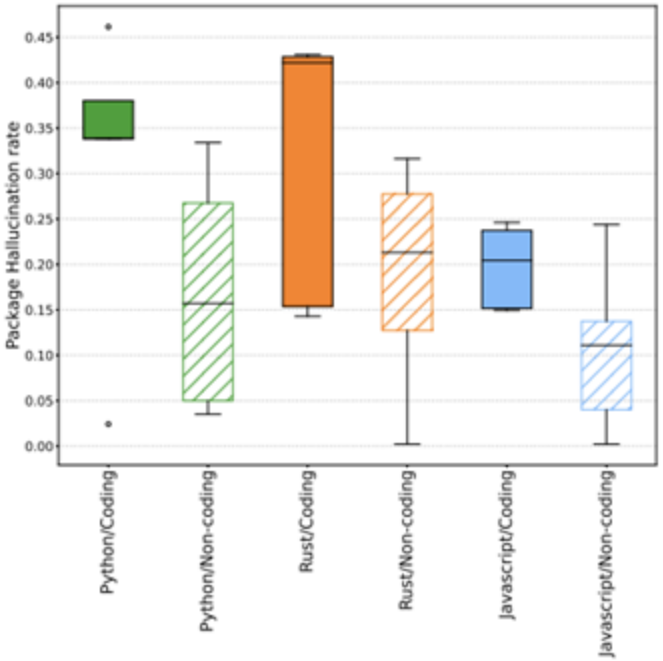


Figure 2. Distribution of package hallucination rate split by code-specialized and non-code models. Lower y-axis scores are better. Coding-specific models plots are solid-shaded, other models are hatch-shaded. Note that for every language, coding-specific models exhibit higher propensity to hallucinate packages.

		JavaScript	Rust	Python
Coding	μ	18.90%	31.58%	30.22%
	σ	4.59	15.28	16.67
Non-coding	μ	9.71%	16.53%	14.64%
	σ	8.86	11.88	13.50

Table 4. Package hallucination rates of coding-specific vs. non-coding models

Model Size

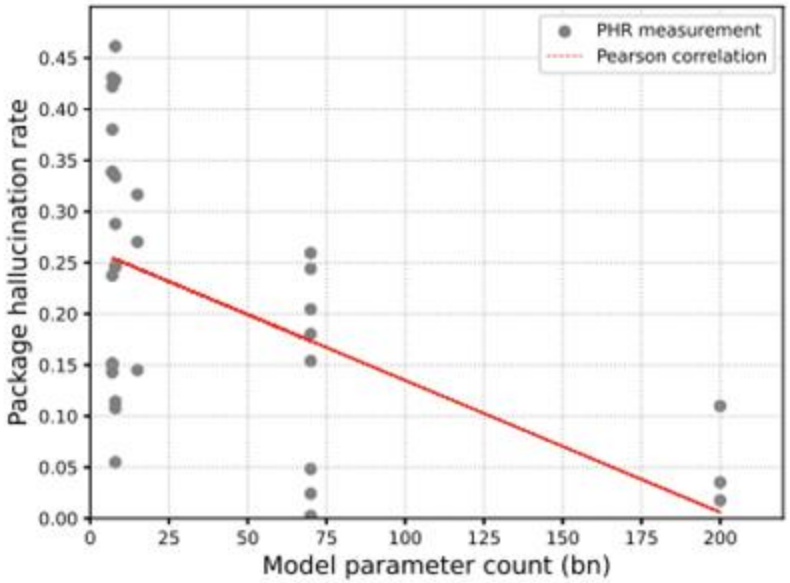


Figure 3. Model Size (x) vs PHR (y). Lower PHR is desirable. Small and low-hallucination models are in the bottom left. Pearson product-moment correlation coeff. of size vs. PHR is $-0.541, p = 0.00114$; coeff of ln(size) vs. PHR is $-0.593, p = 0.00028$.

