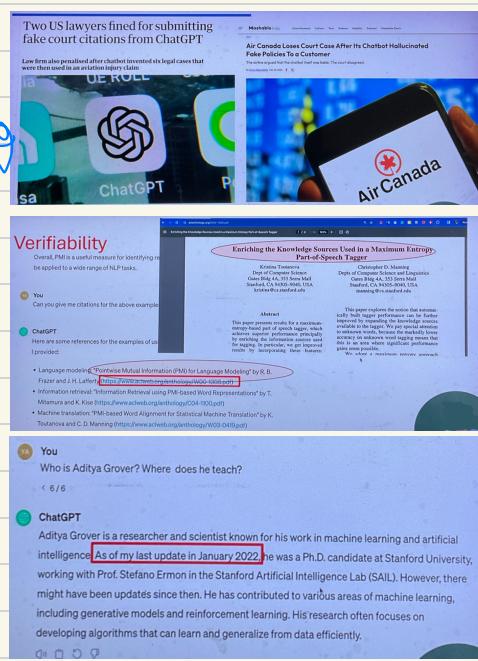


Core issues of Parametric LLMs

- * **Hallucination** can have repercussions which is a big problem when deploying LLMs in critical scenarios (especially where one can be held liable)
- * **Verifiability** is also an issue as it may give sources which exist but not correct.
- * **Knowledge cut-off** is another issue. A pretrained model will only have information up to a certain date & any new information about a person or a topic will not be available
- * **Learning failures** are also a problem. Models cannot memorize everything. They can still make mistakes.



Parametric LLMs - Training vs Testing

- * Parametric LLMs are called that because they store all their information in their parameters (weights). During training we show them the answers to questions & at test time we expect it to generate the answer from its own memory which was updated while training (without anything extra)
- * This is akin to giving a closed book exam where we read a book, learn & during the exam we're supposed to reproduce what we've learned from our memory. But is this mandatory?

* Contrast this with an open book exam, where a model is allowed to use external knowledge at the time of testing as well. This is more similar to a real world scenario because in the real world we don't perform tasks just based on memory; we reference other sources if needed. Such models are known as Retrieval based LMs where in addition to its own parameters, the model also uses information that it can access at the time of testing.

How to use the book?

Output Interpolations - Solve the problem yourself & use the book to verify if the answer is correct or not. eg: **kNN LMs**

Intermediate Fusion - Modify the architecture of the LM to be aware of the book. eg: **RETRO**

Input Augmentation - Given a query, go search the book, retrieve the relevant documents, give it to the LM to attend to while generating the answer. eg: **RAG, REALM**

How do you search in the book?

* But before we do anything, we need to know how to search in the book. This process is called - Retrieval

Retrieval Methods —

- Sparse retrieval
- Document level dense retrieval
- Token level dense retrieval
- Cross-encoder reranking
- Differentiable Search Index (DSI)
- Table of contents based search
- Black-box retrieval (Just ask Google/Bing)

Sparse Retrieval

* Express the query & document as a sparse word frequency vector (usually normalized by length)

q = what is nlp

what
candy
nlp
is
language
...

0.33

0

0.33

0.33

0

$d_1 = \text{what is life?}$
 candy is life!

0.125
0.125
0
0.25
0
...

$d_2 = \text{nlp is an acronym}$
for natural language
processing

0
0
0.125
0.125
0.125
...

$d_3 = \text{I like to}$
do good research
on nlp

0
0
0.125
0
0
...

$$q \cdot d_1 = 0.165$$

$$q \cdot d_2 = 0.0825$$

$$q \cdot d_3 = 0.0413$$

vector length
is the length
of the vocabulary

Q) How do we find the document that matches the query?

A) Cosine similarity / Dot product - because when we use dot product we get only those entries which are present both in the query & document. Rest of the values will be 0. So, more the no of common words, higher the similarity.

* Finally, the document with the highest inner-product on cosine similarity is chosen.

Leaves → In the above example, the document d_2 appears to be the one that can answer the question correctly but d_1 has the highest similarity score. Why?

- Stopwords are not eliminated

- All words have equal weightage (there is frequency but no implicit weight to the word itself)

So, words like what & is are overpowering the imp. words like nlp.

So, what can we do?

- ↳ stopword removal
- ↳ Term Weighting

Term Weighting (Manning et al 2009)

* Some terms are more important than others; low frequency words (NLP, candy) are often more important than (the, a, for, then, them...)

x) Term frequency - Inverse Document Frequency (TF-IDF)

$$TF(t, d) := \frac{freq(t, d)}{\sum_{t'} freq(t', d)}$$

$$IDF(t) = \log \left(\frac{|D|}{\sum_{d \in D} freq(t, d) > 0} \right)$$

→ Total no. of documents in the corpus
→ No. of documents containing the word t

$$TF-IDF(t, d) = TF(t, d) \times IDF(t)$$

- In IDF, if the denominator is low, the IDF will be high. So more weightage to terms that occur in fewer documents.

* BM25: TF term similar to smoothed count-based LMs

$$BM25(t, d) = IDF(t) \times \frac{freq(t, d) \cdot (k_i + 1)}{freq(t, d) + k_i (1 - b + b \cdot \frac{|d|}{avgdt})}$$

→ Takes care of saturation effect i.e., if a word occurs multiple times in a document we want to normalize its effect at some point.
→ length of the document
→ avg length of documents in the corpus

- Intuition: If there is a very big document in the corpus that contains almost every word in the vocabulary - it will show up in every search query & so we want to penalize that

Inverted Index

- * It is not feasible to iterate over all the documents & compute the dot product each time for a new query to retrieve the top documents.
- * To speedup this process we use an inverted index - a data structure that allows for efficient sparse lookup of vectors

Sparse Vectors			Index
	d ₁	d ₂	d ₃
what	2	0	0
candy	1	0	0
nlp	0	1	1
is	2	1	0
language	0	1	0
...

{ "what": [1], "candy": [1], "nlp": [2,3], "is": [1,2], "language": [2], ... }

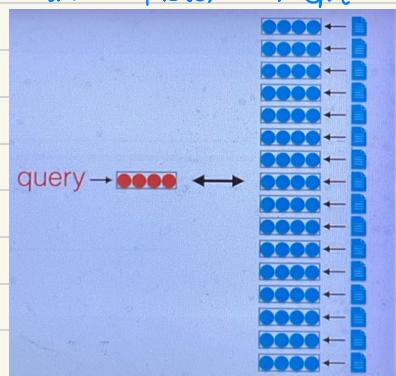
example software: Apache Lucene

Issues with Token based matching techniques

- Semantic meaning of the query as a whole is not captured
- Out of vocabulary (OOV) tokens are an issue.
- Things like synonyms are not captured.

Dense Embeddings

- * Encode all documents using a LM and index them (one time task). Can use:
 - Out-of-the-box embeddings. Eg: BERT
 - Learned Embeddings



- * At test time:

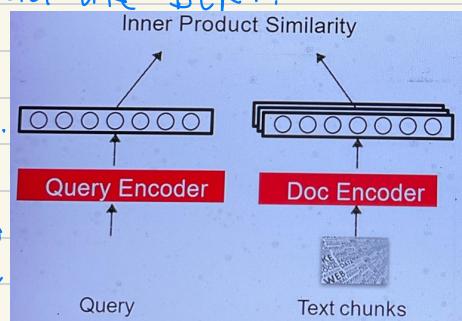
- Encode the query
- Use Nearest Neighbor Search to find similar documents

Training Dense Embeddings

- * Intuitively, the issue with using out-of-the-box embeddings is

that the query & document might be in different space. What makes us sure that the query & document embeddings will be aligned i.e., why will the 2 embeddings be similar when using an out-of-the box model like BERT?

- * Models like BERT, RoBERTa, etc. are trained on context word prediction & next sentence prediction kind of tasks. They might not be able to match the embeddings of queries & documents. Hence, we need to train models for this specific task.



- * For training, we want that if a query can be answered by a particular text chunk, then the query embedding should be close to that of the chunk.

So, how would the training data look like? - query, document pairs $q_i, d_i^+, d_i^-, d_2^-, d_3^- \dots$

↴ We need these negative samples because if we naively maximize the dot product of query & document, it might just maximize the score for any query & any document. There is no contrastive learning. i.e., we are not saying that 'x' query is answered by 'y' document so bring it close to 'y' and take it away from 'a', 'b' & 'c' which are irrelevant. We just try bring it close to 'y'. So, we might not end up getting good embeddings.

- * It is not feasible to compute the similarity w.r.t all the documents in the corpus during training. (Especially if there are millions of documents). So we need some kind of negative sampling.

$$L(q, p^+, p_1^-, p_2^-, \dots, p_n^-) = -\log \frac{\exp(\text{sim}(q, p^+))}{\exp(\text{sim}(q, p^+)) + \sum_{j=1}^n \exp(\text{sim}(q, p_j^-))}$$

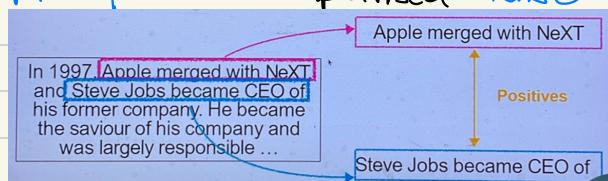
Contrastive Learning



- ④ How do we create the negatives?
- 2 solutions
 - Statically fix the negatives (won't work because this will learn to distinguish only the positives with a fixed set of negatives i.e., the ones we choose)
 - In-batch negatives (we create the batches randomly & hope that the query see all the other documents in the training corpus.)

- DPR (Karpukhin et al. 2020): Learn encoders based on a 8M 25 hard negatives & in-batch negatives
 - Hard negatives: Negative documents that are kind of close to positive document but don't really answer the query. e.g.: a document on general ML & AI for the query 'what is nlp?'
 - We do sparse retrieval & use the top documents retrieved (that are not the positive document) as negatives.

- Contriever (Izacard et al. 2022): contrastive learning using two random spans as positive pairs - Unsupervised dense retrieval model
 - ↳ Say we do not have query-document pairs. We can take 2 chunks from the same passage and treat them as positives for each other for training.



NOTE: We may have 2 different embedding models for document & query or we can have a common model & prefix the query & documents with some special tokens to identify which one is the query embedding & which is the document embedding.

Q) For sparse embeddings we had inverted index to speed up the process of searching for the nearest neighbors of a query? How do we search in the localized space of query for dense embeddings?

A) We can do some kind of clustering & have a representation for each cluster. Each query first matches with its cluster's embedding to narrow down the search & we can repeat this process recursively. This reduces the search time from $O(N) \rightarrow O(\log N)$

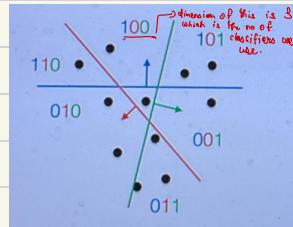
NOTE: We do train the model on hard negatives with BM25 so it may be possible that they get so far separated that are assigned to separate clusters that create issues. But this is not very likely since embeddings are naturally created in such a way that completely unrelated text should be much farther apart.

Approximate Nearest Neighbor Search - Max Inner Product Search (MIPS)

* Methods to retrieve embeddings in sub-linear time

Locality Sensitive Hashing

- Partitions the entire space into 2 parts (using a classifier) & keep doing it repeatedly.
- Ends up giving us a partitioning space.
- We also get a hashed map/sparse representation for each cluster
- When we get a query we hash it (by passing it through all the classifiers) & identify where it lies to then map it (alternatively, we can also create an inverted index with these sparse embeddings)



Graph-based Search

Graph-based search: create "hubs" and search from there

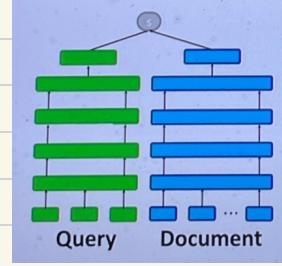


It is the same as the cluster based approach that has been described previously.

Softwares: ANNOD (Spotify), FAISS

Bi-Encoder Scoring

→ All the methods discussed till now are bi-encoder scoring i.e., we take the query & embed it, take the document independently & embed it & score them later on.



Advantage → All docs can be embedded offline & stored in the index for search.

Issues → In contrast to sparse embeddings which have a representation for each token, in dense embeddings the granular token level information is lost. Especially since the document chunks are huge & we force it to a smaller dimension like 768 or 1024.

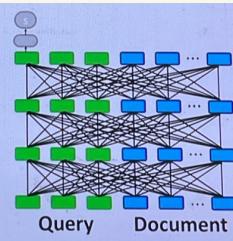
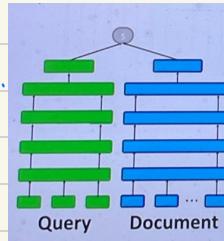
Q) How do we fix this i.e., how do we preserve the granular token level information?

Cross-Encoder Reranking

→ Jointly encode both the query & the document (Nogueira et al. 2019)

→ The resultant embedding will have taken into account all the tokens.

Issue → How do we go about searching for the document?



The embeddings cannot be pre-computed because now they are contextual embeddings, contextualized on the query.

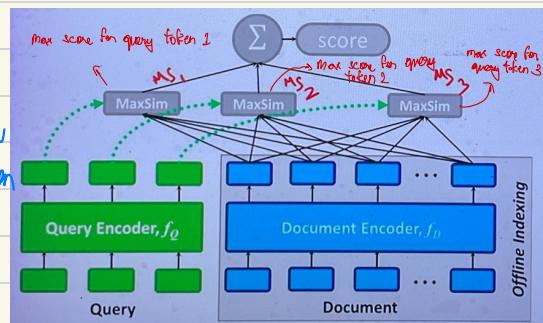
- * To compute the document embeddings at test time for each query is very difficult & hence this method cannot be used directly.
- * But we can re-rank using this model. We can use a bi-encoder to fetch (say) top 100 documents & re-rank them using the cross-encoder.
- * But it is still expensive (Need to compute 100 embeddings at test time)

Q) Can we get the best of both worlds somehow? (i.e., have the document embeddings stored but still have the query attend to each token in the document when computing the score instead of a single embedding)

Token level dense retrieval

CoBERT: Effective & Efficient Passage Search via Contextualized Late Interaction over BERT.

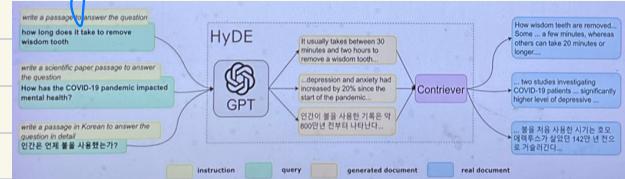
- * Document embeddings are computed offline but we store all the token embeddings instead of just the document embedding (So storage cost increases)
- * Once we fetch top 100 documents, then to rerank - we compute the similarity of every token in the query with every token in a document & take the max of it (for each token in the query). Then we add it to get the total similarity.
- * We repeat this process for all the top 100 documents



choose the best ones.

Hypothetical Document Embeddings (Gao et. al 2023)

- * We motivated learning a model with a belief that query & embedding lie in different spaces.
- * So, here we pass the query through an LLM (like GPT-3) & prompt it to generate a document that may contain the answer
- * Now we can use any out-of-the-box embedder to embed the document (because the generated hypothetical document & the actual document will be in the same space)
- * Can also be used with sparse encoders - can be thought of as query expansion where given a query we expand it to cover more words related to that (synonyms etc.). Then we can search with something like BM25 which was unable to handle synonyms (This is a hybrid approach)



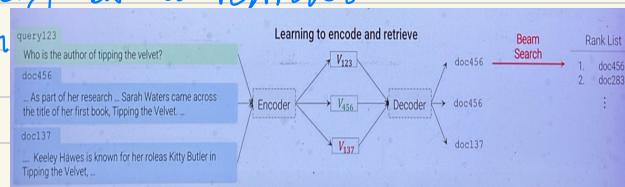
Differentiable Searchable Index (DSI)

- * Till now, we have created the retriever using an index & learned an embedding model to embed the query & documents and search using an index.
- * But LLMs are powerful enough to memorize the entire corpus.

Q) Can we use LLMs directly as a retriever?

A) We can have an encoder-decoder

model which takes in the query & documents (with ids) & spits out the correct document id.



- * Fully parameterized multi-stage retrieve then rank pipeline with a single neural model.
- * Train a seq2seq LLM for:
 - Indexing Task: Map each document to its doc id.
 - Retrieval Task: Map each query to the corresponding doc id.

We will have some identifier like $\langle I \rangle \{ \text{DOCUMENT HERE} \}$ which will output a doc_id for the indexing task

Similarly, we will have some identifier like $\langle Q \rangle \{ \text{QUERY HERE} \}$ which will output the doc_id for the retrieval task.

Q) How do we represent the document? And how do we represent the doc_id?

Document Representation

- Direct Indexing - first L tokens of the document
- Set Indexing - represent as a set of words after removing stopwords
- Inverted Index - Take random contiguous chunks

→ best experimentally (for the paper)

Doc id Representation

- Unstructured Atomic Identifiers

↳ Use a token to represent id of a document

↳ Take softmax over doc_id tokens

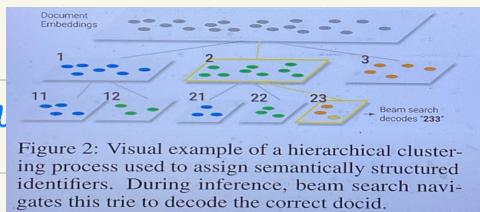
$$\text{New vocab size: } |V| + |\mathcal{D}|$$

old vocab size document corpus size

- Naively Structured String Identifiers

↳ Decode the string representation of the doc_id e.g.: doc_id5

- Semantically Structured Identifiers
 ↳ Create a hierarchical tree structure over document embeddings



Results

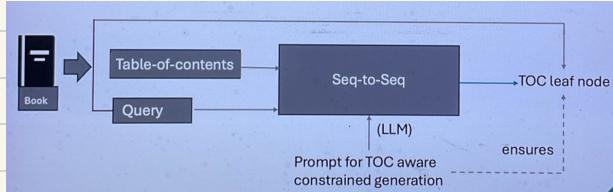
Model	Size	Params	Method	NQ10K		NQ100K		NQ320K	
				Hits@1	Hits@10	Hits@1	Hits@10	Hits@1	Hits@10
BM25	-	-	-	12.4	33.5	20.9	46.4	11.6	34.4
T5	Base	220M	Dual Encoder	16.2	48.6	18.7	55.2	20.5	58.3
T5	Large	800M	Dual Encoder	18.8	55.7	22.3	60.5	22.4	63.3
T5	XL	3B	Dual Encoder	20.8	59.6	23.3	63.2	23.9	65.8
T5	XXL	11B	Dual Encoder	22.1	61.6	24.1	64.5	24.3	67.3
DSI	Base	250M	Atomic Docid	13.0	38.4	23.8	58.6	20.7	40.9
DSI	Large	800M	Atomic Docid	31.3	59.4	17.1	52.3	11.6	37.6
DSI	XL	3B	Atomic Docid	40.1	76.9	19.0	55.3	28.1	61.9
DSI	XXL	11B	Atomic Docid	39.4	77.0	25.3	67.9	24.0	55.1
DSI	Base	250M	Naive String Docid	28.1	48.0	18.7	44.6	6.7	21.0
DSI	Large	800M	Naive String Docid	34.7	60.5	21.2	50.7	13.3	33.6
DSI	XL	3B	Naive String Docid	44.7	66.4	24.0	55.1	16.7	58.1
DSI	XXL	11B	Naive String Docid	46.7	77.9	27.5	62.4	23.8	55.9
DSI	Base	250M	Semantic String Docid	33.0	57.3	19.0	44.9	27.4	56.6
DSI	Large	800M	Semantic String Docid	37.5	65.1	20.4	50.2	35.6	62.6
DSI	XL	3B	Semantic String Docid	41.9	67.1	22.4	52.2	39.1	66.8
DSI	XXL	11B	Semantic String Docid	48.5	72.1	26.9	59.5	40.4	70.3

- * Significant improvement over the baseline dual encoder where performance nearly doubles for NQ10K subset.
- * New direction in the retriever space.

TOC aware Search

- * Prompt an LLM with TOC (Table of Content) & ask a question, and ask it to generate the chapter title/subsection containing the answer.
- (Q) Can the LLM answer in zero shot? Can we fine tune an LLM to answer such queries?

(Work still under submission)

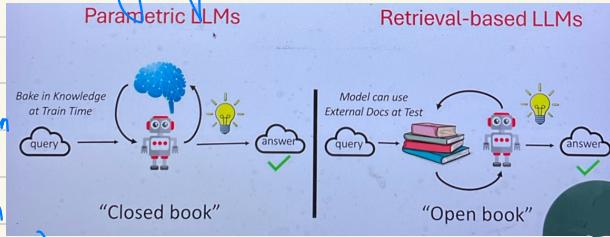


Preliminary Results

Domain	Book #	Title	#Pages	#TOC Nodes	#TOC Depth	#Test Queries	mstralai/Mistral-7B-Instruct-v0.2 BM25	DSI	TOC Search	Gain DSI	Gain BM25	
Education	Book 1	Action Research	137	8	1	437	0.35	0.63	0.746	0.794	4.8	15.6
Finance	Book 1	Principles of Financial Accounting	318	129	3	1214	0.199	0.599	0.681	0.752	7.1	15.3
Law	Book1	Introduction to Intellectual Property	209	52	2	820	0.212	0.751	0.833	0.854	2.1	10.3
Medicine	Book1	Nursing Assistant	659	132	2	1996	0.39	0.7	0.783	0.876	9.3	17.6
Natural Sciences	Book1	Introduction to Soil Science	182	42	2	579	0.461	0.741	0.839	0.874	3.5	13.3
Social Sciences	Book1	Foundations of Aural Skills	674	132	2	956	0.234	0.523	0.506	0.698	19.2	17.5
Education	Book 2	Teaching Methods and Practices	151	29	3	183	0.607	0.76	0.842	0.913	7.1	15.3
Finance	Book 2	Financial Accounting	572	96	2	2177	0.242	0.684	0.723	0.823	10	15.9
Law	Book 2	Business Law, Ethics and Sustainability	652	14	1	2697	0.476	0.781	0.879	0.884	0.5	10.3
Medicine	Book 2	Nursing Fundamentals	1327	143	2	3837	0.178	0.655	0.792	0.826	3.4	17.1

Lecture 16.2 - Retrieval-based Language Models - II

- As opposed to parametric LLMs, in retrieval based LLMs we can access external information at test time.
- 3 ways of using this information
 - Output Interpolations (kNN-LM)
 - Intermediate fusion (RETRO)
 - Input Augmentation (RAb)

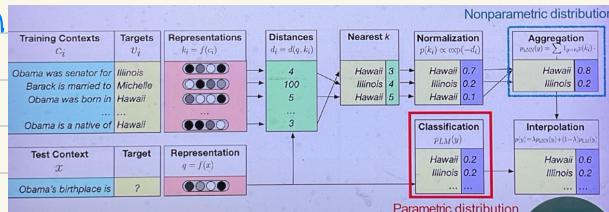
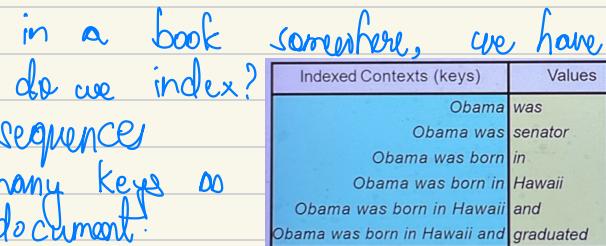
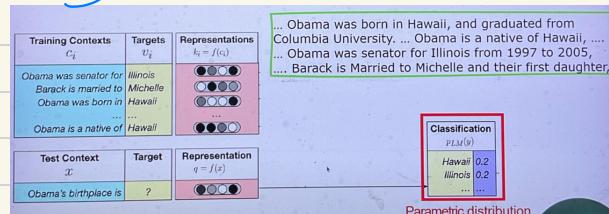


Common question in all 3 approaches: How to read from the book?
covered in previous lecture ↪

How to use the book?

kNN-LM (Khandelwal et.al. 2019)

- LM's task is predict the next token given a context.
- When we pass a query through an LM we get a distribution over the possible answers
- Now, given some information in a book somewhere, we have already indexed it. And what do we index?
 - We index all possible subsequences
 - So we end up creating as many keys as the no. of tokens in the document.
- When we get a query, we search for it in the index, we get the top-k indices (with min. distance), and we normalize the distances



to get the probability distribution (over the values)

f) Since, a token can be repeated so we aggregate that to get the distribution over the next token from the index.

g) We combine this distribution with the output distribution from the parametric LM using a linear interpolation to get the final distribution.

$$P_{\text{KNN-LM}}(y|x) = (1-\lambda) P_{\text{LM}}(y|x) + \lambda P_{\text{KNN}}(y|x)$$

\hookrightarrow hyperparameter

KNN-LM - Computational Cost

* Key Embedding Computation

- Single forward pass over the data - fraction of cost for training one epoch.

* Building cache using FAISS Index

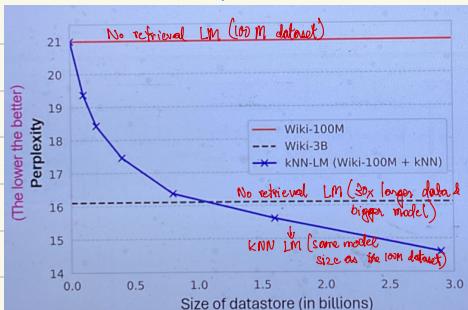
- 10³M - 2 hours on a single CPU

↳ Wiki 10³M

* Inference Overhead

- 260k tokens: 25 minutes with k=1024 (On validation dataset)

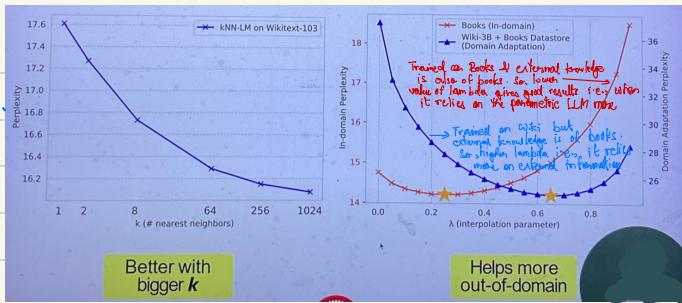
KNN-LM - Results



* Advantage - Can combine any pretrained LM with the external corpus.

* KNN-LM trained model outperforms even a model pretrained on a much larger corpus of data.

- * Increasing the no of neighbors retrieved improves performance.
- * Performs well on out-of-domain datasets as well.



KNN-LM how to finetune on downstream tasks?

- * In LM task, "input" is a sub-sentence, & "output" is the "next word". KNN-LM organizes the "unstructured knowledge" as "input-output" pairs. We search for the most similar "input" (sub-sentence) in the corpus, and use its corresponding "output". But how to fine-tune such a model for a downstream task?

- for that we would need the most similar "input"
 - ↳ i.e., need examples labeled for the target task.
 - ↳ Not clear how to organize unstructured text as input-output pairs for the desired task.

RETRO - Retrieval Enhanced Transformer

- * A key for each token in KNN-LM creates 2 issues
 - Restricts corpus size
 - Returns distribution over only k tokens

- * Instead of retrieving just the next token, we can retrieve the entire continuation itself. So, the retriever now returns the key along with its continuation in the passage.
- * This also reduces the size of the index drastically. e.g.:

Indexed Contexts (keys)	Values
Obama was	... Obama was born in Hawaii, and graduated from Columbia University. Obama is a native of Hawaii, Obama was senator
Obama was senator	
Obama was born in	
Obama was born in Hawaii	... Barack is Married to Michelle and their first daughter...

In kNN LM	
• We have an indexed key in KNN LM for each token. This causes two issues:	<ul style="list-style-type: none"> • Restricts the size of corpus that can be indexed. • k-neighbors returns only k tokens. • What if we retrieve the entire continuation instead of just one token?

Indexed Keys (N)	Values (N,F)
Obama was born in Hawaii and graduated from Columbia University	... Obama was born in Hawaii and graduated from Columbia University
and graduated from Columbia University	... and graduated from Columbia University. Obama is a native of Hawaii

If key length is 100 tokens then the index is 100 times smaller than kNN-LM index.

Q) How to use the k retrieved chunks?

- Split the input into smaller chunks (Remember, it is still a language modeling task)

x = World Cup 2022 was / the last with 32 teams / before the increase to 48 in 2026.

- At the end of each input chunk,

↳ retrieve "k" chunks similar to the input chunk.

(We have a decoder-only model. When we are processing 'was' to get a distribution over the next token, we have already decoded the previous tokens. So, we can use x_i to retrieve information from the index - i.e., k chunks simultaneously)

NOTE: The retrieved values will have the continuation as well.



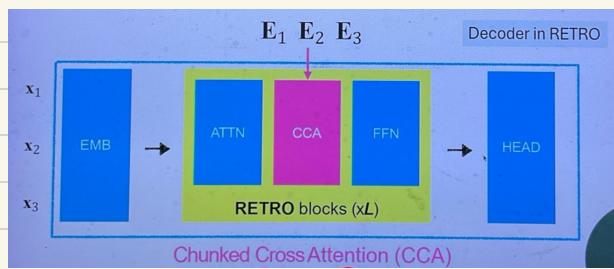
- x_1 , x_2 & x_3 are the

retrieval queries. It goes to a retrieval encoder & gets encoded (or embedded), we search in the index & we retrieve k chunks for each split (NOTE: This can be done in parallel only in training time since we already have the text sequence (that must be generated), but at test time it is done as all decode)

- The retrieved chunks are then encoded (E_1 , E_2 & E_3 in above image) as well. Each encoding block has k-chunks, with r-tokens per chunk & d-dimension vector per token so the output is an $r \times k \times d$ matrix.

How to use the k retrieved chunks?

- * To do this, they change the transformer block & add a cross-attention layer in between.
- * After the self-attention block, we attend over E_1 , E_2 & E_3 as well. NOTE: We can start attending on E_1 only from the last token of the x_1 block (i.e., when we use it as input) because only when we have that token (as the output of previous step) can we retrieve the E_1 block & then start attending on it. We can do this till we reach the last token of x_2 after which we start attending on E_2 .

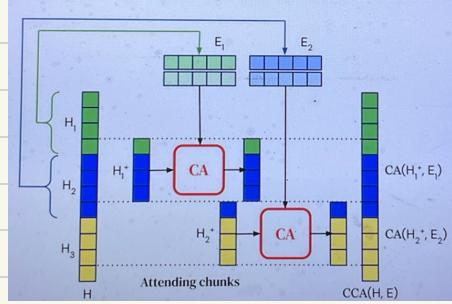
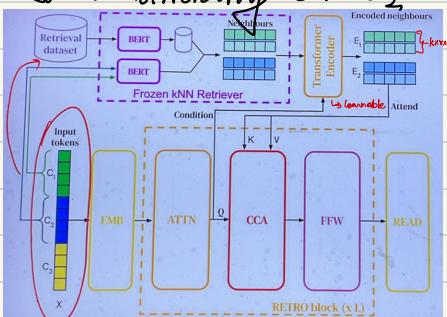


attending on E_1 only from the last token of the x_1 block (i.e., when we use it as input) because only when we have that token (as the output of previous step) can we retrieve the E_1 block & then start attending on it. We can do this till we reach the last token of x_2 after which we start attending on E_2 .

- * The kNN retriever is frozen whereas the transformer that encodes the retrieved documents is a learnable model.

- * We perform Chunked Cross Attention (CCA) in the following way:

- for the last token on the x_1 block & all (but the last) token on x_2 we attend on E_1 & get its embeddings
- for the last token on x_2 & all (but the last) token on x_3 we attend on x_2 & so on.
- We concatenate them all to get the final embedding.



RETRO Results

- * Since, it indexes chunks instead of tokens, RETRO is able to

Model	Retrieval Set	#Database tokens	#Database keys	Valid	Test
Adaptive Inputs (Baevski and Auli, 2019)	-	-	-	17.96	18.65
SpanLM (Weninger et al., 2021)	Wikipedia	3B	3B	12.20	17.60
kNN-LM (Khandelwal et al., 2020)	Wikipedia	3B	3B	16.06	16.12
Megatron (Shoeybi et al., 2019)	-	-	-	-	10.81
Baseline transformer (ours)	-	-	-	21.53	22.96
RETRO (ours)	Wikipedia	4B	4B	18.52	19.54
RETRO	Wikipedia	4B	0.06B	18.46	18.97
RETRO	C4	174B	2.9B	12.87	10.23
RETRO	MassiveText (1%)	18B	0.8B	18.92	20.33
RETRO	MassiveText (10%)	179B	4B	13.54	14.95
RETRO	MassiveText (100%)	1792B	28B	3.21	3.92

index significantly more tokens compared to kNN-LM which reduces the perplexity drastically ($16 \rightarrow 4$ as shown above)

How to fine-tune RETRO for downstream tasks?

- * fine-tune on NQ (Question Answering dataset)
- * Format the data as: "[PAD] question: Question 3\nanswer: [answer]

↓
this is to ensure that "answer"
coincides with end of first
chunk (i.e., size of X_1, X_2, X_3)

↑
this is what the LM
should generate

- * They retrieve information using this input & generate after that.
- * fine tuned a 7.5B model using 26k steps with 20 retrieved passages for each example.

Results

- Performs similar to other methods, except FiD
- Increasing the # retrieved passages beyond 20 doesn't help.

Model	Test Accuracy
REALM (Guu et al., 2020)	40.4
DPR (Karpukhin et al., 2020)	41.5
RAG (Lewis et al., 2020)	44.5
EMDR ² (Sachan et al., 2021)	52.5
FiD (Izacard and Grave, 2021)	51.4
FiD + Distill. (Izacard et al., 2020)	54.7
Baseline 7B (closed book)	30.4
RETRO 7.5B (DPR retrieval)	45.5

frequency of calling the retriever

- * RETRO triggers retriever after every L tokens
- * Can we trigger it on demand?
 - Generate a spl token that triggers retriever call [Toolformer Schick et al., 2023]
 - Call it when LM itself is uncertain about the prediction [FLARE Jiang et al., 2023]
 - RIBR - Retrieval Interleaved Generation [Radhakrishnan et al., 2024]

ToolFormer

- * LM generates [QA (Who is the ...)] → and then the question goes to the retriever which gives the answer that we fill in.
- * Can also be used to invoke external tools & not just external knowledge.
e.g.: calculator

The New England Journal of Medicine is a registered trademark of [QA("Who is the publisher of The New England Journal of Medicine?") → Massachusetts Medical Society] the MMS.

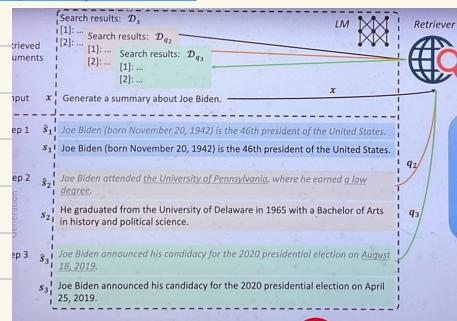
Out of 1400 participants, 400 (or [Calculator(400 / 1400) → 0.29] 29%) passed the test.

The name derives from "la tortuga", the Spanish word for [MT("tortuga") → turtle] turtle.

The Brown Act is California's law [WikiSearch("Brown Act") → The Ralph M. Brown Act is an act of the California State Legislature that guarantees the public's right to attend and participate in meetings of local legislative bodies.] that requires legislative bodies, like city councils, to hold their meetings open to the public.

Active Retrieval-Augmented Generation - FLARE

- * first retrieve something with a query. D_x here is the 1st set of results.
- * Generate a response → accept if you are confident about it!
- * How to compute confidence?
 - Probability
 - 2nd sentence → say you are not confident about it → go back to the retriever, fetch information (D_{q_2}) & generate again.
 - 3rd sentence - repeat (i.e., check confidence; if not confident, go to retriever)

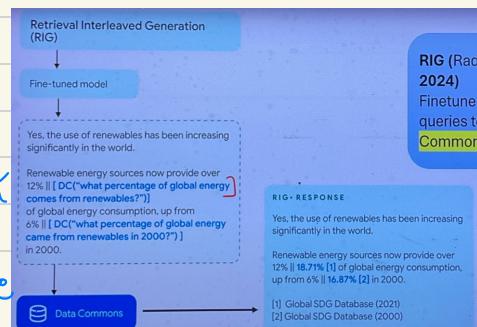


Retrieval Interleaved Generation (RIG)

- * Similar idea to ToolFormer.
- * Create a public open source knowledge Graph (KG) called Data Commons.
- * Train the LM to generate queries that can be answered by the KG.

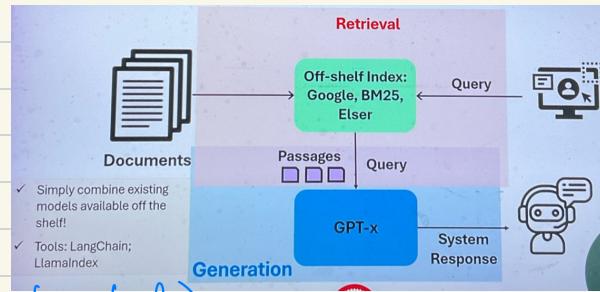
Data Commons → open source initiative by Google

- ↳ Aims to organize world's public datasets in a KG.
- ↳ Encompasses a large range of statistical data from public sources such as - UN, health dept., NGOs, economic agencies, environment agencies.
- ↳ Includes over 250B data points & 2.5 trillion triples from hundreds of global sources.



RAG - Architecture

- *) Idea here is to call the retriever only once & use that information.
- *) RAG Architecture is as shown in this image -
 - Index all the documents (one time task)
 - When a query comes, we query the index & get the top-k passages from the retriever
 - We concatenate the passages to the query & pass it to the LM to get the system response
- *) Can use any off-the-shelf retriever (or even a Google search) & any off-the-shelf LM as a generator because most of the LMs these days are fine-tuned for reading comprehension & summarization kind of tasks where they learn to answer from the given context.
- *) Trainable components in retrieval based LMs
 - ↳ Retriever
 - ↳ Generator



REALM (Guu et. al 2020): Retrieval-Augmented Language Model

Pre-Training

RAG (Lewis et. al 2020): Retrieval-Augmented Generation for knowledge-intensive NLP tasks

- *) The above 2 works jointly train the retriever & generator.
 - A) The retriever ranks the documents given a query. It has 2 trainable components - query encoder (to encode the query) & document encoder (to encode the documents & create an index)
 - A) We search using a similarity (or distance) metric b/w the

embedding of the query & document

REALM

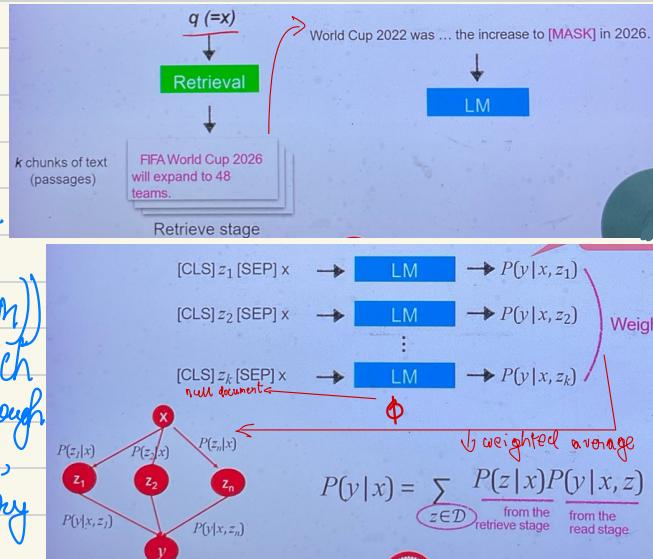
* focuses on masked language models (like BERT)

* we give an input with masked tokens & the task is to predict the [MASK] token.

(this is the pre-training part of the Masked Language Model (MLM))

* we modify the pre-training such that we pass the query through the retriever, get the k chunks, append the chunks to the query one-by-one, get the output distributions for each chunk appended query. (Another way could be to append all the chunks together)

* To combine those distributions to get a single output distribution, we marginalize i.e., a weighted average weighted on the probability of retrieving a chunk given a query.



$$P(y|x) = \sum_{z \in D} P(z|x) P(y|x, z)$$

from retrieve stage from the read stage

Q What are the issues in maximizing?

- We have 2 sets of parameters η & θ for the retriever & generator but it is possible to jointly maximize them.
- A practical challenge is that to find $P(y|x)$ we need

to sum it up over all the documents. But the size of the index may be in billions so it is not possible. So, we assume that $P(z|x)$ is non-zero only for the retrieved top- k documents & zero for the rest.

$$z \in D \rightarrow z \in \text{top-}k$$

$$\text{Maximize } \sum_{z \in \text{Top-}k(p_n(\cdot|z))} p_n(z|x) p_o(y_{[\text{MASK}]} | x, z)$$

↓
2 parameters η_q (for query) & η_d (for document)

$$e_x = E_{\eta_q}(x)$$

$$e_z = E_{\eta_d}(z)$$

$$P(z|x) \propto \text{similarity}(e_x, e_z)$$

$p_o(y_{[\text{MASK}]} | x, z)$ is just the output of the transformer model for the [MASK] token. (NOTE: not the output of previous token. This is not an auto-regressive model)

REALM : Training Approximations

- * Once we backpropagate, the η_q , η_d & D weights are supposed to change.
- * But η_d was used to index.
- * So, do we re-index after every training step?
- * The above is not feasible so we update the index at a certain frequency.

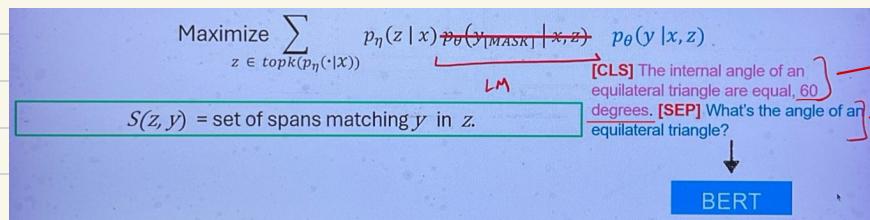
- * So, we:
- freeze top-k documents
 - freeze index (doc. embeddings), but search top-k documents
 - update every T steps

NOTE: η_θ still gets updated

REALM: fine-tuning

- * Pre-training was simple as we just needed the output of the mask token but now we have questions & answers like an (x, y) pair.

$$\text{Maximize} \sum_{z \in \text{topk}(\eta_\theta(\cdot | x))} p_\eta(z | x) p_\theta(y_{[\text{MASK}]} | x, z) p_\theta(y | x, z)$$



- * So, say we have a question & we have some retrieved information as shown in the fig above. The answer here is **60 degrees**.
- * The model assumes that the answer is a span of the document
- * 60 degrees is a span of the retrieved document.
- * During training, we find all the spans which match with the answer & try to maximize its probability.

$S(z, y) = \text{set of spans matching } y \text{ in } z.$
$p(y z, x) \propto \sum_{s \in S(z, y)} \exp (\text{MLP}([h_{\text{START}(s)}; h_{\text{END}(s)}]))$
$h_{\text{START}(s)} = \text{BERT}_{\text{START}(s)}(\text{join}_{\text{BERT}}(x, z_{\text{body}})),$
$h_{\text{END}(s)} = \text{BERT}_{\text{END}(s)}(\text{join}_{\text{BERT}}(x, z_{\text{body}})),$

SPAN - Any contiguous subsequence of the sequence (including one letter words)

How?

The information & question are passed through a language model and we get the embeddings over all the tokens. To find the probability of '60 degrees' as an answer we take the start & end token, concatenate it and pass it through an MLP to get the score. We do this for all spans & then normalize to get the probability.

Σ Q

60 degrees

Cold Start Problem

- * Initially, the retriever & the LM are both initialized with random values, so the retriever retrieves mostly junk & the LM might learn to just ignore the information from the retriever. This is called the Cold Start problem.
- * So, the idea is to first pre-train both the retriever & LM separately & once they are at a certain level, then train jointly.
- * **Inverse Cloze Task:** method to pretrain retriever embeddings
 - from the document, take a random sentence in between as the query & the remaining part of the context (i.e., the surrounding text) as the document & try to increase their similarity.