	PHR (%)		
Model	**JavaScript**	**Rust**	**Python**
Dracarys	20.44	15.38	2.42
Codegemma	23.74	42.20	33.85
StarCoder2	14.51	31.65	27.03
Granite-3.0	24.62	42.86	46.15
Llama-3.1-70B	24.40	18.02	25.93
Llama-3.1-8B	11.43	28.79	5.49
Mamba-Codestral	14.95	14.29	33.85
Nemotron-Llama-3.1	0.22	0.22	4.84
Minitron-Mistral	10.77	24.62	33.41
GPT-4o	1.76	10.99	3.52
Qwen2.5-Coder	15.16	43.08	38.02

Table 3. Overall package hallucination rate for JavaScript, Rust, and Python

Coding Benchmark Scores

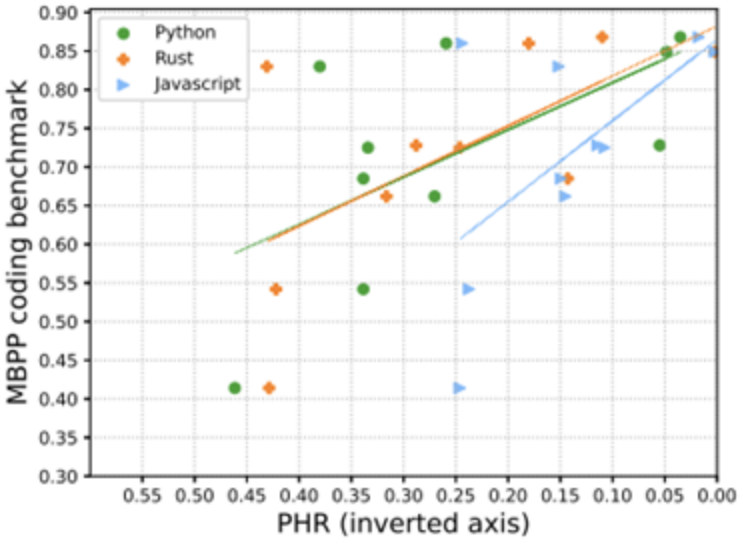


Figure 4. MBPP Coding score vs. package hallucination rate

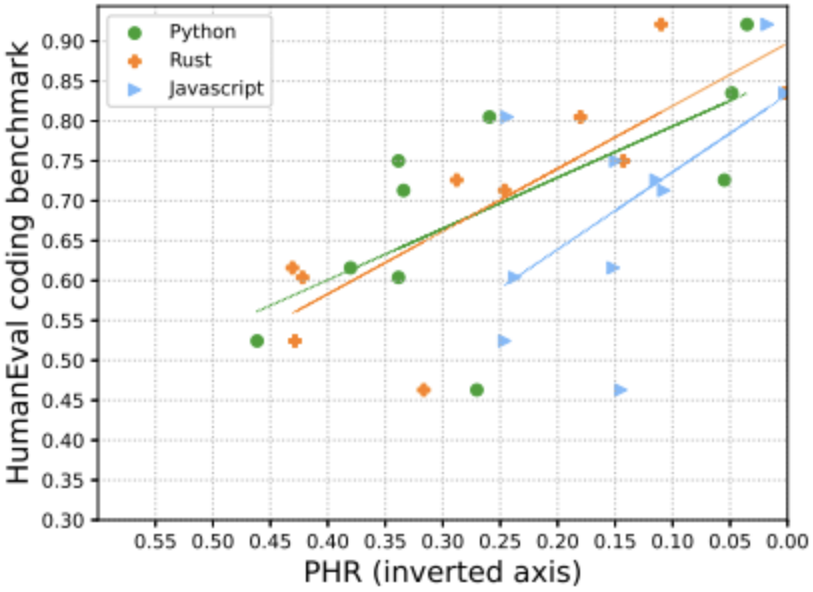


Figure 5. HumanEval Coding score vs. package hallucination rate

Proposed Mitigation Strategies

Verify Package Names Against Historical Data

Implement Hallucination Detection in AI-Assisted Coding

Encourage Developers to use Well-Known Libraries

Specify Preferred Packages when Prompting LLMs

Preemptive Package Registration and Monitoring

Related Work

Related Work

1. LLM Security Research

1. Package Confusion & Supply Chain Attacks

1. Analysis of Hallucination in Code-Generating LLMs

1. LLMs in Malware Detection

Conclusion

Key Findings

Research Area	Key Finding
Programming Language	Rust and Python had higher hallucination rates due to smaller package ecosystems
Coding-Specific vs. General Purpose Models	Coding specific tend to hallucinate more than general purpose
Model Size	Larger models tend to hallucinate less
Coding Benchmarks	Higher HumanEval scores correlate with lower hallucination rates

Future Work

Reducing Hallucination in Small Models

Code-Optimization and Security Vulnerabilities

Automated Hallucination Detection & Prevention

Questions?