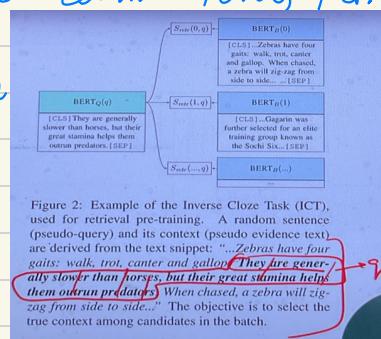


Figure 2: Example of the Inverse Cloze Task (ICT), used for retrieval pre-training. A random sentence (pseudo-query) and its context (pseudo evidence text) are derived from the text snippet: "...Zebras have four gaits: walk, trot, canter and gallop. They are generally slower than horses, but their great stamina helps them outrun predators. When chased, a zebra will zig-zag from side to side..." The objective is to select the true context among candidates in the batch.

REALM-Results

* Exact match of 40% on NQ.

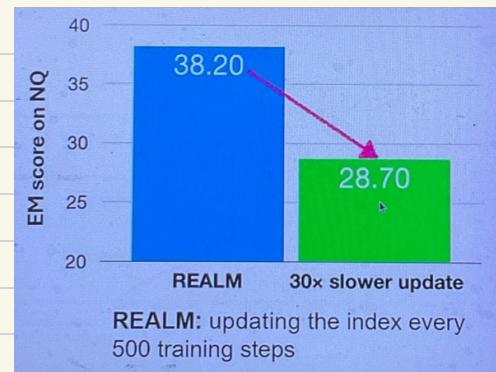
ORQA = REALM - joint pretraining with retriever.

Name	Architectures	Pre-training	NQ (79k/4k)	WQ (3k/2k)	CT (1k /1k)	# params
Baselines with Frozen retriever + reranking						
DrQA (Chen et al., 2017)	Sparse Retr.+DocReader	N/A	*	20.7	25.7	34m
HardEM (Min et al., 2019a)	Sparse Retr.+Transformer	BERT	28.1	-	-	110m
GraphRetriever (Min et al., 2019b)	GraphRetriever+Transformer	BERT	31.8	31.6	-	110m
PathRetriever (Asai et al., 2019)	PathRetriever+Transformer	MLM	32.6	-	-	110m
ORQA (Lee et al., 2019)	Dense Retr.+Transformer	ICT+BERT	33.3	36.4	30.1	330m
REALM						
Ours (λ' = CC-News, Z = Wikipedia)	Dense Retr.+Transformer	REALM	40.4	40.7	42.9	330m

REALM: Index update rate

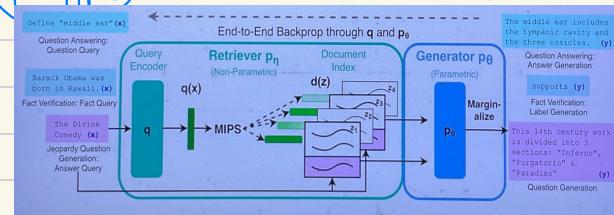
* If we update the index often, then the performance is much better.

But updating too frequently is very expensive & thus infeasible.



RAG₁

- * REALM was a good work & introduced the concept of joint training. However, it was restricted to Masked Language Models (encoder only type) & thus not suited for generation tasks.
- * At the same time, RAG₁ focused more on generation tasks & could work on multiple problems (as it is a decoder only model). e.g.: QA, fact verification, etc.



RAG₁: Joint Training Equation

$$\text{Maximize} \sum_{z \in \text{topk}(p(\cdot|x))} p_n(z|x) p_o(y|x, z)$$

$$= \sum_{z \in \text{topk}(p(\cdot|x))} p_n(z|x) \prod_i^N p_o(y_i|x, z, y_i; \theta)$$

- * Given a document, we compute the probability of generating an entire sequence at once & then multiply it with the probability of retrieving the document.

Then we marginalize over all the retrieved documents

Q) Can we do it another way? Can we generate one token given all documents, and then proceed to the next token?

$$\text{Maximize} \prod_{i=1}^N \sum_{z \in \text{Top-}k(p(\cdot|x))} p_n(z|x) p_0(y_i|x, z, y_{1:i-1})$$

↳ Probability $(y_i|x, z_1, \dots, z_k)$

RAG: Results

Table 1: Open-Domain QA Test Scores. For TQA, left column uses the standard test set for Open-Domain QA, right column uses the TQA-Wiki test set. See Appendix D for further details.

Model	NQ	TQA	WQ	CT
Closed Book	T5-11B [52]	34.5	- / 50.1	37.4 -
Open Book	T5-11B+SSM [52]	36.6	- / 60.5	44.7 -
REALM [20]	40.4	- / -	40.7	46.8
DPR [24]	41.5	57.9 / -	41.1	50.6
RAG-Token	44.1	55.2 / 66.1	45.5	50.0
RAG-Seq.	44.5	56.8 / 68.0	45.2	52.2

* Compared to REALM, get 44.5% exact match.

Training Methods for retrieval augmented LMs

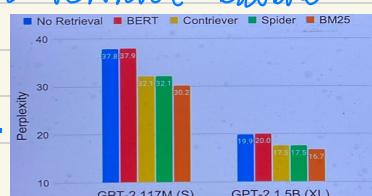
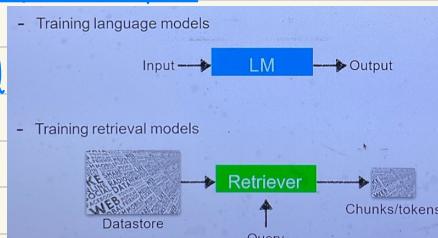
① Independent training

* Retrieval & language models are trained independently

* This is what is used mostly these days where we use out-of-the-box retriever & language model for applying on our dataset & our specific task.

* The LM has learnt to use In-Context information during pretraining & fine-tuning & even though they are not retriever aware they give good results.

* Improving each component separately, we can improve the overall performance of the system.



<u>Advantages</u>	<u>Disadvantages</u>
<ul style="list-style-type: none"> - Work with off-the-shelf models (no extra training required) - Each part can be improved independently 	<ul style="list-style-type: none"> - LMs are not trained to leverage retrieval - Retrieval models are not optimized for LM tasks/domains

② Sequential Training

- * Like in RETRO where we do freeze the retriever & train only the LM.
- * Similarly, there is REFLUG, where we freeze the LM and train only the retriever

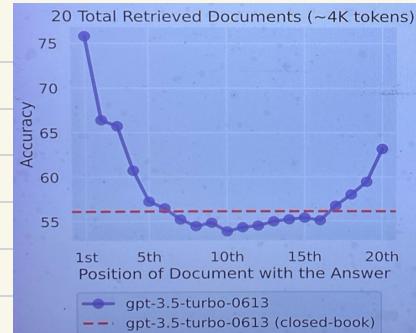
<u>Advantages</u>	<u>Disadvantages</u>
<ul style="list-style-type: none"> - Work with off-the-shelf components - LMs are effectively trained to leverage retrieval results - Retrievers are trained to provide text that helps LMs the most. 	<ul style="list-style-type: none"> - One component is still fixed & not trained.

③ Joint Training

<u>Advantages</u>	<u>Disadvantages</u>
<ul style="list-style-type: none"> - End-to-end trained - each component is optimized - Good performance 	<ul style="list-style-type: none"> - Training is more complicated (async update, overhead, data batching, cold start, etc) - Train-test discrepancy still remains

Issues - Lost in the Middle!

- * As context increases, models miss relevant info.
- * "lost-in-the-middle" (Liu et al., 2023) demonstrates that if the context of the generator increases a lot then the model focuses more either on the starting or end tokens. The tokens in the middle are paid less attention to.

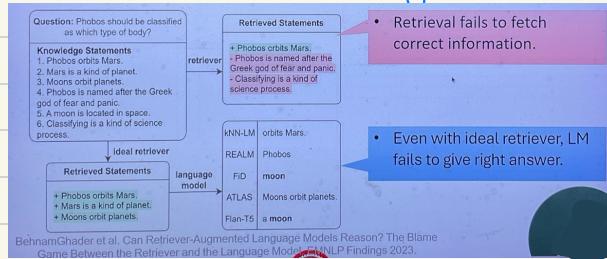


Issue - Hallucination

- * Retrieval augmented LMs can still hallucinate.
- * Can we quantify hallucination in a post facto manner?

i.e., we use the RAG to generate the answer but then use another model/method to quantify whether the answer is faithful to the retrieved passages. This is an area of active research.

Issues - Retrieval & Reasoning failures

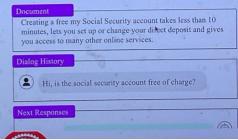


Pointwise Mutual Information Based Metric and Decoding Strategy for Faithful Generation in Document Grounded Dialogs

Yatin Nandwani, Vineet Kumar, Dinesh Raghu, Sachindra Joshi and Luis A. Lastras
IBM Research, AI
(yatin.nandwani@, vineeku@in, diragh@in, jsachind@in, lastrasl@us).ibm.com

Abstract

A major concern in using deep learning based generative models for document-grounded dialogs is the potential generation of responses that are not *faithful* to the input document. Existing automatic metrics used for evaluating the faithfulness of response with respect to the grounding document measure the degree of similarity between the generated response and the document's content. However, these methods



Adapting LMs to Domain Corpora?

RAFT: Adapting Language Model to Domain Specific RAG

Tianjun Zhang, Shishir G. Patil, Naman Jain, Sheng Shen, Matei Zaharia, Ion Stoica, Joseph E. Gonzalez
tianjunz@berkeley.edu, shishirpatil@berkeley.edu
UC Berkeley

Abstract

Pretraining Large Language Models (LLMs) on large corpora of textual data is now a standard paradigm. When using these LLMs for many downstream applications, however, new knowledge (e.g., financial news, or private domain knowledge) introduced after the training cutoff is essential to many emerging applications (Yu et al., 2023; Lazaridou et al., 2022) and is the focus of this work.

This paper studies the following question – How to adapt pre-trained LMs for Retrieval Augmented Generation (RAG) in specialized domains?

Fine-tune the LM based on the retrieved passages i.e., infuse the knowledge in the parameters of the LM. Can cause the model to lose its generic capabilities focus only on specific task ↵

Imp Resources: LangChain, LlamaIndex – Overall frameworks
Lucene-BM25 sparse retriever

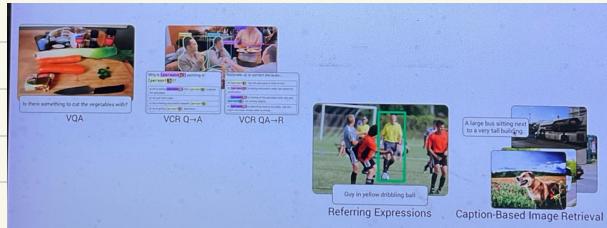
ANNOY, FAISS, ChromaDB – dense embeddings & retrievers
Comprehensive RAGs (CRAG) Benchmark

Lecture 17.1 - Multimodal Models

Vision & language Tasks

* There are various multimodal (vision & language) tasks like

(i) Visual Question Answering - An image with a question based on the image.



(ii) Visual Common Sense Reasoning - Given an image of a situation & a question, figure the correct option from a set of answers.

An extension to this task is given the image, question & an answer, what is the rationale. (Again choose from a set of options)

(iii) Referring Expressions - Given an image & a bounding box, choose a text that describes what is in the bounding box.

(iv) Caption based Image Retrieval - Given a caption & a set of images, retrieve the most relevant image.

* All these methods involve encoding the image & text in some form (sometimes jointly & sometimes individually) & compute similarity at some level.

* Now we need to look at the different kinds of models people use to solve these vision & language tasks.

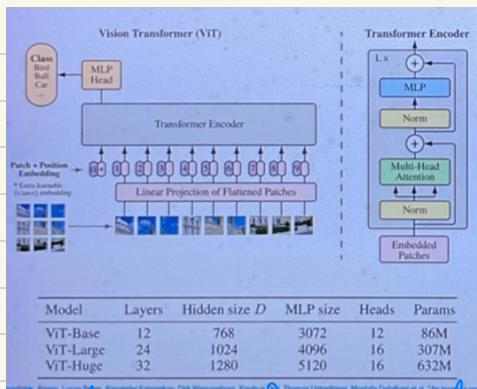
Vision Transformers

- * Until 2020, CNNs were very popular to encode images (and they still are).
- * But Vision Transformers have come up as another popular method

to encode these images.

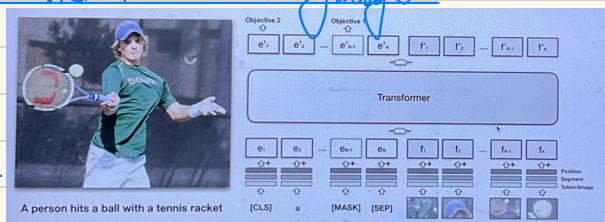
Q) How does it work?

- Split an image into fixed size patches
- Linearly embed each of them
- Add 1-D positional embeddings
- Feed the resulting sequence of vectors (prepended by [CLS] token) to a standard Transformer encoder
- Classification MLP head with 1 hidden layer at pre-training and just a single linear layer at fine-tuning time.
- Pre-train datasets: ImageNet-1k, ImageNet-21k & JFT
- ViT-L/16 means the "Large Variant" with 16x16 input patch size. Smaller patch size \rightarrow Longer sequence length higher latency but possibly higher accuracy
- Match or exceed the accuracy of ResNets on many image classification datasets.



Joint representation model for vision & language

* This model called VisualBERT integrates BERT for NLP and pretrained object proposals systems such as Faster-RCNN.



Q) How to pretrain?

- The idea is to pre-train in such a way that the model is able to link patches in the image with text words.
- Unlike BERT which needs to only understand relationship b/w words, VisualBERT needs to link text to visual patches of images.

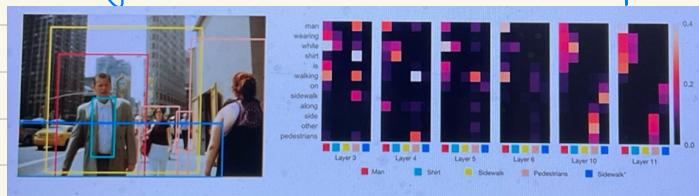
- Should be done preferably in a self-supervised manner with no extra human labeling required.
- People realised that there is a ton of image captioning data available & we can use (image, text) pairs to pretrain this model.
- Leveraged the MS-COCO dataset with 120k images - each with 5 different captions, leading to about 600k (image, text) pairs.
- Used a standard transformer encoder to do this pre-training.
- A [CLS] token followed by input text, a [SEP] token & image pieces go in as input to the transformer.
- Similar to BERT they also mask some text tokens.
- [SEP] token separates text & images.
- Image pieces that go into the transformer are not exactly patches or files. The authors used a Faster-RCNN model to take the image & divide it into patches such that each patch captures a relevant object in the image (like ball, cap, etc.). So, the image patches/tokens being fed into the model are actually objects detected by the faster-RCNN model.
- 3 different kinds of things go in as input at every position:
 - (i) Position embedding
 - (ii) Segment embedding (to identify text or image segment)
 - (iii) Token/Image embedding

- There are 2 different objective functions:
 - (i) Masked Language Modeling (MLM) - Mask some of the text tokens & try to predict the masked word at the output leveraging knowledge from other text tokens & image

patches/tokens. The accuracy of this should be higher than normal BERT MLM because not only do we have the surrounding text but also the image tokens to help predict the missing word.

- (ii) Sentence Image Prediction Task - In a batch of samples, we would have some +ve pairs (where image & caption are related) & some -ve pairs (where image & caption are unrelated). The objective is to figure out if the NLP head is able to predict correctly if it is a +ve or -ve pair.

- * Image alongside shows the attention weights of some selected heads in VisualBERT.

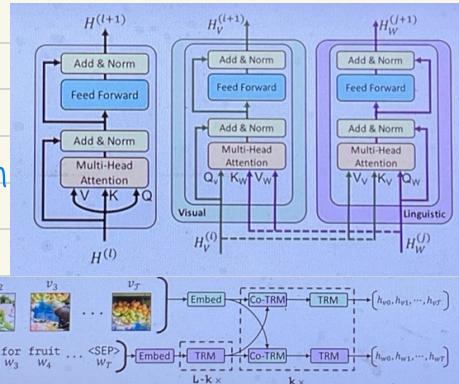


- * In high layers (e.g. the 10th & 11th layer), VisualBERT is capable of implicitly grounding concepts (e.g. "other pedestrians" & "man wearing white shirt")
- * The model also refines its understanding over the layers, incorrectly aligning "man" & "shirt" in the 3rd layer but correcting them in the higher layers.

ViLBERT

- * ViLBERT believes in a 2-tower model as opposed to VisualBERT which is a single-tower model (i.e., the concatenation of text & image modalities happen in the first layer itself)

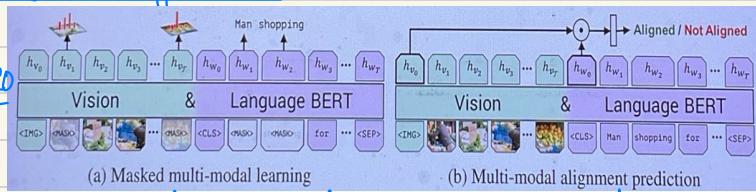
- * ViLBERT processes text & image separately over a few layers.
- * The text stream has significantly



- more processing before interacting with visual features-
- *) The fusion happens later where cross attention layers try to fuse the information across both the pipelines.
 - *) The linguistic stream is initialized with BERT BASE.
 - *) For the visual stream we use Faster R-CNN pre-trained on the Vision Genome Dataset.

VILBERT Training Tasks & Objectives

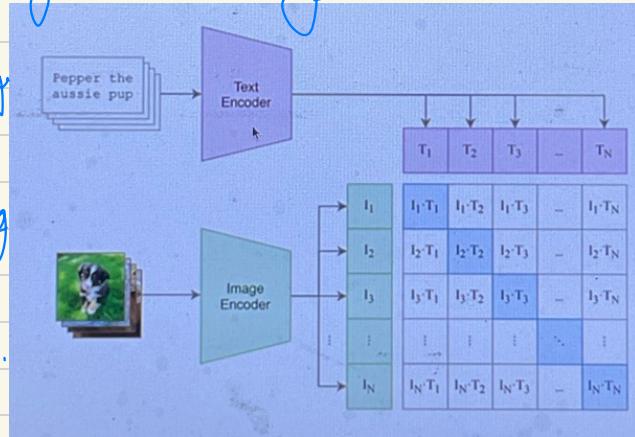
- *) Rather than depending on manually labeled MSCOCO data, they train ViLBERT on Conceptual Captions (~3.3M images) to learn visual grounding. This dataset is obtained in an automated manner by scraping the web - where there are images with a caption below (like Wikipedia), images with good ~~tags~~ tags.
- *) There are 2 pre-training loss functions:
 - (i) Masked Multimodal learning: Here we extend the masking to images as well. The task is to reconstruct image region categories or words for masked inputs given the observed inputs.
 - (ii) Multimodal Alignment Prediction: Predict whether or not the caption correctly describes the image content.



NOTE: In 2019, there weren't many good models/technology to generate image pieces. ViLBERT tried to fix this by generating a distribution over the class labels. With FRCNNs, we know what particular object at a particular position indicates. The objective function is such that it motivates the model to learn the right distribution over the objects at the right position through cross entropy loss & backprop.

CLIP (Contrastive Learning - Image Pre-training)

- * ViLBERT & VisualBERT have image-text pairs for joint modeling to obtain embeddings.
- * Over time the focus has shifted to Contrastive loss based training & CLIP is an example of that.
- * CLIP is also a 2-layer model & has a contrastive loss at the end.
- * We again have (image, text) pairs.



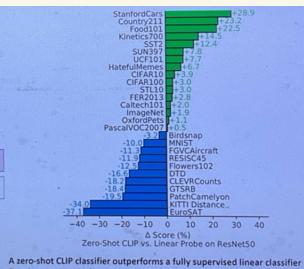
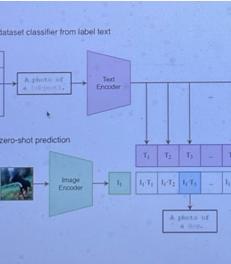
- * We use a pre-trained text encoder to encode the captions & a pre-trained image encoder to encode the images.
- * Text Encoder - 12 layer transformer
- * Image Encoder - Experimented with multiple things
 - 5 ResNets: ResNet-50, ResNet-101
→ RN50x4, RN50x16, RN50x64: Use 4x, 16x & 64x the compute of ResNet-50
- * 3 Vision Transformers (ViT) → ViT-B/32, ViT-B/16, ViT-L/14
- * Pre-trained using a contrastive loss using WebImageText (WIT) 400 M (image, text) pairs.
- * We consider a batch of N instances with N real (image, text) pairs.
- * We take a pool of text & image embeddings & compute the cosine similarity b/w each (image, text) pair.
- * The objective is to maximize the cosine-similarity of the N real pairs & minimize it for the $(N^2 - N)$ incorrect pairs.
- * Tested on 30+ CV tasks like OCR, action recognition in video, geo-localization, etc. where it performed very well.

* In fact 0 shot clip is often
≡ fully supervised baseline.

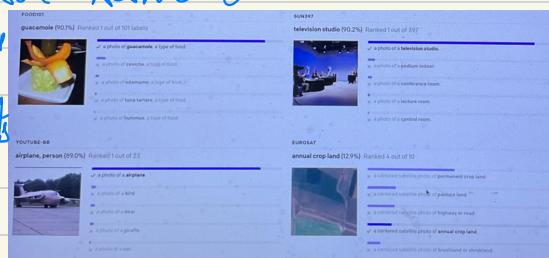
* CLIP performs well even for
0 shot classes i.e., new classes
seen at test time. To see
if a new test time class label

holds good for an image, we take the label, pass it through
the test encoder to get the embedding, pass the image through
the encoder & get its embedding & check the similarity.
So, if try with multiple labels, the label with the highest
score is chosen as the correct label. On this task, CLIP
performed better than a supervised ResNet-50

* The image here shows some examples
of CLIP applied to Food [10],
SUN397, YouTube-8B & EROSAT datasets
& it is found to be very good
in identifying 0 shot classes like
guacamole.



A zero-shot CLIP classifier outperforms a fully supervised linear classifier fitted on ResNet-50 features on 16 datasets, including ImageNet

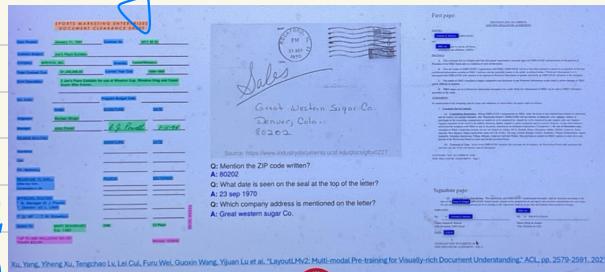


Visually Rich Document Understanding

* Models similar to CLIP have been trained & used for document understanding.

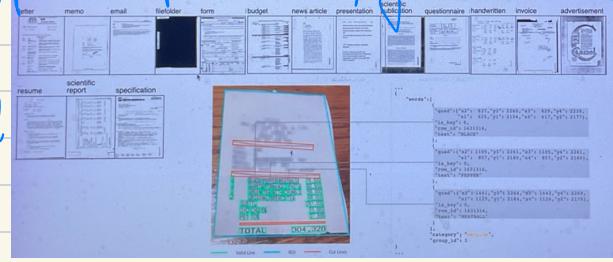
* In this 1st image we see a set of (key, value) pairs highlighted in this clearance sheet. Another image shows a scan of a postcard.

* We use a model called LayoutLM to effectively extract these (key, value) pairs from the document, or do some kind of question answering on the postcard image, ask



Xu, Ying, Yiheng Xu, Tengchao Li, Lei Cui, Furu Wei, Guorui Wang, Yuan Lu et al., "LayoutLM2: Multi-modal Pre-training for Visually-rich Document Understanding," ACL, pp. 2579-2591, 2021.

to extract important keywords/phrases from the legal document
(last image)



* We have a lot of documents on our drives or on the cloud & we could use the model to categorize them into meaningful categories like personal ids, invoices, forms, etc.

* We can also use them to perform some cognition over supermarket receipts.

Layout LMv2 Architecture

* Text : Initialized using UniLMv2.

* ResNeXt-FPN architecture with MaskRCNN backbone of the visual encoder.

* Trained using document specific loss functions.

* Use output feature map ($W=H=7$)

* first take the document & hide some lines.

* They also perform OCR (using the non-hidden version of document) on the document & extract the text (But we have the information of which line is hidden)

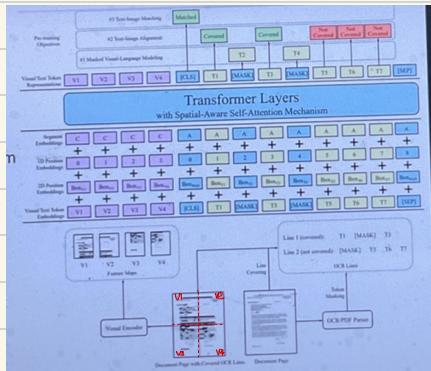
* 4 different things are passed to the transformer:

(i) Segment embeddings (Text or Image differentiation)

(ii) 1-D position embeddings

(iii) 2-D position embeddings (to identify which image patch it belongs to)

Concat(PosEmb2Dx($x_{min}, x_{max}, width$), PosEmb2Dy($y_{min}, y_{max}, height$))



(iv) Text & Visual Embeddings

* Experiment with 2 different models:

Base: 12 layers (200M params)

Large: 24 layers (426 M params)

② 3 pretraining objectives:

- Masked Visual Language Modeling - Predict mask tokens
- Text Image Alignment (TIA) - Predict if a particular token belongs to the covered or non-covered part
- Text Image Matching (TIM) - If the text & image match each other.

* Dataset: 11M scanned docs

Text OCR: Microsoft Read API

Video Tasks

* Text \rightarrow Video Retrieval

- Given a text & collection of videos, find the relevant ones.

* Multiple Choice Video QA

- Given a video, a question & multiple candidate answers, choose the best one

* Action Segmentation/Action Step Localization

- Assign each token (or frame) of a video with one of the pre-defined labels (or steps) to separate meaningful segments of videos.

- Similar to sequence labeling (eg: NER) in NLP

Video CLIP

* Video is nothing but a sequence of image frames.

* If an image is a 2-D matrix, a video is a 3-D cube.

* We could use 3-D CNNs to encode videos or other latent models like transformers. (NOTE: There are many interesting video encoding models like 1D, 3D-Convex, etc. but not covered in this lecture)

A) The idea is the same as before - Contrastive approach to pre-train a unified model for zero-shot video & text understanding using the popular Noise Contrastive Estimation (NCE) loss.

B) How to get these (video, text) pairs?

- We can make use of transcripts & visual information from the video & we can align them. But we need to be cautious about this because text & video frames may not align perfectly. e.g.: This lecture starts with "Today I will be talking about multimodal models" but the actual content comes later; or a recipe which starts with "Let's learn to make 'Chole Bhature' actually shows 'Chole Bhature' much later.

* Thus, we have:

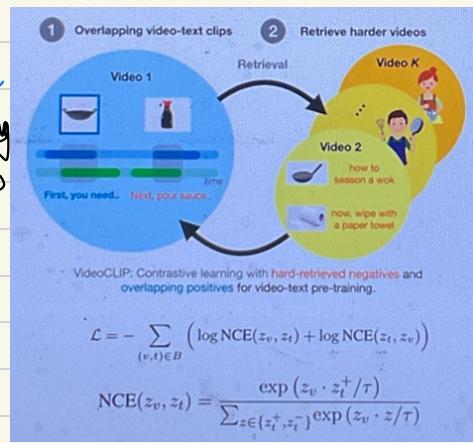
- Loosely temporally overlapping (video, text) pairs, instead of enforcing strict start/end timestamp overlap.
- Model negatives using nearest neighbor retrieval that uses video clusters to form batches with mutually harder videos.

* Use BERT-base-cased for both video (6L) & text (12L)

* Video: frozen pretrained CNN, projected to video tokens using a MLP layer (for the transformer embeddings)

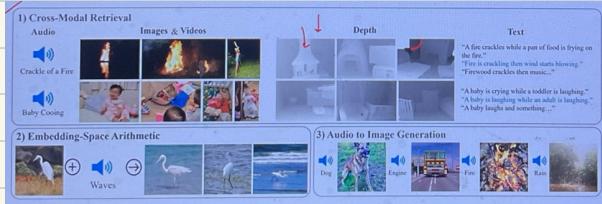
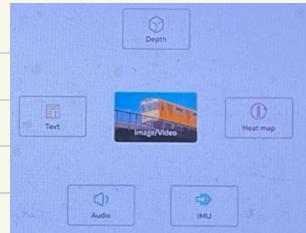
* Average pooling over the sequence of tokens for video & text.

* Pre-training data: HowTo100M dataset



ImageBind

- * Moving towards more modalities.
- * Tries to extend to 6 different modalities - Image, Text, Audio, Depth, Thermal & Inertial Measurement Unit (IMU)
- Depth - how far is each pixel from the camera
- Thermal - Like Infrared Imaging & FLIR images which pick up temperature
- IMU - Like time series sensor data
- * A model that can process all these modalities can solve many cross-modal tasks.
- * Challenge - no dataset across all these modalities. So, even if we want to build an application which uses thermal images alongwith sensor data, we may not have aligned data there.
- * But images tend to bind it all - An image of a beach can remind us of the sound of waves, the texture of sand, a breeze or even inspire a poem.
- * So, instead of trying find pairwise data across all modalities, if we can find data of every other modality with images it could help us solve this problem.
- * ImageBind relies on this. They make use of a set of image data combined with other modality data to train a multimodal model.
- * This helps not only in multimodal understanding but also in generation. eg of tasks include: Cross modal retrieval, Audio-to-image generation, Embedding space arithmetic (i.e., combining info from modalities)



How is ImageBind trained?

*) Datasets

- (video, audio) pairs from

Audioset

- (image, depth) pairs from SUN RGB-D

- (image, thermal) pairs from LLVIP

- (video, IMU) pairs from Ego4D

- (image, text) pairs from Large-Scale web data



*) Model uses a large deep neural net to encode the image & all other modalities.

$\mathbf{q}_i = f(\mathbf{I}_i)$ and $\mathbf{k}_i = g(\mathbf{M}_i)$ where f and g are deep networks

- *) Info NCE loss (Contrastive loss). Symmetric loss $L_{I,M} + L_{M,I}$
- *) ViT-H (30M params), text encoders (302M params) from OpenCLIP (frozen)
- *) Same encoder for images + videos. Treat videos as multi-frame images.

Lecture 17.2 - Multimodal Models - II

*) Multimodal generation can mean a lot of things like text generation, image generation, video generation & multimodal output generation. However, this lecture focuses on text generation with multimodal inputs.

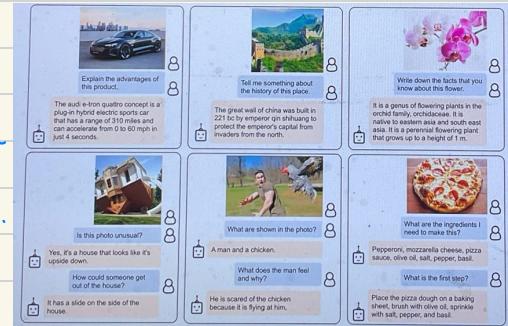
* There are a large no of cases where multimodal inputs lead to text output generation. An obvious one could be image captioning (or video captioning) but there could be much more.

Visual Conversation

* Here we provide a model with an image & some text associated with it to reference for a conversation. eg: advantages of the car.

* Apart from factual conversations there could be logical conversation.

eg: In the 5th photo, what does the man feel, & why? The model can infer that the man is scared as the chicken chases him.



Multimodal ChatBots

* Multimodal chatbots can give detailed answers to queries related to images, videos, etc.

Scene Understanding & Knowledge Grounded Image description

* Could also be used for understanding a scene like the model correctly predicts some kind of natural disaster here.

* Knowledge grounded description is another task where the model can give an output based on internalized (or external)

Multimodal ChatBot



When visiting this peaceful location with a pier extending over a serene lake, surrounded by mountains, there are several things to be cautious about. First, pay attention to the weather conditions. Sudden changes in weather, especially strong winds or rain, can cause the water to become turbulent and dangerous. If you are fishing, be aware of your surroundings and avoid crowding. If there are too many people or excessive weight, this could potentially damage the pier or cause it to collapse. Additionally, be aware of the surrounding wildlife and respect their space; do not harm or disturb them in any way. Finally, if you decide to go fishing, ensure that you adhere to any local fishing regulations and use appropriate fishing techniques to keep the lake ecosystem healthy and maintain the balance of the food chain.

Compose an engaging response, we will be happy to hear from you!

Traveling to this picturesque location with a serene lake and mountains in the background was a refreshing and rejuvenating experience. The pier extending over the calm water provided an excellent vantage point from which to appreciate the breathtaking scenery. As I stood on the pier, I couldn't help but feel a deep connection with nature, taking a moment to savor the peaceful atmosphere.

One of the highlights of my trip was undoubtedly the fishing experience. The area is teeming with life, and I was thrilled to land a steerable catch, despite the modest size of the lake. The joy of fishing in such a beautiful setting was truly rewarding, and it allowed me to take a break from my busy life and truly immerse myself in the tranquility of nature.

While exploring the surrounding area, I came across a family of birds that had built a nest close to the pier, adding to the charm of the location. Interacting with the wildlife and observing their behavior was an enriching experience that I cherish for years to come.

In conclusion, this idyllic destination provided me with an opportunity to unwind and recharge, surrounded by mesmerizing natural beauty and the peace of the water and wildlife. If you're looking for a serene escape from the hustle and bustle of everyday life, this pier overlooking the lake is the perfect place to find solace and reconnect with nature.



What could have happened based on the current scene?

Based on the current scene in the image, it is possible that a hurricane or severe weather event caused significant damage to the buildings and infrastructure in the area. The person walking through the debris-covered area suggests that they are surveying the damage and assessing the situation. The presence of palm trees indicates that the location may be in a tropical or sub-tropical region, which can be more susceptible to hurricanes and other natural disasters.



Introduce me this painting, in detail.

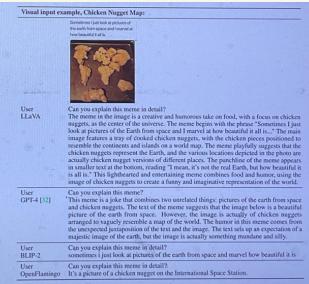
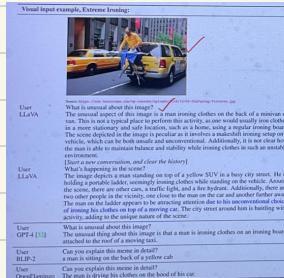
The painting depicts a young girl with a pearl earring, also known as "The Girl with a Pearl Earring" or "Girl with a Pearl Earring". She is wearing a blue dress and has a pearl earring on her left ear. Her hair is pulled back into a bun, and she is looking directly at the viewer. This painting is a famous work of art by Dutch painter Johannes Vermeer, created in 1665. It is considered one of his masterpieces and is widely recognized for its intricate details and realistic portrayal of the young girl's facial expression.

knowledge. For example, here when asked about the painting, it clearly identifies it as "The Girl with the Pearl Earring" by Johannes Vermeer.

Visual Question Answering

* Given an image & a question about the image or something in the image - answer the question.

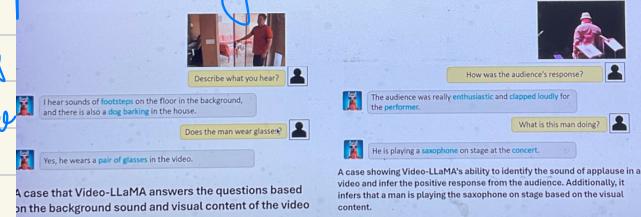
* The example here shows some of the outputs generated by popular multimodal models like GPT-4, LLaVA, BLIP2, OpenFlamingo, etc.



Audio-visual integration perception ability

* Multimodal models don't just deal with image (or video) inputs but other modalities like audio as well.

* In fact, here is an example of how an example of a conversation based on audio-visual input from a video using a Llama model.

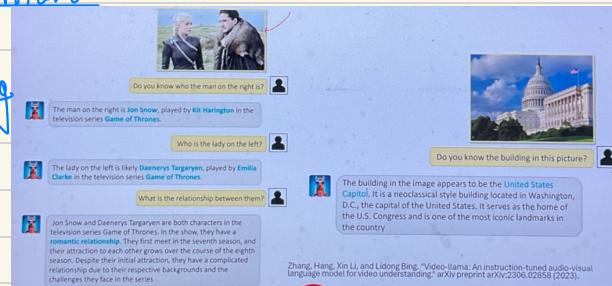


A case showing Video-LLaMA answers the questions based on the background sound and visual content of the video

Common knowledge, concept recognition

* These multimodal models can also hold dialogue in a very interesting & knowledge grounded manner.

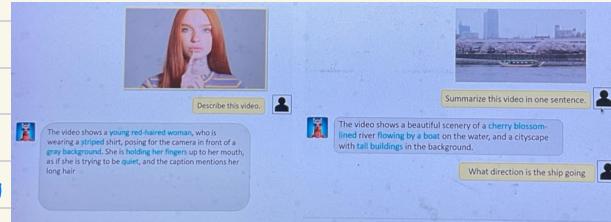
* In this example it is able to identify Jon Snow from the Game of Thrones & the US Capitol Building.



Zhang, Hang, Xiu Li, and Lidong Bing. "Video-llama: An instruction-tuned audio-visual language model for video understanding." arXiv preprint arXiv:2306.02858 (2023).

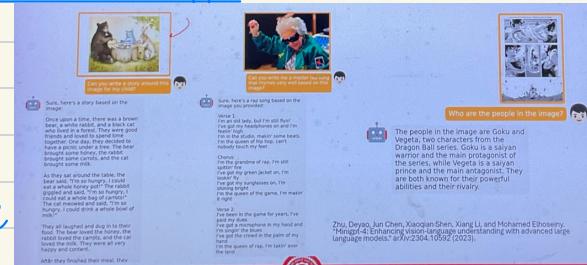
Capturing Temporal Dynamics in Videos

- * Multimodal models are also capable of capturing dynamics like gesture recognition & direction of movement (of the boat) as shown in the image.



Story & song generation: comic understanding

- * Can also be used for story generation given images & other creative tasks like song writing.
- * It can also be used to describe comics & their scenes.

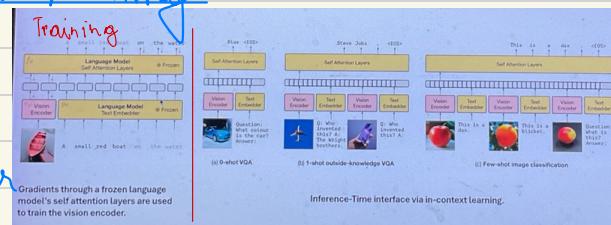


Who are the people in the image?

Zhu, Deyao, Jun Chen, Xiaogang Shen, Xiang Li, and Mohamed Elhoseiny. "Hiragō-4: Enhancing vision-language understanding with advanced large language models." arXiv:2304.10592 (2023).

Frozen: Multimodal few-shot learning

- * Tasks like classification can be done with just multimodal encoders. However, generation task requires a Text decoder as well.



- * Most of these models are trained using image caption data. Like here we have an image of a boat with the caption "A small red boat on the water". We are given the image of the boat & an incomplete caption & our task is to complete it.

- * for images, we have a Vision Encoder for which we could use any ViT or even ResNet models.
- * for text, we have a language encoder like BERT
- * Finally, we need a decoder LM (one that works in an autoregressive

manner)

- * The LM (both encoder & decoder) are frozen when training the model on image caption data. We only end up fine-tuning the Vision Encoder.
- * Gradients through a frozen LM's self-attention layers are used to train the vision encoder.
- * Training is done only on image caption data but at test time we can use it in an In-context learning kind of setup where we can do 0-shot, 1-shot & few-shot learning.
eg: In the figure image we give the model a picture of a car & we give a question as the text & the model responds appropriately (what color is the car? - Blue)
Similarly, we can also do 1-shot Visual Question Answering, few shot image classification (shown in above figure) & many other tasks.
- * The model can also be used in an interleaved image, text manner i.e., input can have multiple images & text chunks & we could use it to generate outputs. eg: like conversing on WhatsApp where we share multiple images & text.

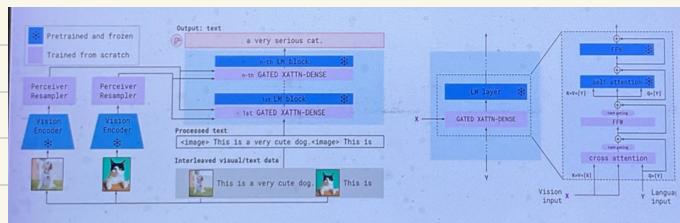
Flamingo

- * Presented at NeurIPS in 2022.

* Has the same basic elements i.e., an image encoder, a text encoder & a text decoder.

* Can also take interleaved text, visual data. eg: If there are 2 images - both are passed through the same Vision encoder (which is pre-trained & frozen)

* Unlike Frozen where the LM is frozen, in Flamingo, the

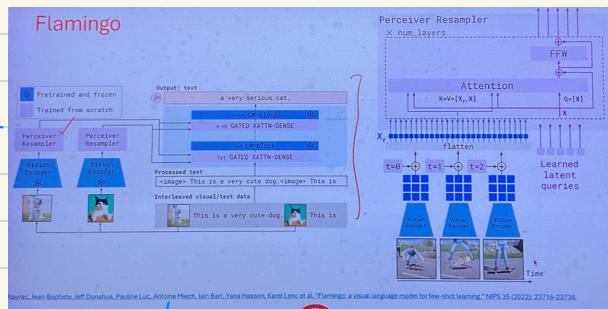


Vision Encoder is frozen.

- * Vision Encoder is a pre-trained ResNet.
- * Depending on the size of Vision Encoder, Text Encoder & text decoder, Flamingo is available in 3 different sizes: 3B, 9B & 20B. There is also an OpenFlamingo model available on Hugging Face.
- * The images go through the Vision encoder & then processed by a Perceiver Resampler.
- * Then we have a Transformer decoder (no separate text encoder in this model).
- * The Transformer Decoder gets a processed text (i.e., text in which images are replaced by a special <image> token) which is fed through multiple layers of a transformer decoder.
- * The decoder uses the inputs coming from the image side for cross attention i.e., the keys & values for the image tokens (X in the above figure) come from the visual encoder whereas the query (Y in above figure) comes from the LM embedder.
- * The rest is almost similar to any other transformer like residual connections, positionwise FFN, self-attention, etc.
- * Since, this is an autoregressive decoder, at any text token, the model should attend to all visual tokens of the image that appeared before. But in Flamingo, the masks are defined in such a way that the model doesn't do that, but only attends to the image just before the token.

- * Dataset - Multimodal Native Web (MSW): image & text dataset.
 - Pairs of image/video & text: ALIGN dataset, LTIP (Long Text & Image Pairs), VTP (Video & Text Pairs)

- * The Perceiver Resampler is capable of processing multiple images (or even videos as frames).
- Positional encodings are added according to the sequence in which the images appear.
 - If we have a model like ViT, the image will be divided into tiles/patches & passed through the vision encoder to get the embeddings before adding the frame position embedding & passing it to the Perceiver Resampler.
 - In the above figures, X_f is just the flattened representation of these patches: i.e., if we have 3 images & each one splits into 8 patches then X_f is the flattened set of 24 embeddings.
 - We may have multiple & variable no of images but we want a fixed size output. To do that we use the Perceiver Resampler.
 - For this, we make use of Learned Latent queries. So if we want a fixed size output (say 32 tokens/embeddings), we make use of these 32 latent queries (X) which are randomly initialized & trained over time.
 - The resampler has multiple layers of cross-attn where we have X_f (representing the current images) & X (representing the learnt latent embeddings).
 - We use X as the queries & X_f as the keys & values and compute attention (not causal/autoregressive; complete attention) followed by pointwise FFN to get a 32 sized output at the end which is then fed further to the next cross att blocks.



Naric, Jean-Baptiste, Jeff Donahue, Pauline Luc, Antoine Miech, Ian Barr, Yana Hasson, Karel Lenc et al. "Flamingo: a visual language model for few-shot learning." NIPS 35 (2022): 23716-23726.

BLIP

- *> Came in 2022 in an ICLR paper.
- *> BLIP can be used both as a multimodal encoder for image/video understanding & also for generation.

- *> Earlier methods were capable of doing only one or the other.

*> Unlike previous models which were trained on small hard-labeled datasets like COCO or noisy web data like Common Crawl (CC), BLIP was trained after a very rigorous cleaning pipeline

(NOTE: Can read more about it in the paper BLIP - Li et al., 2022)

- *> BLIP's architecture can be described as a multimodal mixture of encoder & decoder.

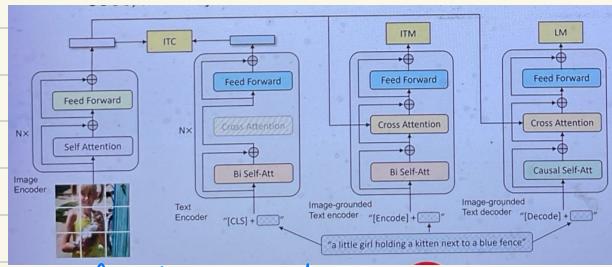
- *> It comprises of these 4 components/towers as shown in the above figure:

- i) Unimodal Image Encoder (ViT-B/16)
- ii) Unimodal Text Encoder (BERT-base)
- iii) Image grounded text encoder [For multimodal understanding]
- iv) Image grounded text decoder [For multimodal generation]

*> First the image gets divided into 16×16 patches & fed through a ViT which gives us an encoding for the image. Model is mostly trained on (image, caption) pairs.

*> The text data is prepended with a [CLS] token, passes it through the text encoder (with bidirectional self-attention and residual connections).

*> We have N layers of both the above & the two are pre-trained together using Image Text Contrastive Loss very much like the CLIP model i.e., in a batch of B samples.



We will have B correct (image, caption) pairs & $B^*B - B$ incorrect pairs & we try to maximize the similarity of the B pairs & minimize it for the $B^*B - B$ pairs.

- * Once we do that we get very good image embeddings from the image encoder which we will use for further computations.
- * The image grounded text encoder takes the same text but with a special [ENCODE] token prepended to it which is initialized randomly & learned via backpropagation.
 - This encoder is very similar to our unimodal text encoder, except that it has a cross-att^{er} layer.
 - This cross-att^{er} layer is fed with inputs from the image encoder model.
 - The model is trained using Image Text Matching Loss. This loss is enabled by providing the model with matching (text,image) pairs & mismatching pairs & asking if to predict a 0 or 1 accordingly.

* Then we have the image grounded text decoder.

- It has a causal self-attention layer.
- Also has a cross att^{er} layer feed from the image encoder.

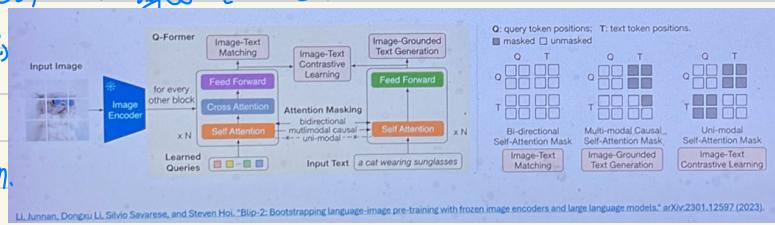
BLIP2

- * Came in 2023 (Li et al., 2023)
 - * Used a Q-Former (similar to perceiver resampler in Flamingo) which gives improved performance.
 - * Very different from the original BLIP architecture.
 - * Similar to BLIP however, the model is capable of both Vision Language Representation Learning (for encoding, classification, etc.)
-
- The diagram illustrates the BLIP2 architecture. It features two main vertical columns separated by a dashed line, representing parallel learning paths.
- Vision-and-Language Representation Learning:** This path starts with an "Image Encoder" (blue diamond) that processes visual input. Its output feeds into a "Q-Former Querying Transformer" (grey rectangle). This transformer also receives "Text Queries" (represented by colored squares) and outputs to a "Large Language Model (LLM)" (blue rectangle).
- Vision-to-Language Generative Learning:** This path starts with a "Large Language Model (LLM)" (blue rectangle) that generates text. An example prompt is shown: "Write a romantic message that goes along this photo." The generated text is then used to "Bootstrapping Pre-trained Image Models" (blue text), which in turn feeds back into the "Image Encoder".

or Vision-to-Language Generative Learning.

- * The former involves bootstrapping pre-trained image models & the latter involves bootstrapping LLMs
- * Pre-training happens in 2 stages:
 - (i) Vision-Language Representation learning stage with frozen language Encoder.
 - (ii) Vision-to-Language generative learning stage with a frozen LLM.
- * In the 1st stage, we have the Q-Former which takes the text, the output of image encoder & a bunch of extra queries (similar to Perceiver Resampler).
- * However, unlike Flamingo's Perceiver Resampler, the Q-Former also takes in the text along with image encoder outputs to compute revised latent representation of the queries in a text sensitive manner (not just image representation).
- * In fact, the Q-Former has an image & text transformer within itself to process both images & text.

- We take the input image, pass it through an image encoder (like ViT) & get the representation.
- The image transformer within the Q-Former has 3 layers per block - self-att^h, cross att^h & FFN.
- The input to this image transformer are the learned query vectors of a fixed size (say 32).
- The image encoder outputs are used in the cross-att^h layer not in every other block (a standard mechanism)
- The text is fed to the standard text transformer encoder which has a self-att^h and an FFN.



Li Junnan, Dongyu Li, Silvio Savarese, and Steven Hoi, "Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models," arXiv:2301.12597 (2023).

- However, the self-attⁿ layers in these transformers are very different.
- The entire Q-former is trained together using 3 different pre-training losses - (i) Image-Text Matching Loss (Bidirectional mask)
 (ii) Image-Text Contrastive Learning Loss (Unimodal self-attⁿ mask)
 (iii) Image-grounded text generation Loss (Multi-modal Causal self-attⁿ mask)

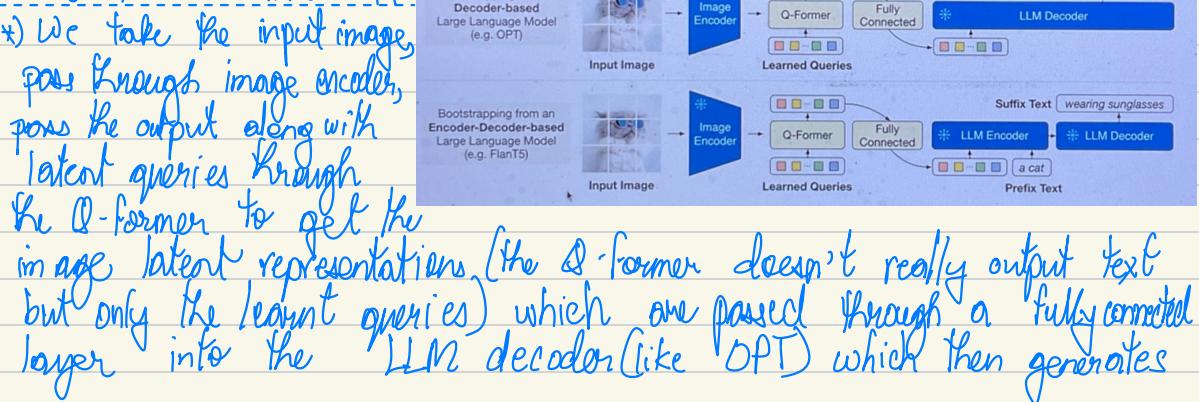
Q) Why the different masks for the different train methods?

A) ITM loss - Bidirectional mask because in this method we train both text & image transformer together & we want to ensure they fuse to the same embedding space so no mask.

ITCL Loss - Here the objective is to run the 2 models separately & maximize the similarity b/w matching pairs & minimize it b/w incorrect pairs (like CLIP). Hence, it has a unimodal mask where query & text tokens are hidden from each other.

ITG Loss - Here the objective is causal text generation hence the query tokens cannot see the text & text tokens have a causal attⁿ mask.

How to use this model?



the text.

* But sometimes we would like to process text as well like a visual QA task. Here we do almost everything similar except the output of the Q-former is fed into the LLM Encoder along with the text prefix (to an encoder-decoder model like Flan T5) & use the decoder to generate the output.

* The pre-trained image encoder:

- ViT-L/14 from CLIP; ViT-g/14 from EVA-CLIP

* frozen language model

OPT-decoder based LLMs; Flan T5 for encoder-decoder based LLMs

* Pre-Training dataset

- Same as BLIP (The rigorously cleaned dataset mentioned in BLIP)
- 12M images from COCO, Visual Genome, CC3M, CC12M, SBU
- 11.5M images from LAION400M dataset.

mPLU61

* With time, people started not just fine-tuning but also instruction-tuning multimodal LLMs. Models like: mPLU61, LLaVa, MiniGPT4

* mPLU61 has a simple architecture

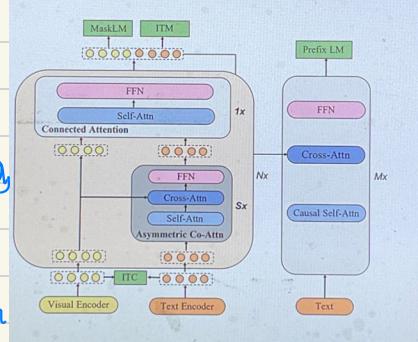
- Visual Encoder, Text Encoder & Text Decoder

- Text Encoder: first 6 layers of BERT base

- Last 6 layers is used for cross modal skip connected network (colored box in the figure)

- 12-layer Transformer for the decoder

- CLIP-ViT for Visual Encoder (ViT-B/16 or ViT-L/14)



Li, Chenliang, Hailiang Xu, Junfeng Tian, Wei Wang, Ming Yan, Bin Bi, Jiabo Yu et al. "mplug: Effective and efficient vision-language learning by cross-modal skip-connections." arXiv:2205.12005 (2022).

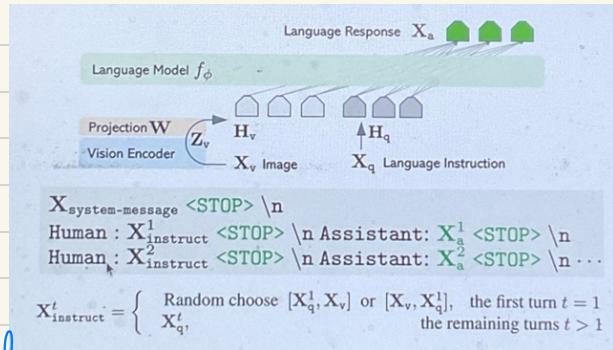
- Cross-modal skipconnected network: Output of visual & text encoder is fed into this block. The text encoder output goes through self-att^b, followed by cross-att^b with visual encoder outputs & then a FFN. The output is then concatenated with the visual encoder output & passed through a self-att^b layer followed by a FFN.
- There are 2 layers of the asymmetric co-att^b followed by 1 layer of the connected att^b & this block is repeated twice which leads to SL from BERT base.
- The output from the encoder (i.e., the above block) is then fed to the decoder for cross-att^b to generate outputs.
- * Pre-trained using 4 loss funcⁿ:
 - (i) Image Text Contrastive Loss
 - (ii) Image Text Matching Loss
 - (iii) Masked Language Modeling
 - (iv) Generative Prefix LM Loss (auto-regressive loss)
- * Pre-training data:
 - 14M images with texts from MS COCO, Visual Genome, Conceptual Captions, Conceptual Captions 12M, SBU captions.

LLaVa

* Came in 2023 (Liu et al, 2023)

* Architecture:

- Pass image through encoder
- Learn a projection layer to project the embeddings to the text/LLM space.
- If we have some text associated with the image, we concat it and give it to the LLM which generates the response.



- * We use the COCO image captioning data for pre-training & language only GPT-4
- * 158k language-image instruction following samples: 58k conversations, 23k detailed description, 77k complex reasoning.
- * All the data generated using few-shot prompting for GPT-4.
- * Pre-training is in two stages:
 - Stage 1: Pre-training for feature alignment
 - filter CC3M to 595k image-text pairs
 - Also, use the 58k conversations which is used later for instruction tuning
 - Visual Encoder & LLM weights are frozen & only the projector is learnt.
 - Loss is defined over the answer tokens & backpropagated
 - Stage 2: fine-tuning end-to-end
 - Keep the visual encoder weights frozen.
 - Update both the pre-trained weights of the projection layer & LLM.
 - Fine-tuned on entire 158k instruction following examples.

* LLM Used: Llama
 Visual Encoder: Pre-trained CLIP ViT-L/14

- * When doing all this, we also give language instructions - like a system-message to tell the model what to do
 e.g.: system-message = A chat between a curious human and an AI assistant. The assistant gives helpful, detailed & polite answers to the human's questions.
- * Even though we don't specifically tune the model for safety or anything but we assume that if we ask the model to be polite then it will be polite & generate

safe answers.

- * Input to the LLM is templated as follows:
 - We give the system-message followed by a <STOP> token & a newline.
 - Then we say "Human:" followed by the instruction & then <STOP> followed by "Assistant:" & then the response & <STOP>.
 - The above block keeps on repeating for the conversation.

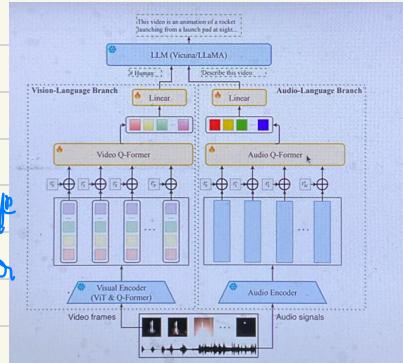
- * The instruction could be (text, image) or (image, text)

NOTE: LLaVa models have been trained with image & text & not with multiple interleaved images.

So, after passing (image, text) pair for the 1st turn, we provide only text in the continuing conversation.

Video LLaMa

- * Used to process videos.
- * Pass the video through a visual encoder (ViT & QFormer combination)
- * Video QFormer: aggregates frame level representations
 - Shares some architecture with Query Transformer (Q-Former) in BLIP-2
- * Audio Encoder: takes in audio segments & creates the spectrogram out of it (audio comprises of frequencies, wavelengths, amplitude, etc. which can be represented as a nice image called a spectrogram). The spectrogram is passed through an audio encoder which is an Audio Spectrogram Transformer from ImageBind
- * The output of both video & audio encoder is then added with positional encodings and passed through the



Zhang, Hang, Xin Li, and Lidong Bing. "Video-llama: An instruction-tuned audio-visual language model for video understanding." arXiv:2306.02858 (2023).

respective Q-Former which gives us some no of queries from both.

* Then we do a linear transform on both the video and the audio to bring it into the text space & then we can combine it with text instructions

*) Uses Vicuna/LLaMa as the LLM.

*) Perform multi-branch cross-modal training

- Vision language

↳ Pre-train on (a) video caption dataset, Webvid-2M, with a video-clip-to-text generation task (b) image caption data, CC3M

↳ Fine-tune on a video-based conversation dataset to execute visual instruction tuning.

↳ image detail description dataset from Mini-GPT-4

↳ image-instruction dataset from LLaVa

↳ video-instruction dataset from Video-Chat.

- Audio language

↳ Pre-train audio-related components on an audio-caption dataset.

MiniGPT-4

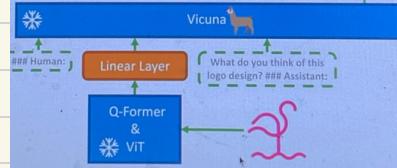
*) Architecture is very similar to LLaVa & Video-LLama.

*) Takes "### Human: " as an input followed by image/video.

*) The image/video is passed through a ViT & Q-Former.

*) The output is projected via a linear layer to the LLM followed by other text tokens to generate the output.

The logo design is simple and minimalist, with a pink line drawing of a flamingo standing on one leg in the water. The design is clean and easy to recognize, making it suitable for use in various contexts such as a logo for a beach resort or a flamingo-themed event. The use of a flamingo as a symbol adds a touch of whimsy and fun to the design, making it memorable and eye-catching. Overall, it's a well-designed logo that effectively communicates the brand's message.



Zhu, Deyao, Jun Chen, Xiaqian Shen, Xiang Li, and Mohamed Elhoseiny. "Minigpt-4: Enhancing vision-language understanding with advanced large language models." arXiv:2304.10592 (2023).

- * Thus, it aligns a frozen visual encoder with a frozen advanced LLM, Vicuna, using a projection layer.
- * Training is in 2 phases:
 - Stage 1 (pre-training)
 - ↳ Vision encoder & LLM are frozen (only train the project^b layer)
 - ↳ Use a combined image captioning dataset that includes images from LAION, Conceptual Captions & SBUs.
 - Stage 2 (fine-tuning)
 - ↳ 3500 detailed image-instruct^b pairs
 - ↳ #####Human <Image feature> <Instruction> #####Assistant:
 - ↳ <Instruction> could be something like "Describe this image in detail" or "Could you describe the contents of the image for me"

Some other Multimodal Models

Meeting notes with multiple speakers (speaker diarization)

Input: How many speakers in this audio and what happened?



Output

- There are four speakers in the audio. It sounds like a project meeting where the project manager, Mark, is introducing himself and asking the team members to introduce themselves and state their roles. After the introduction, the team members additional speakers introduce their roles: Dirk as the marketing expert, Nick as the user interface designer, and Xavier as the industrial designer. They briefly mention their responsibilities for the project.

https://openai.com/index/hello-gpt-4o/

Lecture summarization

- Input
- Here is a video of a presentation on techniques for maximizing the performance of large language models (LLMs) and was delivered by Colin Jarman and John Altonen at the QwenAI DevCon. This was part of Qwen's first developer conference and aimed to provide insights into optimizing LLMs for specific tasks.



https://openai.com/index/hello-gpt-4o/

Outline

• Colin Jarman is the head of QwenAI's solutions practice in Europe, and **John Altonen** is a principal engineer at Qwen. Both are experts in LLMs.

• The presentation highlighted the challenges of optimizing LLMs, emphasizing the need for a multi-faceted solution.

Introduction

• Qwen Jarman is the head of QwenAI's solutions practice in Europe, and **John Altonen** is a principal engineer at Qwen. Both are experts in LLMs.

• The presentation highlighted the challenges of optimizing LLMs, emphasizing the need for a multi-faceted solution.

Challenges of Optimizing LLMs

1. **Identifying Signal from Noise**: It's challenging to identify the exact problem.
2. **Abstract Performance Metrics**: Measuring performance can be difficult.
3. **Choosing the Right Optimization**: It's hard to know which approach to use.

Optimization Flow

• The presenters introduced a framework for optimizing LLMs based on two axes: *Context* vs *Model*, and *Performance* vs *Efficiency*. This is the flow below.

• The framework includes four main steps:

- **“From scratch”**: Starting point for optimization.
- **“Retrieval-Augmented Generation (RAG)”**: For context optimization.
- **“Parameter Tuning”**: For LLM optimization.
- **“Pull it all together”**: Combining all techniques.

Can accurately identify speakers, transcribe audio, answer questions based on audio etc.

Can summarize & even do generation on very long videos like this 15 min video. Can even create slides based on the lecture.

NOTE: Not much is known about the architecture of these models.

Lecture 18.1 LLMs & Tools: Tool Augmentation

* Lecture 18 is in 3 parts

→ Incorporating tools during fine-tuning (Tool Augmentation)

→ Teaching LLMs to use APIs (Function Calling)

↳ eg: for flight search, money transfer, etc.

→ Automating Complex tasks, Multistep tasks (Agentic Workflow)

→ Like how to book a flight → Go to Google, Skyscanner, MMT
→ Identify the flight
→ Check for timings & other things like seats, meals, etc
→ Look for cost effectiveness
→ Book the flight

Limitations of LLMs

1. Inability to access up-to-date information on recent events.

You 22:52
Who is the current president of India?

AI 22:52
As of my last update, the current President of India is Ram Nath Kovind. He has been in office since July 25, 2017. However, please verify from a reliable source as this information might have changed after my last update.

↓
Uses no tools

Searched 3 sites ^ Who is the current prime minister of India?
Q "current prime minister of India October 2024"
bing.com
PM India — Prime Minister of India
narendramodi.gov.in
BYJU'S — Cabinet Ministers of India 2024 [Updated] Know the Portfolios of Council of Ministers...
byjus.com

The current Prime Minister of India is Narendra Modi. He began his third consecutive term on June 6, 2024, following a decisive victory in the general elections. This term builds upon his previous initiatives focusing on economic development, national security, and welfare programs

Interaction with ChatGPT

Used 2 tools:
(i) Calendar (for today's date)
(ii) Search Engine (for documents)

2. A lack of mathematical skills to perform precise calculations

You 00:30 What is the result for 3716521×3 ?

AI 00:30 Hello! I'd be happy to help you with your question. However, I notice that the question contains a typo. The correct calculation is 3716521×3 , not 3716521^3 .

To perform the calculation, we need to multiply 3716521×3 :

 $3716521 \times 3 = 1,114,956,330$
So the result of 3716521×3 is $1,114,956,330$.
Please let me know if you have any other questions or if there's anything else I can help you with!

Interaction with llama-2-13b-chat

↑
Used no tools

What is the result for 3716521×3 ?

The result of $3,716,521 \times 3$ is $11,149,563$.

python
Calculating the product Always show details ⚡ Copy code
result = 3716521 * 3
result

Result
11149563

Interaction with ChatGPT

Correct answer: 11,149,563

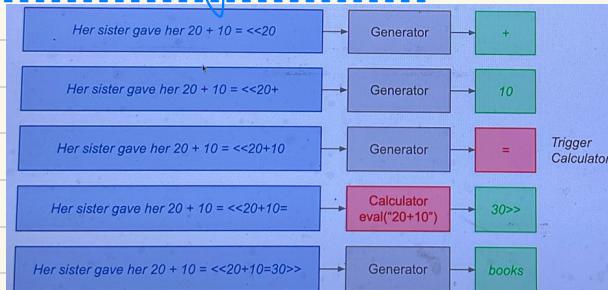
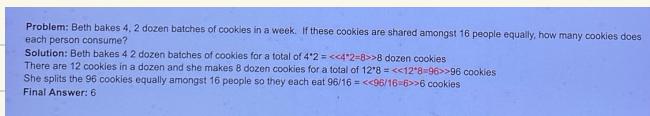
↑
Used a python interpreter to run the calculation

GSM8K : Math Word Problems

- * The main contribution of this paper was creating a dataset of 8.5k high quality linguistically diverse Grade School Math word problems.
- * Besides the final answer, humans created step-by-step solutions for these problems in a way that grade school students would solve them.
- * The annotations like $\ll 4*2=8 \gg$ were not present in the dataset when created. It was created post creating the dataset, in an automated manner.
- * When they fine-tuned GPT (J2SB) model with this, the performance was really poor (below 30% accuracy).
- * So, they decided to augment the dataset with tool calls.
- * For this, they used an automated method to insert possible calculator annotations before any calculations were made (Not all but they annotated a good amount of them).
- * They identified that the most no of places where the model made a mistake was not when reasoning in natural language but rather during executing the command for get the output (like $4*2$, $12*8$, etc.)

How to leverage calculator annotations during inference?

- They used the LM to generate the expression (like $\ll 20+10=$).
- Once, the " $=$ " was generated, they checked if the expression started with " \ll " and if yes, they executed the expression & calculate using a calculator.



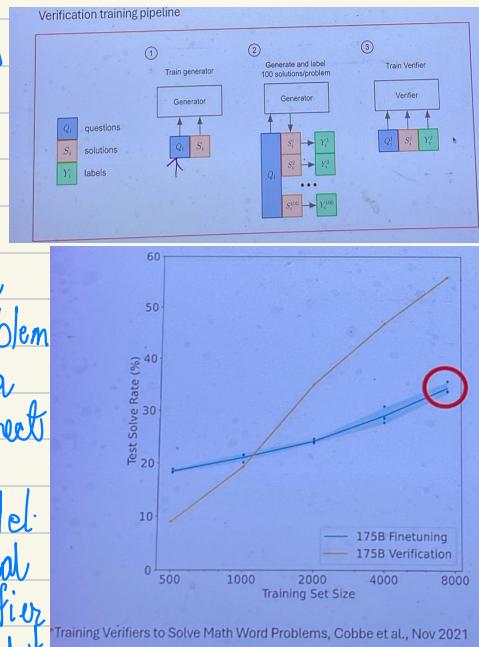
- The result was then added back (along with the closing "»" tag) & generation continued.
- * However, even with this, when they used 8.5K training examples, they achieved results only slightly above 30%.
- * One insight they derived was that instead of greedily decoding (i.e., only picking one high probability completion), they sampled a lot & found that some of these answers were correct. So, even with tools, the LLM was making mistakes.

* The next idea they came up with was to sample a lot & use a verifier to score each sample & pick the one with the highest score.

* Once we train the generator (the 175B model on 8.5K problems), we ask the generator to generate 100 solutions/problem. Then we label the solutions & train a verifier to check if the answer is correct or wrong.

- The verifier is also a GPT-175B model. They take the GPT model, remove the final output projection layer & add a classifier on top of the final token representations & train it to predict right or wrong. The classifier gives out probabilities & we choose the one with the highest scores.

* They found that this method of verification started giving performance improvements with very few examples (as early as 1000).

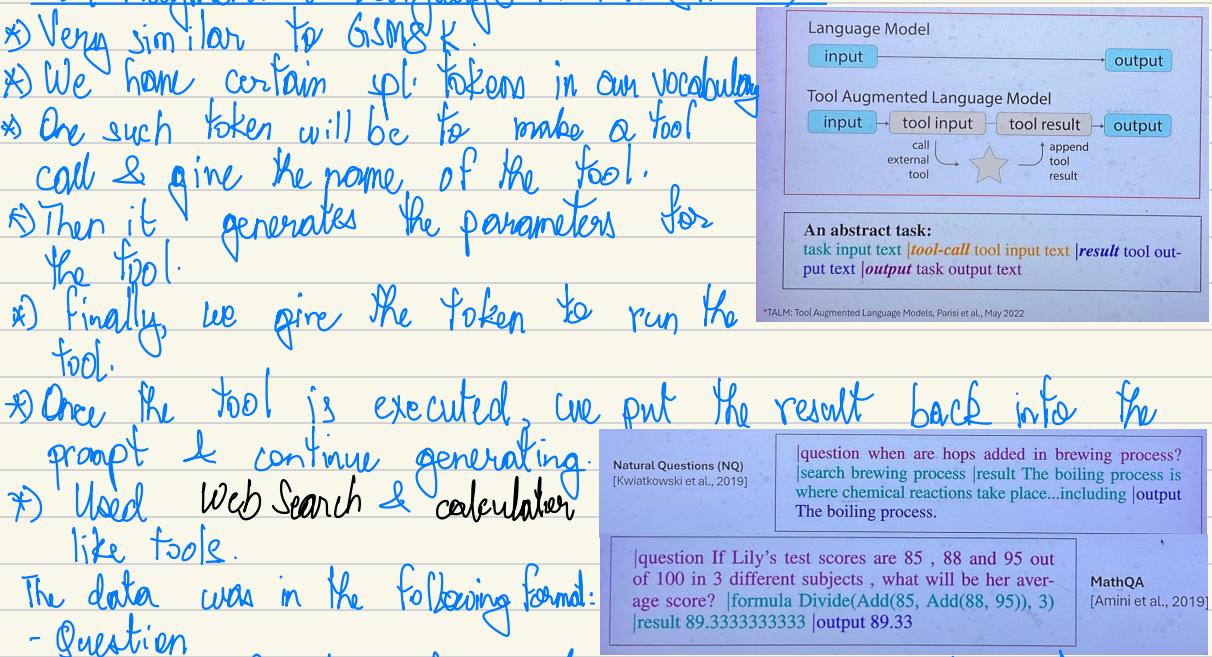


& over the entire 8.5K dataset, they were able to boost the performance from 30% to ~55%.

Summary

- Using tools will help improve performance in certain specific tasks.
- If we have less data, using verifiers can be very helpful instead of just fine-tuning.

Tool Augmented Language Models (TALM)



The data was in the following format:

- Question
- Annotation for tool for use ("search" in above (first) figure)
- Tool parameters ("brewing process" i.e., the search query)
- "Result" token (As soon as we got this token we run the search tool with the query & append the response to the prompt)
- "Output" token
- After all the above, we expect the model to generate the output.

Issue - Creating the dataset is a big challenge!

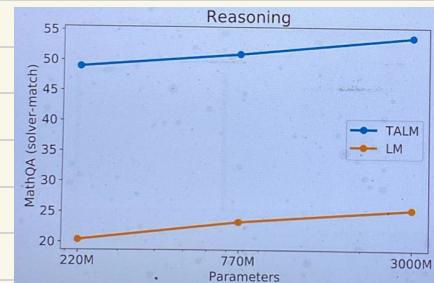
*) Also, the amount of data we need depends upon the "complexity of the task" & how naive the LLM is. e.g.: teaching an LLM to search might not be so difficult but mathematical problems might be more challenging.

Iterative Self-Play Algorithm

- GSM&K annotated the entire training data for tool calling.
- Here, we start with a small set & the algorithm will automatically create more annotations for itself.
- So, we start with D annotations & try to annotate the remaining T automatically where $T \geq D$.
- Alongwith, the input & output, we give the tool annotation & result of tool execution as input to the model.
- We first fine-tune our model with the tool use set.
- Once we do that, the likelihood of it generating the correct answer increases a bit (than the base model)
- Now, we run the model on all data points & whichever for data point we're able to generate the correct answer, we add them to the tool use set.
- The above process is repeated

Algorithm 1 Iterative Self-Play Algorithm.

```
x: task input, y: task output, t: tool input, r: tool output
1:  $T = \{x_i, y_i\}_T$  # task set
2:  $D = \{x_j, t_j, r_j, y_j\}_D$  # tool-use set
3:  $P_\theta \leftarrow$  pretrained LM
4: for  $t \in [0, 1, \dots, R]$  do # self-play rounds
5:    $\theta \leftarrow \operatorname{argmax}_{\theta} \prod_D P_\theta(y_j | x_j, t_j, r_j) P_\theta(t_j | x_j)$ 
6:   for  $x_i, y_i \in T$  do # iterate task set
7:     for  $n \in [0, 1, \dots, N]$  do
8:        $t_n \leftarrow P_\theta(t | x_i)$  # sample tool query
9:        $r_n \leftarrow \text{Tool}(t_n)$  # call tool API
10:       $y_n \leftarrow P_\theta(y | x_i, t_n, r_n)$  # get task output
11:      if  $|y_n - y_i| < th$  then # filter wrong output
12:         $D \leftarrow D \cup \{x_i, t_n, r_n, y_n\}_1$ 
13:      # update tool-use set
14:
```



*) Results show a considerable jump (20-50%) vs. non-augmented LM even for smaller models on MathQA dataset.

*) For NQ \rightarrow ISO demonstrations were used
MathQA \rightarrow $\sim 2K$ demonstrations were used

Summary

- Introduced a general framework (to use any kind of tools)
- Needs slightly less human annotations (generates some using self-play)
- Cannot generalize & still requires annotations

PAL: Program-aided Language Models

*) Here, the main idea was to use a programming language (i.e., a Python interpreter).

*) Tools like calculators have limitations where we can do only certain mathematical operations whereas a Python interpreter can do symbolic problems, logical problems & much more!

*) Created an in-context learning setup where we create answer with what we wish to do and below that we provide the Python equivalent of that. e.g.: Roger started with 5 tennis balls.

*) Doesn't require a huge amount of annotations because LMs are already trained on and quite good at code generation.

*) We don't need to come up with our own syntax either because Python syntax is already defined.

*) Can be extended to beyond simple math problems. e.g.: in the first question in this figure, we need to count only the no. of vegetables. So, the model first identifies the vegetables & then gives their count.

(from the list of given items)

Input:

```
Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?
```

A: `# Roger has 5 tennis balls
tennis_balls = 5
2 cans of tennis balls each
bought_balls = 2 * 3
tennis_balls = tennis_balls + bought_balls`

Q: The bakers at the Beverly Hills Bakery baked 200 loaves of bread on Monday morning. They sold 93 loaves in the morning and 39 loaves in the afternoon. A grocery store returned 6 unsold loaves. How many loaves of bread did they have left?

Model Output:

```
A: The bakers started with 200 loaves
loaves_baked = 200
# They sold 93 in the morning and 39 in the afternoon
loaves_sold_morning = 93
loaves_sold_afternoon = 39
# The grocery store returned 6 loaves
loaves_returned = 6
The answer is
answer = loaves_baked - loaves_sold_morning - loaves_sold_afternoon + loaves_returned
>>> print(answer)
```



```
# Q: I have a chair, two potatoes, a cauliflower, a lettuce head, two tables, or
cabbage, two onions, and three fridges. How many vegetables do I have?
# A: I'm not counting the chair, tables, or fridges
vegetables_to_count = {
    'potato': 2,
    'cauliflower': 1,
    'lettuce_head': 1,
    'table': 2,
    'cabbage': 1,
    'onion': 2
}
print(sum(vegetables_to_count.values()))

# Q: I have a drum, a flute, a clarinet, a violin, four accordions, a piano, a
trombone, and a trumpet. How many musical instruments do I have?
musical_instruments_to_count = {
    'drum': 1,
    'flute': 1,
    'clarinet': 1,
    'violin': 1,
    'accordion': 4,
    'piano': 1,
    'trombone': 1,
    'trumpet': 1
}
print(sum(musical_instruments_to_count.values()))
```

Results

*) Got a bit better results for Chain of Thought (Codex) on GSM8K.

↓
By OpenAI

	GSM8K	GSM-HARD	SVAMP	ASDIV	SINGLEQ	SINGLEOP	ADDSUB	MULTIARITH
DIRECT_Codex	19.7	5.0	69.9	74.0	86.8	93.1	90.9	44.0
COT_UL20B	4.1	-	12.6	16.9	-	-	18.2	10.7
COT_LaMDA-137B	17.1	-	39.9	49.0	-	-	52.9	51.8
COT_Codex	65.6	23.1	74.8	76.9	89.1	91.9	86.0	95.9
COT_PaLM-540B	56.9	-	79.0	73.9	92.3	94.1	91.9	94.7
COT_Minerva-540B	58.8	-	-	-	-	-	-	-
PAL	72.0	61.2	79.4	79.6	96.1	94.6	92.5	99.2

Problem solve rate (%) on mathematical reasoning datasets.

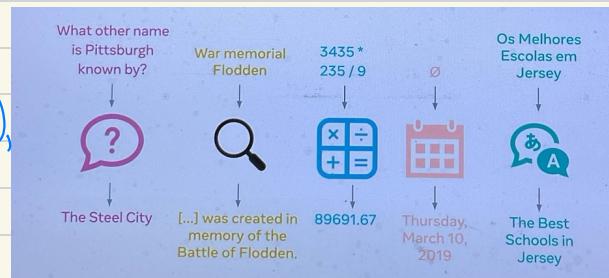
- * But gives much better performance on GSM-Hard compared to CoT (Codex) [Their intuition was that most of the calculations done in GSM&R were single or double digit operations & the LLM had seen so many such calculations during its fine-tuning that it had memorized them without computing. So, they created a new dataset where they replaced the small numbers with much bigger numbers which they called GSM-Hard]
- * Also performed better than contemporary models on other benchmarks.

Summary

- Requires fewer human annotations
- Cannot generalize the tool to use similar tasks (e.g.: the above vegetable example had a different prompt than the other questions. It had the note of what we are not counting. Such examples are needed if we change the question even slightly)

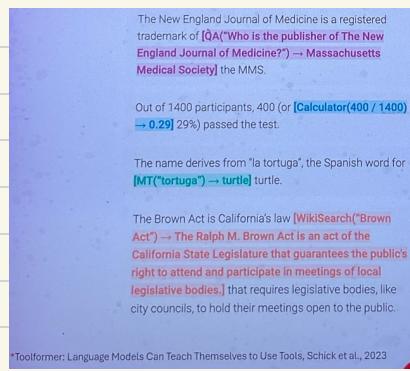
Toolformer

- * Assume that they have access to certain basic tools like - QA system (LM search; factoid), Search on Wikipedia, Calculator, Calendar, Translation app.



- * How to use the tools during runtime - which tool to use, what arguments to pass, how to use the output? (and also generalize in the process i.e., if something unseen comes in the input it should be able to figure out if it can use a tool)
- Went along the same line of thinking as the previous papers did i.e., we need to annotate examples which specifically tells us which APIs to call.

- Tool former autonomously decides
 - ↳ which APIs to call
 - ↳ when to call them
 - ↳ what arguments to pass
 - ↳ how to best incorporate the results into future token prediction

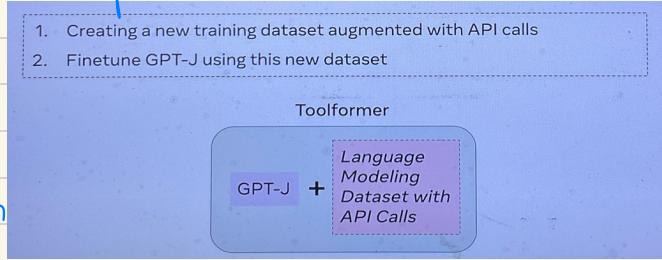


e.g.: "The New England Journal of Medicine is a registered trademark of"- at this point the model needs to figure out that it must call the QA system & with a particular question (as shown in the figure), and once it gets the answer it should be able to use it in generating the next steps
 Some thing goes for other tools as well. e.g.: the 2nd statement "Out of 1400 participants, 400..." is not really a math problem but we want the model to identify that using a calculator is better than directly trying to predict 29%.

*) So, the model ends up learning what the purpose of the tool is rather than just examples of where it can be used.

Q) How does we create this data? (without human intervention - this is the 2nd major contribution of this paper besides what is mentioned above)

A) The following is the pipeline that they used to create this dataset:



- Initially, start with any text corpora (here they use CCNet) & once we execute the pipeline, we will end up with an annotated text corpora based on the tools we have.
 - They create a prompt for each task (all 5 tasks in this case) - give a set of instructions of how to format it & then give some few shot examples.
 - When we use this & sample a lot, it gives us multiple candidates - all of which may not be right. So, we need to figure out what to keep & what to reject.
 - So, we execute the tool & get the answers for each of the candidates that we generate.
 - Then we give the original LM (GPT-J here) the PREFIX & compute the -ve log likelihood of the remaining completion. We do the same for the prefix & the question (w/o answer) & generate the likelihood. We also do the same with the prefix, the question & the answer.
- The min. of L_A & L_B is what the model inherently knows (since the question is just rephrasing the prefix). If we give the actual answer, how much does the likelihood increase (or -ve log likelihood drops) - if -ve log likelihood drops by a lot on giving the answer then it is a very good candidate otherwise it is a bad candidate.
-

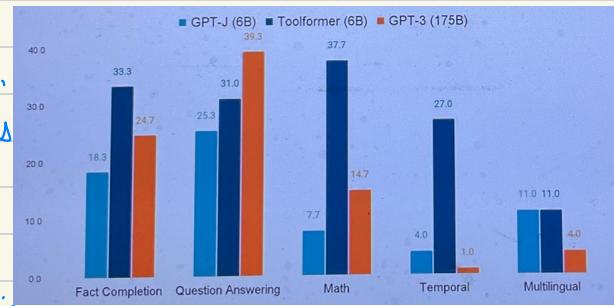
They use this strategy for all the tools & generate a lot of training data.

Results

Shows improvement over GPT-J base.

→ In many cases, it even outperforms the much larger GPT-3 (175B) model.

→ Only exception was QA because QA dataset was based on Wikipedia & GPT-3 had basically memorized Wikipedia.



Summary

- Requires fewer human annotations.
- Can generalize the tool use to similar tasks.

Search Engine as a Tool: Web GPT

(Q) How was the training data created?

A) Created a UI to enter a search query, view the result & add a quote (i.e., save a part of the text from a particular result which can help answer the question). The process was repeated for multiple links.

- Once convinced they had enough information to answer, they ask the LLM to answer the question using all the quotes.
- The tool was then given to humans to use naturally & thus they ended up generating a dataset that looked like figure (b) above.

The screenshot shows a search interface for "How to train the cranes in my neighborhood to bring me gifts?". The results include a link to "How to Make Friends With Cranes | Outside My Window" and a quote from it: "Many animals give gifts to members of their own species but cranes and other birds are unique in that they also give gifts to humans." Below the quote is a note: "Number of quote tokens len: 463". To the right, a generated response starts with "Answer: Inside My Window the crane in my neighborhood to bring me gifts! -". It includes several numbered steps, such as "1. Go outside and make new crane friends with your neighborhood cranes.", and ends with "Done yet? Write an answer.".

(a) Screenshot from the demonstration interface.

(b) Corresponding text given to the model.

[WebGPT: Browser-assisted question-answering with human feedback, Nakano et al., June 2022]

- Then they ask the model to generate the next action which could be something like - click on another link, go to a new page, stop & generate answer, etc.

- This way they try to mimic how a human would use a search engine to come up with an answer.
- Then they used this to create supervised training data, did some rejection sampling, RLHF, etc. to train the model.

Command	Effect
Search <query>	Send <query> to the Bing API and display a search results page
Clicked on link <link ID>	Follow the link with the given ID to a new page
Find in page: <text>	Find the next occurrence of <text> and scroll to it
Quote: <text>	If <text> is found in the current page, add it as a reference
Scrolled down <1, 2, 3>	Scroll down a number of times
Scrolled up <1, 2, 3>	Scroll up a number of times
Top	Scroll to the top of the page
Back	Go to the previous page
End: Answer	End browsing and move to answering phase
End: <Nonsense, Controversial>	End browsing and skip answering phase

Lecture 18.2 - LLMs & Tools: Function Calling

* Tool Augmentation is helpful but we might have custom tools or APIs. Fine-tuning for each tool is very difficult & not scalable either. This is where function calling comes into picture.

Motivation

- * Let's say you're tasked to build a chatbot for your college using LLMs.
- * The goal is to add conversational interface that supports:
 1. institute rules/policy related queries
 2. searching [adding/dropping] courses
 3. querying academic calendar
 4. paying fees

How do you build such a chatbot?

- * It is easy to do with a simple RAG based system but 3,4

are functional.

- *) Most of the usecases for chatbots have a very subset of tasks that involve searching static documents & giving responses. The majority of the tasks (especially for customer support use cases) is transactional - e.g: helping a customer book an airline ticket, sending an email, etc.

What is function calling?

- *) Here is how the architecture of function calling typically works:

- User gives a prompt
- The application (chatbot) has a list of functions that it can use.

- It sends the user query & function list to the LLM.
- LLM identifies what function to use & the function parameters & returns it to the application.
- Application executes the API, gets the results & gives it back to the LLM to generate a response to return to the user.

- *) User only sees the final output.

Q) Why such an architecture?

- A) The functions that we use (like open source libraries, proprietary tools, etc.) get deprecated at a rapid pace. With this kind of architecture we would just need to change the API specification without having to retrain the model again & again.

These are 4 important steps:

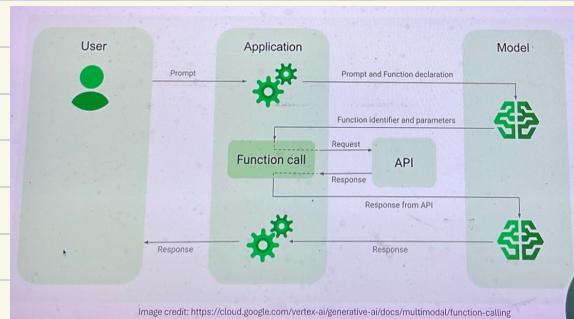
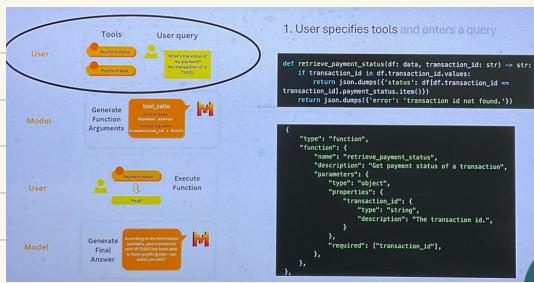


Image credit: <https://cloud.google.com/vertex-ai/generative-ai/docs/multimodal/function-calling>

- 1) User specifies tools & enters a query
- Tools are not necessarily REST APIs only.
 - Could even be a simple python pandas function working with excel sheets in the back.
 - Irrespective of that, we need to create a standard way of specifying what this particular API/function can do.
 - This could include name, a natural language description of what the API/function does, accepted parameters, etc.



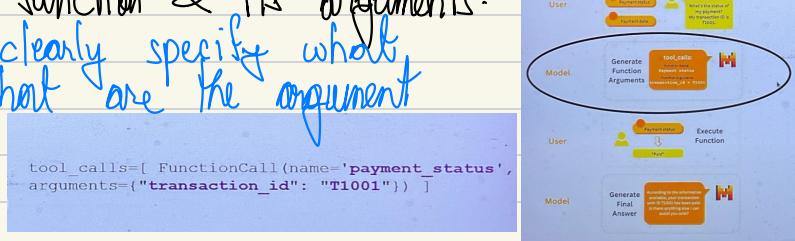
1. User specifies tools and enters a query

```
def get_payment_status(transaction_id: str) -> str:  
    if transaction_id in df_transaction:  
        return json.dumps({'status': df[df.transaction_id == transaction_id].payment_status.item()})  
    return json.dumps({'error': 'transaction id not found.'})
```

```
{
  "type": "function",
  "function": "get_payment_status",
  "description": "Get payment status of a transaction",
  "parameters": "select",
  "properties": {
    "transaction_id": {
      "type": "string",
      "description": "The transaction id."
    },
    "required": ["transaction_id"]
  }
},
```

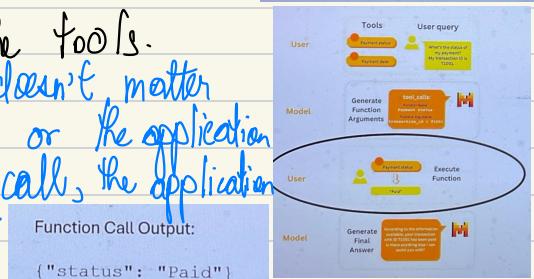
2) Model identifies the function & its arguments.

- LLM output should clearly specify which function to call, what are the argument values to pass.



3) User executes the function to obtain the tools.

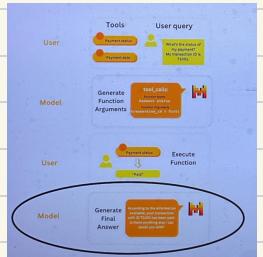
- From the LLM's point of view, it doesn't matter who executes the function - the user or the application.
- The LLM stops by giving the function call, the application (or user) executes it, gets a response & sends it back to the LLM alongwith the query.



4) Model uses the result to generate the answer.

- The LLM uses the query along with the final output of the function call to generate a natural language response.

The status of your transaction with ID T1001 is "Paid". Is there anything else I can assist you with?

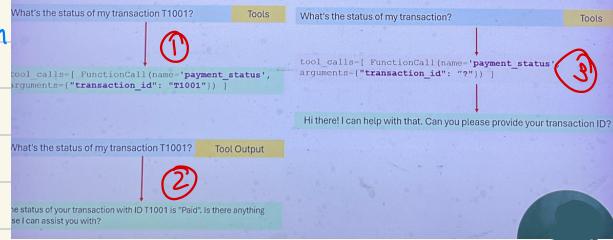


What data do we need to train?

- * The way is to give the conversation so far alongwith the query & the set of tools we have. Then The output should be in the form of a tool call with the parameters.
- * The second thing is that we have the tool output & the conversation so far & the output should be a natural language response based on the output.
- * A third thing is that sometimes a user might just ask what if wants without giving all the required parameters to call the function. In that case, the model should be able to figure out what is missing & request the user for the same in a natural language form.
- * These are the 3 kinds of tasks that the model must learn to perform.
- * There can be other things as well. eg: not enough info to narrow down which function to call, a request which is not at all supported in the function set, etc.

Outline

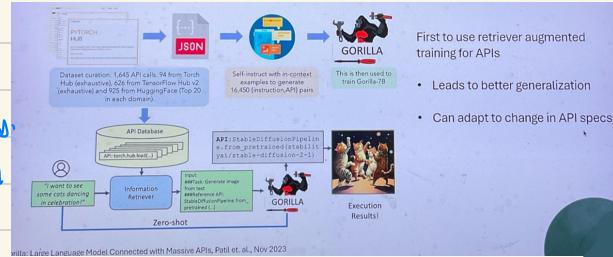
- * We will study the following models:
 - (i) Gorilla
 - (ii) Tool Alpaca
 - (iii) Tool LLM
 - (iv) APIGen
- * Since, OPENAI GPT models are already adept at this task, most of these (especially open) models try to match its performance in benchmarks.



Gorilla: LLM connected with Massive APIe

- x) Synthesised a dataset named APIBench
 - Using model cards in Huggingface Model Hub, Torch Hub & Tensorflow Hub.
 - Each model would be used for a specific purpose. eg: TTS, Image Generation, Time Series data, etc.
- x) fine-tuned Llama-7B with APIBench to create the Gorilla model.
- x) An example data synthesised is shown in the figure here! The user specifies some constraints (like model from Tensorflow Hub) & gives a task. The ssd_mobilenet API from TF Hub is the 'Gold Standard API' to call in this case. The rest of the data helps generate better information.

```
### User: Our customer is a zoo and we want to help them detect movement of different animals. Write a python program in 1 to 2 lines to call API in TensorFlowHub.  
### Assistant:  
<domain>: image-object-detection,  
<api_call>: detector = hub.load('https://tfhub.dev/google/openimages_v4/ssd_mobilenet_v2/1'),  
<api_provider>: TensorFlow Hub,  
<explanation>: This API call loads the pre-trained Single Shot Detector (SSD) MobileNet V2 model from TensorFlow Hub, which has been trained on the Open Images v4 dataset. This model is used for object detection in images. Once loaded, you can pass an image (and an optional score threshold for detection confidence) to the detector to get back the detected objects along with their bounding boxes, class labels, and associated confidence scores.  
<code>: import tensorflow as tf; import tensorflow_hub as hub; # Load the SSD MobileNet V2 model; detector = hub.load('https://tfhub.dev/google/openimages_v4/ssd_mobilenet_v2/1'); # Use the model to detect objects in an image (assuming 'image' is a TensorFlow tensor representing your image); result = detector(image, score_threshold=0.5)
```



- x) Curated model cards that were specified in detail (on the Hubs) & match the required specifications.
- x) Go through 1645 model cards from these 3 websites & they converted them all to a common format.
- x) Used self-instruct with in-context examples to generate more (16450 (instruction, API) pairs) examples.
- x) Then they fine-tune the model.
- x) When curating the dataset, they also created a database with all the model specifications (Because we have 1645 API calls with lengthy descriptions so can't give it all in

the prompt. We need a retrieval mechanism which fetches a subset of relevant APIs from the database & give it to the LM which alongwith the input question is able to generate the API call which when executed gives the result.

*) They suggested not to let the model memorize all the APIs but rather provide them during runtime so that we don't overfit on those specifications & generalise better

Evaluation

*) Came up with an idea to evaluate these function calling models.

*) Most of these models output function calls, programs, etc.

*) They borrow from the programming language paradigm where to compare two programs they create an abstract syntax tree (AST).

*) They also came up with a way to figure out where the model fails.

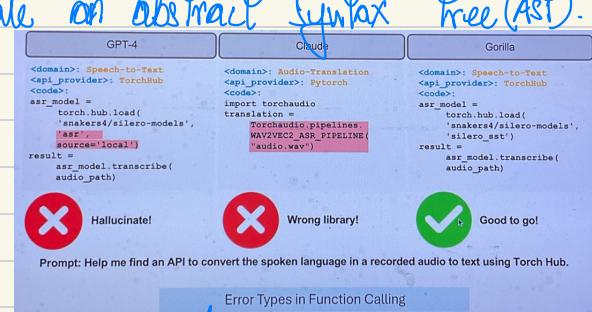
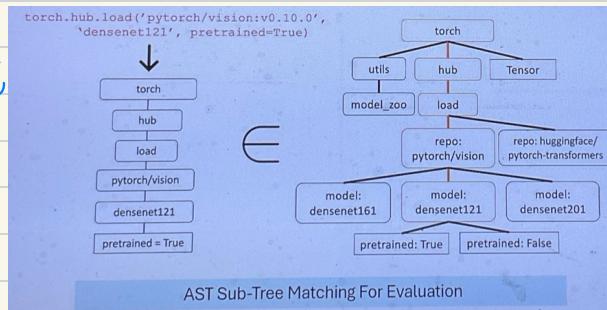
*) Divided the failure cases into 2 parts:

(i) Hallucinating i.e., giving out API names that don't exist at

all or cooking up arguments that are not a part of the function specification.

(ii) Wrong library i.e., it gives an acceptable function that is part of the input but not the correct one

- In the 2nd case the failure is because the model's understanding was incorrect (maybe the description was confusing), but if the model hallucinates then it is an inherent model problem.



*) This helped them narrow down in consecutive iterations of model training.

Results

- *) They have 2 different experimental setups - one where they provide retrieval results based on a real retriever system (i.e., the correct API may not be present in the retrieved results every time); the 'Oracle' scenario is where the correct API is always present in the input.
- *) Rare LLaMa model which was not tuned for function calling had a big hallucination issue whereas models trained for function calling have very few hallucinations (but more semantic errors i.e., picking the wrong API, etc.)
- *) The hallucination & error rate of Gorilla was much lower than the other models on average.

LLM (retriever)	TorchHub			HuggingFace			TensorFlow Hub		
	overall ↑	hallo ↓	err ↓	overall ↑	hallo ↓	err ↓	overall ↑	hallo ↓	err ↓
LLAMA (GPT-Index)	14.51	75.8	9.67	10.18	75.66	14.20	15.62	77.66	6.71
GPT-3.5 (GPT-Index)	60.21	1.61	38.17	29.05	7.85	44.80	65.59	3.79	30.50
GPT-4 (GPT-Index)	59.13	1.07	39.78	44.55	11.18	44.25	43.94	31.53	24.52
Claude (GPT-Index)	60.21	3.76	36.02	41.37	18.81	39.82	55.62	16.20	28.17
Gorilla (GPT-Index)	61.82	0	38.17	47.46	8.19	44.36	64.96	2.33	32.70
LLAMA (Oracle)	16.12	79.03	4.83	17.70	77.10	5.20	12.55	87.00	0.43
GPT-3.5 (Oracle)	66.31	1.60	32.08	89.71	6.64	3.65	95.03	0.29	4.67
GPT-4 (Oracle)	66.12	0.53	33.33	85.07	10.62	4.31	55.91	37.95	6.13
Claude (Oracle)	63.44	3.76	32.79	77.21	19.58	3.21	74.74	21.60	3.64
Gorilla (Oracle)	67.20	0	32.79	91.26	7.08	1.66	94.16	1.89	3.94

BFCL Leaderboard										
Rank #	Overall Acc	Model	Latency (s)			Single Turn			Multi Turn	
			Cost (\$)	Mean	Non-live (AST) ▶ AST Summary	Non-live (Exe) ▶ Exec Summary	Live (AST) ▶ Overall Acc	Multi Turn Multi turn ▶ Overall Acc		
1	59.43	GPT-4-turbo-2024-04-09 (FC)	34.98	2.87	82.63	83.8	73.39	21.62		
2	55.28	GPT-4o-2024-09-06 (FC)	8.37	1.31	85.52	82.96	71.79	21.25		
3	55.11	xLAM-Ba2Bn (FC)	N/A	2.64	80.75	88.32	71.81	15.62		
4	58.43	GPT-4o-min-2024-07-18 (FC)	0.95	1.67	82.03	83.8	67.53	25.75		
5	57.61	xLAM-Ba7Bn (FC)	N/A	1.21	84.44	85.39	71.97	13.75		

Summary

- *) Low diversity in APIs
- *) Single turn dialogue (very simple; give input & output is what function to call)

Tool Alpaca

- *) Synthesised a dataset with 3000 examples

- Using 400 real-world inspired tools spanning 50 categories.

- x) fine-tuned Vicuna-7B (& 13B) model to create ToolAlpaca-7B (& 13B)
- x) Only used the tool name & introduction (or small description) by scraping the web. Rest everything was synthesised.
- x) Given the tool name & short description, they ended up creating a longer description of what the tool could be capable of & post that the names of functions/APIs available within the tool & the detailed specifications (parameters & other things)
- x) Created a nice conversation for an end-to-end function calling model training. e.g.: here the model needs to find what holidays are there next year but is unsure about what year it is so it asks the user. Then when it gets that information it can call the getHolidays function with the specified parameters & get the holidays & send the appropriate response to the user.

The screenshot shows the "Public Holidays" section of an API documentation page. It includes an icon of a person at a computer, a title, and several code snippets illustrating how to use the API. The snippets show examples for getting a list of holidays for a country, retrieving details for a specific holiday, and searching for holidays by keyword. The code uses HTTP methods like GET and parameters like 'country' and 'year'.

This screenshot shows a conversational interface between a user and a bot. The user asks for a list of holidays in Japan for the next year. The bot responds with a message asking for the year. The user then specifies the year as 2024. The bot then asks if the user wants to plan a trip to Japan. The user replies that they are planning a trip to Japan next year. The bot then asks if the user needs a list of holidays for 2024. The user replies that they need a list of holidays in Japan for 2024. The bot then sends a message with the status code 200 and the response, which includes the list of holidays for New Year's Day in Japan for 2024.

Evaluation

- x) Evaluated on tools not seen during training.
- x) Performed much better than the non-function calling version of Vicuna & comparable to GPT-3.5 (but still some difference)

Model	Procedure	Simulated Tools			Real-world APIs		
		Response	Overall	Human	Procedure	Response	Overall
GPT-3.5	77.0	85.0	75.0	79.0	75.4	80.7	72.8
Vicuna-7B	19.0	21.0	17.0	16.0	7.9	11.4	7.9
ToolAlpaca-7B	63.0	69.0	60.0	73.0	63.2	57.9	55.3
Vicuna-13B	17.0	31.0	16.0	25.0	13.2	16.7	12.3
ToolAlpaca-13B	70.0	73.0	70.0	75.0	66.7	67.5	61.4

Evaluation results on unseen simulated tools and real-world APIs

Summary

- * High diversity (400 real world inspired APIs)
- * Multi-turn dialogs with question generation & response generation.

Tool LLM

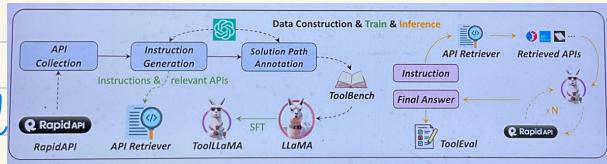
- * Synthesised a dataset named ToolBench.
 - Scrapped 16,464 real-world REST APIs from RapidAPI Hub

- * Fine-tuned Llama-7B with ToolBench to create ToolLlama model.

- * Built conversations on top of real-world API as compared to fixed format APIs generated in ToolAlpaca. More likely to

face bottlenecks/challenges faced in the real world, here because not all APIs follow the same rules, format, etc.

- * So, they generated dialogs in multiple phases -
 - first they generated only the 1st instruction i.e., the first instruction the user might start with.
 - By taking a small subset of APIs, figure out using an LLM (specifically ChatGPT) what sort of queries can you answer. They got some instructions & identified what subset of APIs would be used for the conversation.
 - They soon realised once they created a few utterances is that more APIs were not linked right or didn't have enough information.
(e.g.: use generic variable names like x, y ; missing parameters in descriptions)
 - So, they came up with an approach of **Solution Path Annotation**
 - They used it to create the entire dataset & fine-tune Llama7B on it to give ToolLlama.



- Another good thing was that as a by-product they ended up training a very good API retriever (earlier retrievers were actually built with some other applications in mind. e.g. BERT)

* During inference we:

- Get the instruction
- Use the API retriever to get a list of relevant APIs
- Identify what API to use & call it
- Give the final answer

Synthetic Data Generation

* Before this we used to start with an utterance, generate agent response, run the identified tool & finish the dialog. (if single path)

* But this was not working anymore because the APIs were not as clean & well structured (or properly documented) and that messed up the conversation.

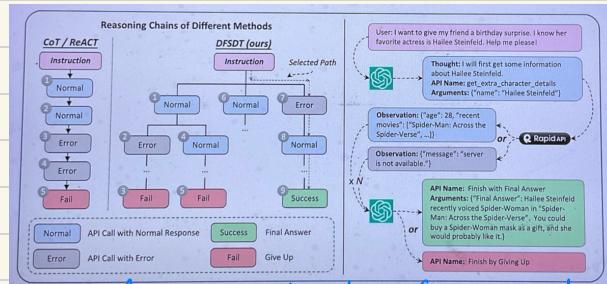
* So they figured out a depth-first tree search where they set some rules of what was a 'normal' execution & if an API executed correctly or not, if the path is meaningful or not.

* Once they failed, they backtracked to the previous point of 'normal' execution & explore a different path.

* Doing this, they were able to some time & get at least one successful path.

* At every stage, they leveraged ChatGPT to make a final call about whether they should stop & track back or what to produce next.

* The main idea is - if a tool fails, try to use another



tool, maybe the argument was wrong - try generating other samples for the same API, etc.

Results

- * Tested it on the unknown APIBench dataset to see how it performs on unknown tools.
- * Identified that with their version of the retriever & their model), the model did a better job on 2 subsets (HF & Torch) & almost equal to Gorilla on the 3rd (TensorHub).
 - TooILLaMA + Oracle
 - Gorilla-ZS + BM25
 - Gorilla-RS + BM25
- This showed the model had generalised well. (However, the retriever they trained & used was much better than BM25 so we can't just say that ToolLlama is better than Gorilla)
- For a more level field they also tried with the Oracle retriever & in this case Gorilla performed somewhat better (although ToolLlama was not very far behind considering Gorilla was trained on that model whereas ToolLlama wasn't)
- But the generalisation capabilities of this model made it a good starting point for using a LLM with a chat application & our own APIs.

Method	HuggingFace		TorchHub		TensorHub	
	Hallu. (↓)	AST (↑)	Hallu. (↓)	AST (↑)	Hallu. (↓)	AST (↑)
TooILLaMA + Oracle	10.60	16.77	15.70	51.16	6.48	40.59
Gorilla-ZS + BM25	46.90	10.51	17.20	44.62	20.58	34.31
Gorilla-RS + BM25	6.42	15.71	5.91	50.00	2.77	41.90
TooILLaMA + Oracle	8.66	88.80	14.12	85.88	7.44	88.62
Gorilla-ZS + Oracle	52.88	44.36	39.25	59.14	12.99	83.21
Gorilla-RS + Oracle	6.97	89.27	6.99	93.01	2.04	94.16

OOD generalization experiments on APIBench

Summary

- * High diversity dataset (16k real world APIs from RapidAPI Hub)
- * Multi-turn dialogue (but restricted to how ChatGPT can interact in many ways; so the dialogue generated were not very clean as we would want to train our model on)
- * Has single & multi-tool setup (in the same conversation)

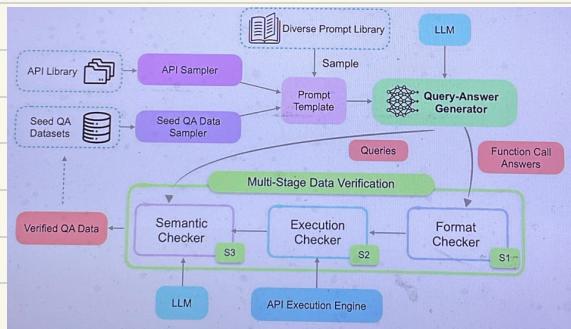
APIGen

→ fully open source including
for commercial purposes

- * Synthesised a dataset named xlam-function-calling-60k
 - Used only 3673 real world REST APIs from Rapid API Hub (Scraped for ToolBench)
- * Fine-tuned DeepLick Coder - 13B instruct (and 7B) model with xlam-function-calling-60k to create the xlam-1B (fc) (and 7B) model
- * Also trained Mistral based models.

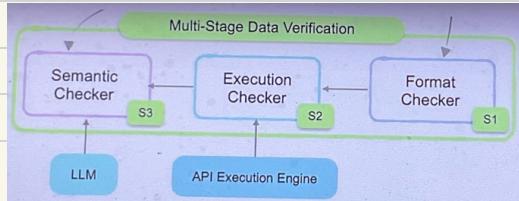
How to generate data?

- * The multi-turn data created by ToolBench was not really usable.
- * To create clean conversations, they started filtering not just on API execution but more at a dialog level.
- * They have a detailed verification stage.
- * Used the same ToolBench RapidAPI dataset but a much more reduced set of 3673 APIs that were filtered based on certain rules.
- * Once they had this clean data, they had seed QA datasets which would serve as few shot examples to generate a lot more data points.
- * Once generated, they used their multistep verification process to get clean data.



What are the verification stages?

- * Firefly, similar to Gorilla they created a clean format which is uniform



for all the tools.

*) Also, created uniform formats for all the APIs & function calls.

*) They also made the output from the LLM follow a json format (By this time, LLMs had been trained to give structured outputs using schemas)

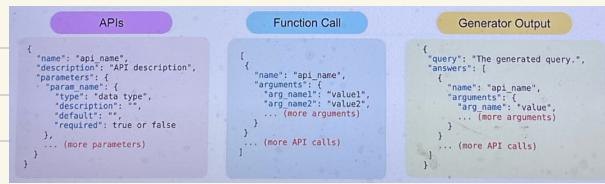
*) Structured outputs/inputs are much easier to clean up. e.g.: if we hallucinate in some part then we can actually identify what was wrong & correct only that part.

Execution Checker

- functions are executed using appropriate backend
 - Python functions are directly imported & executed in a separate subprocess
 - REST APIs are called to obtain results (e.g. fastapi code)
- Unsuccessful executions are filtered out (Done while generation is happening: similar to Tool LLM)

Semantic Checker

- Does the final answer semantically align with the query's objective?
- Query-Answer pairs that execute successfully can produce meaningless results due to:
 - 1) Infeasible queries
 - 2) Incorrect arguments
- Model is trained to ensure that the API gives the correct output format & data so it tries to use that information even when it is meaningless.
- Used LLM as a judge to look at the last responses



generated & see if it looks like a cohesive conversation.
 This helped filter out all unnatural & junk dialogues which were then removed.

* This analysis table shows different models they used for generating data.

- Generated 40K examples from each model.

- DeepSeek-V2-Chat(236B) gave a lot more good data than any other format.

- Also shows exactly where (i.e., at which verification stage) it failed.

NOTE: This doesn't mean the data is perfect. It's just that it passed those 3 verification stages (but there may still be other issues).

Model	Verified Data	Fail Format	Fail Execution	Fail Semantic	Pass Rate
DeepSeek-Coder-33B-Inst	13,769	4,311	15,496	6,424	34.42%
Mixtral-8x7B-Inst	15,385	3,311	12,341	7,963	38.46%
Mixtral-8x22B-Inst	26,384	1,680	5,073	6,863	65.96%
DeepSeek-V2-Chat (236B)	33,659	817	3,359	2,165	84.15%

Filtering statistics for the generated datasets using different base LLMs.

Results

* ILM trained on this dataset achieved very good results, comparable to much larger models.

BFCL Leaderboard								
Rank	Overall Acc	Model	Cost (\$)	Latency (s)		Single Turn		Multi Turn Overall Acc
				Mean	AST Summary	Non-live (AST)	Exec Summary	
1	51.41	torche-2024-04-09 (FC)	\$4.59	2.17	81.65	83.8	73.39	21.42
2	51.23	GPT-4e-2024-08-06 (FC)	6.37	1.33	81.52	82.36	71.79	21.25
3	51.11	ALAH-Ba2B+ (FC)	N/A	2.64	80.75	83.32	72.81	15.62
4	51.43	minn-2023-07-18 (FC)	6.56	1.43	82.43	82.8	67.33	25.75
5	51.94	ALAH-Ba2B++ (FC)	N/A	2.22	80.44	83.89	71.97	15.75

Summary

- * High diversity dataset (3K high quality real world APIs from RapidAPI hub)
- * Has quality multi turn dialogue - thanks to 3-stage filtering.
- * Has single tool setup & multi tool setup along with its parallel variants.

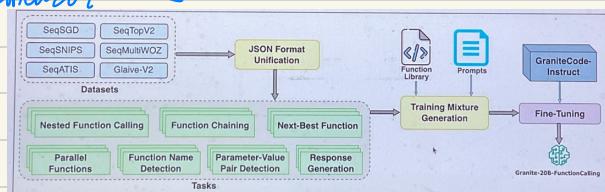
Granite function-calling Model

- *) Created by IBM.
- *) Issues with existing open models:
 - 1) Openness: The best performing models are proprietary & the ones that have open licenses (eg: Gibrilla) are trained using data generated from OpenAI models.
 - 2) Generalizability: Even though the datasets are generated using diverse sets of APIs (eg: RapidAPI), some research (Barry et al., 2024) has shown that models trained on these datasets have difficulty in generalizing to out-of-domain datasets.
 - 3) Granular Tasks: function calling encompasses multiple granular sub-tasks such as function name detection, slot-filling, and detecting the ordered sequence of functions needed to be called. Existing models trained to perform function calling lack the ability to handle these granular tasks independently.

*) Granite

- Synthesise data using models which provide open license
- repurposed existing datasets with permissible license and added new synthesised datasets for better generalisation.
- enabled support for granular tasks.

- *) More developer friendly because of granularity (eg: you just want to generate a response for some json you obtained from outside, or get a list of functions you are likely to call, etc.)



lecture 18.3 - Agentic Workflow

Motivation

Q1) How do we work on a programming assignments?

- A) While an LLM would directly give you the answer, we would think about the problem deeply & maybe divide it into sub-tasks & solve each one at a time & then combine. If you fail or get stuck somewhere, you start debugging by asking an expert, your professor, peers, look at the internet, etc.

Q2) How do we write research papers?

This is a more complex task and one of the biggest hurdles is that we don't know if we did a good job unless we get it peer reviewed. This is an extremely long cycle compared to solving programming assignments. In research, we might write an introduction & design our experiments according to that. If it doesn't work, we may change the introduction to fit the story that we want to convey through our experiments.

- * Almost all real world applications are of this type. We may not be the only actor in the process.
- * We do not always have the entire problem description in one-go & we have the ability to solve it in one-shot.
- * If we want to use LLMs for real world applications, they should be able to work in the same way as humans do.

This way of working is called Agentic flow.

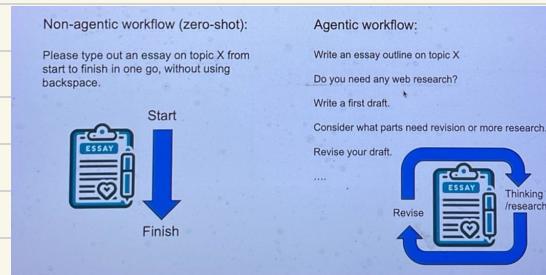
Agentic Workflow

* Typically, all LLMs go from start to finish in one direction & give the answer.

e.g.: In school exams, if we had to write an essay, we would make up a structure in our head

(similar to how LLMs would in diff layers in the transformer) & then spit out one word at a time and finish the essay.

This is Non-Agentic Workflow



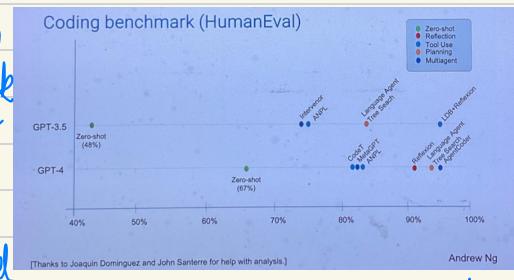
* But writing a research paper is also similar to writing an essay, but we do a lot more. Write a rough draft, write an outline, write some experiment results, correct some things, re-write some parts, etc.

Similarly, for the essay example one could also - search the web, come up with an initial draft, identify what needs more work, revise & so on.

This is Agentic Workflow where we spend a lot more time in iteratively fine-tuning either the entire solution or some parts of it.

* Here is example from a lecture by Andrew Ng. HumanEval is a coding benchmark where we are given a problem statement with a description & we will have to code the meat of the function.

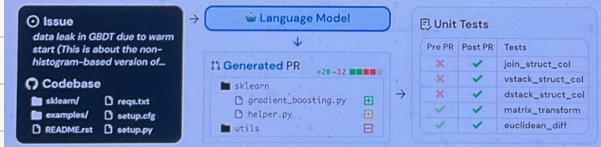
- An LLM like GPT-3 or GPT-4 would generate the function given the description & an empty function block in one go (zero-shot).



- The benchmark is measurable via unit test cases.
- If an output clears all the test cases then it is considered successful.
- GPT-3.5 gets a zero-shot accuracy of 48%. whereas GPT-4 is way ahead with 67% accuracy.
- The other markers on the above figure represent Agentic workflows. They're not one-shot.
- Here, we get a solution, if we can't get it to work on all the unit tests, we figure out where the model fails, feed it back into the model & iteratively refine our solution until a reasonable point.
- The figure illustrates the importance of Agentic flows. The boost we get from GPT-3 to GPT-4 can be surpassed just by using GPT-3 in an Agentic flow (although it may be more expensive because we may need to call the LLM multiple times & with longer context)

SWE-Bench-Agentic Workflow Example

- * LLMs are already expensive as it is.
- * Agentic Workflow says that iteratively using them will give a better solution.
- Q) But is spending 5x-6x more money worth it for the task?
- A) There are some cases. e.g:



- A lot of engineers keep track of their Github Repos.
- Once an issue is submitted, they work on it to identify if it is indeed a bug.
- If it is really so, they'll try to reproduce it,

work on a solution, patch it up without breaking the existing system and release it.

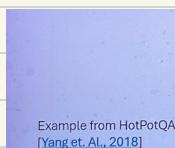
- This is a very time consuming cycle. It may take just a day or two for simple bugs or even weeks for a complicated one.
- So, the amount of money spent on a S/W engineer is much higher & also the amount of time they spend on these bugs is also higher.
- This is one of the cases where we can save money by using Agentic Workflows.
- But spending all this money is not worth it if we only get sub-optimal solutions which only end up working like 5-10% of the time. But if it could solve like 30-40% of the issues, then that could save a lot of time & money.

* It's not just the case with SW Engineering. Even something like S/W support is a good usecase: If a system fails in a cloud environment, the site engineer needs to go through a ton of logs to figure out the issue & come up with a solution. Automating things like that could save a ton of money.

ReACT

* Before discussing ReACT we should look at HotPotQA - a commonly used dataset for most of these frameworks to study.

* Typically, in a document-grounded QA system, we would use a retriever, query the corpus, get the document which most likely has the



Paragraph A: 2016–17 Chicago Bulls season
The 2016–17 Chicago Bulls season was the 51st season of the franchise in the National Basketball Association (NBA). Former 2011 MVP and 2009 Rookie of the Year Derrick Rose was traded from his hometown team to the New York Knicks. This was the first season without Rose since the 2012–13 season.

Dwyane Wade, who played for the Miami Heat from 2003 to 2016, decided to leave the Heat and sign with his hometown team, the Chicago Bulls. The Bulls traded Tony Snell to the Milwaukee Bucks for Michael Carter-Williams on October 17, 2016.

Paragraph B: Michael Carter-Williams
Michael Carter-Williams (born October 10, 1991) is an American professional basketball player for the Charlotte Hornets of the National Basketball Association (NBA). He was drafted 11th overall in the 2013 NBA draft by the Philadelphia 76ers, after playing college basketball for the Syracuse Orange. He was named NBA Rookie of the Year in 2014, and has also played for the Milwaukee Bucks and Chicago Bulls.

Q: To what team was the 2014 NBA Rookie of the Year traded in October 2016?

A: Chicago Bulls

response to our question. This is called single-hop reasoning.

* But this may not always be the case.
*) In the example above, we have "2014 NBA rookie of the year" which would be in one document & then with the name we will need figure out "which team the person was traded to in Oct 2016". This is a two-hop reasoning question. first, we find out that Michael Carter Williams was the 2014 NBA Rookie of the Year from one document. Then we find out that he was traded to Chicago Bulls in Oct 2016.

*) Similarly, we can have multi-hop reasoning.
*) The above example is a good use case to study how different ways of Agentic Workflows fare because it doesn't seem that the answer could be retrieved in one-shot (unless the LLM has already memorized the answer)
*) Another such task is a real world simulation game where you enter a room which has 5 objects & your goal is to put the salt & pepper shaker in the sink. You only see Cabinet 1, Cabinet 2, Drawer 1, Drawer 2, Table, Sink when you enter the room. You need to figure out where the salt & pepper shaker are by opening the cabinets & drawers. Such a task is good for agentic workflows because the model needs to keep track of its past actions & use the current observations to reason & solve the problems in an iterative manner.

*) Similarly, another such HotPotQA question is 'Aside from Apple Remote, what other device can control the program Apple Remote was originally designed to interact with?'

*) Here, we need to figure out what the remote was originally designed to interact with, how

The screenshot shows a reasoning chain for a question about what other device can control an Apple Remote. The chain consists of four steps:

- (1) **Hotpot QA**: Question: Aside from the Apple Remote, what other device can control the program Apple Remote was originally designed to interact with?
- (1a) Standard: Answer: iPod
- (1b) CoT (Reason Only): Thought: Let's think step by step. Apple Remote was originally designed to interact with Apple TV. Apple TV can be controlled by iPhone, iPad, and iPod Touch. So the answer is iPhone, iPad, and iPod Touch.
- (1c) Act-Only: Act 1: Search[Apple Remote] Obs 1: The Apple Remote is a remote control ...
Act 2: Search[Front Row] Obs 2: Could not find [Front Row]. Similar ...
Act 3: Search[Front Row software] Obs 3: Front Row is a discontinued media center software ...
Act 4: Finish[yes]

it interacts with the program & then the alternative.

* Using an LLM directly gave the answer as 'iPod' (1a) which is **wrong**.

* We can also use Chain-of-Thought reasoning

Here, it says that the Remote was designed to interact with Apple TV which can be controlled via iPod, iPhone, iPad (1b). This seems correct

given the assumption that Apple Remote was indeed designed to interact with Apple TV but that is not the case so this is also **wrong**.

* If the model has access to a **Tool** for searching over a corpus of documents (like WebGPT), then it can perform a lot of actions like Search, Go to another page, Go back, Reframe search query etc. If we use tools like this, we may get the right answer. Here, after a few searches it was able to figure out front Row Software was the right software. However, the final answer it gave was a 'YES' (1c) which is **wrong** & in fact doesn't even make sense for the query asked. So, the reasoning here was incorrect.

* In ReAct, we combine CoT & 'Act only' together. Once we get an observation back from the action taker, then just think about it. Here, the thought after the '1st' search result is that Apple Remote was originally designed to control the 'Front Row media center' program and need to search for Front Row next & find what other device can control it. It searches for it & doesn't find anything so it thinks of searching for 'Front Row Software'. Once it finds it, the final thought is that Front Row software is controlled by Apple Remote or the keyboard function keys so the answer is 'Keyboard function keys' which is **correct**.

(1d) ReAct (Reason + Act)
Thought 1: I need to search Apple Remote and find the program it was originally designed to interact with. Act 1: Search [Apple Remote]
Obs 1: The Apple Remote is a remote control introduced in October 2005 by Apple, originally designed to control the Front Row media center program.
Thought 2: Apple Remote was originally designed to control the Front Row media center program. I need to search Front Row next and find what other device can control it. Act 2: Search[Front Row]
Obs 2: Could not find [Front Row]. Similar: ['Front Row Seat to Earth', 'Front Row Motorsports', 'Front Row (Software)', ...]
Thought 3: Front Row is not found. I need to search Front Row (Software) . Act 3: Search[Front Row (Software)]
Obs 3: Front Row is a discontinued media center software ...
Thought 4: Front Row (software) is controlled by an Apple Remote or the keyboard function keys. So the answer is keyboard function keys. Act 4: Finish;Keyboard function keys

- * The notion of making the model reflect on what it observed from the outside environment was crucial.
- * Similar to RL agents - interact with environment, observe the environment & then proceed accordingly (in some sense Act is also like that). The only difference is that in RLs there is an idea that the action space is discreet (or we will never converge). The thought & action are the action space here. So, we have a discreet action space for searching & looking up & a countably infinitely large language action space so the idea doesn't naturally come to mind.
- * But the good part here is that LMs were so good by this time, that we don't have a 'cold start problem' i.e., we don't start with a policy that doesn't know anything. Even a few in-context examples will help us get an action with good non-zero rewards.
- * This technique would fail if LMs were not as good as they are (because we feed in a ton of input & little knowledge about what to do so it cannot figure out anything)

Results

- * CoT-SC (self-consistency) - An extension over CoT which says that rather than doing greedy decoding sample 20-25 outputs & do majority voting (or some other marginalisation) to figure the one with higher likelihood of being the answer.
- * They implemented Act & ReAct on top of HotpotQA but it performed worse than CoT (in this setup). [In fact, even standard output seems better]
- * They dug deeper & figured out that the source of failure for

Prompt Method ^a	HotpotQA (EM)	Fever (Acc)
Standard	28.7	57.1
CoT [Wei et al., 2022]	29.4	56.3
CoT-SC [Wang et al., 2022a]	33.4	60.4
Act	25.7	58.9
ReAct	27.4	60.9
CoT-SC → ReAct	34.2	64.6
ReAct → CoT-SC	35.1	62.0
Supervised SOTA ^b	67.5	89.5

PaLM-540B prompting results on HotpotQA and Fever

ReAct & CoT are different. CoT gives a really plausible answer even if its chain is incorrect. Often the CoT is not right but it still arrives at the right answer. ReAct cannot move ahead unless it has evidence (Because if the Act doesn't give it what it wants then it tries different Acts to converge at the solution. It cannot move ahead w/o that). A good analogy for this would be an assignment which has multiple modules. One student completes the assignment with some mistakes in (easy) module 2 (CoT). Another student tries multiple ways to solve module 2, spending a lot of time there & not moving forward (ReAct). The second method can be good at times but not always).

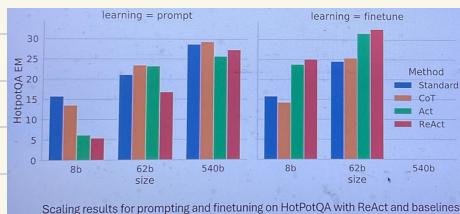
- * Since all methods have their pros & cons, so they decided to combine all & see how it works.
- * One method was they tried CoT-SC & applied some rules on the marginalised distribution. If the rules suggest CoT-SC is a winner then we use the output, otherwise we resort to ReAct.
- * Another approach was do ReAct (iteration of thought & action) for 5-6 steps. If it doesn't get the answer by then, we switch to CoT-SC.
- * Both the above techniques outperformed CoT-SC on Hotpot QA.

NOTE: CoT works only in scenarios where LLM understands the domain. It cannot be cues in all tasks. ReAct is a more open framework

- * Other tasks in the paper show how ReAct is better than the best baselines. eg: Symbolic logical reasoning
- * One way to perform all these techniques is by feeding few

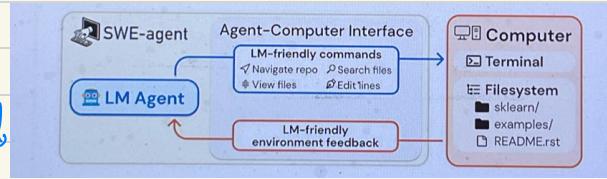
shot examples in the prompt & asking the model to follow the framework.

- * Another technique is (as discussed in the previous function calling lectures) to synthesise data (w/o human intervention) in an iterative manner & then use it to fine-tune a model.
 - A With this even a smaller model fine-tuned with ReAct (or Act) can surpass much larger models with simple prompting.
 - * So, this is a cost saving technique because we end up generating data for free as well, w/o the need for annotating the ReAct framework in the examples, given we have the tools at hand & the dataset with (input, output) pairs, we can simulate the ReAct framework, identify which subset we want to use for fine-tuning, fine-tune a model & end up with a model better at our particular task than a much larger model.



SWE-Agent using ReACT

- * The environment here is what a S/W engineer would interact with on a daily basis (repo, tools, terminal, debugger, issues, PRs, etc.)
- * The SWE Agent would work in the same kind of environment. It has access to all the files in our repo, an editor to edit the files, an environment to execute it, etc.
- * The SWE Agent using ReACT was able to solve over 20% on the SWE Bench (as opposed to RAct based solution previously which could only solve 10%).
- * Rather than giving it everything that the Linux Shell was capable of, they gave a very small subset of the commands (taking



info account how much we can fit in for the prompt context window).

*) Instead of giving the entire codebase, we can give it some tools to localize some parts of the codebase which is of interest to us & save the space to make reasoning over a small amount of data & the task more tractable than before.

*) Here we have a system instruction (which tells the agent what it is supposed to do) followed by a bunch of in-context examples.

*) Then we give the current issue & some information about how we could solve this (like an artifact) & then it follows the cycle of observation, thought & action to finally give us a patch with which we can create a PR.

*) Some examples of action & response would be to open a file name. Rather than opening the file, it will open a part of the file with some lines above it & some below it as well. It should decide if it wants to scroll up or down.

*) There are other things like find a file, search in a directory, etc.

*) Once it localizes where to fix in the code repo to get the right output, it thinks about it & figures out what to fix & using a specific edit command it changes the code.

Summary

ReACT

→ reasoning + act to arrive at the final answer

→ Can also have RAG based ReACT for Hotpot QA & do many other things

The screenshot shows a code editor with several panels:

- System Prompt:** Describes environment and commands, specifies response format.
- Demonstration:** Full trajectory of a successful example.
- Issue statement:** Reports an issue description, instructions to resolve issue, high-level strategy tips.
- Thought & Action:** Environment response collapsed.
- Environment Response:** Thought & Action.
- Environment Response:** Thought & Action.
- Environment Response:** Thought & Action.
- Submit:** Patch File diff -git a/src/sqlfluff/rules/rule008.py b/src/sqlfluff/rules/rule008.py --- a/src/sqlfluff/rules/rule008.py +++ b/src/sqlfluff/rules/rule008.py @@ -1,3 +1,3 @@

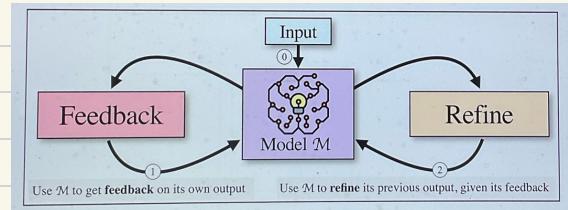
Response (Thought + Edit Action)

We need to add a check to ensure that all powers are the same. Let's edit the code to include this check.

```
edit 404:407
      elif (total_degree > 3 and
           len(set(k.exp[k.is_Pow])) == 1):
          if all(coeff[k] == 1):
            diop_type = 'general_sum_even'
end_of_edit
```

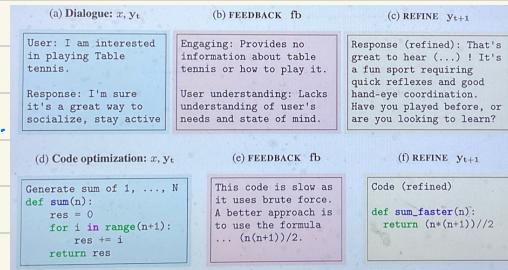
Self-Refine

- * Also works iteratively but in a different manner.
- * Doesn't go step-by-step trying to figure out what tools to use, what prompts to use & so on.
- * Generate the output however you want (printing, CoT, ReACT).
- * Once you arrive at the solution, look at it and at the question & judge if it is right.
- * When the input comes it generates out the output, sends it to the feedback module, based on the feedback it refines the output again & the process repeats till the judge (feedback module) is happy with the output.
- * Very simple but effective solution.



Where can we use it?

- * When building a chatbot, LLMs may not have as much filters as we want. They may not be able to follow very fine-grained instructions. So, we let the LLM generate a response & have a list of instructions which we can check individually or in a grouped manner if they are being followed. Once we get the output we can identify what is not being followed or where we need changes & it is easy to instruct the LLM to make changes to only those parts.
- * A big use case is in code optimization. Once the code is generated, we can have a prompt in the feedback to specifically discuss the complexity. In the above example, the model generates a code to find the sum of n numbers by iterating through



them. The feedback models suggest that this code is slow & a better solut^by would be to use the formula $n(n+1)/2$ which helps the model refine its output.

* Self-Refine is like a prompting strategy. In addition to getting the output, refine it with a different prompt & see how it iterates.

Results

* Huge boost on multiple datasets using self-refine.

- * A good thing is that we don't need to worry about what the base prompting strategy is. The base strategy might be something like CoT-SC which is already very good. Here, we just write a wrapper on top of it.
- * It is more of a complimentary technique (to methods like standard prompting, CoT, CoT-SC, ReAct which are of the same type)
- * Hence, in st cases it either maintains the same performance or gives a good jump. eg: code readability, dialogue response, etc.
- * Discrimination is an easier task than generation. Given an output it is easier to verify if it is correct or not compared to generating one. This is one of the reason this technique works well.
- * Another reason is that while a system prompt contains detailed instructions of how to perform a task, feedback model can specifically point out where the LLM is not doing a good job so it can correct itself. This difference in prompting strategies helps refine the results. (Also feedbacks can be multi-dimensional; and we can score the output on each dimension & set an overall (or dimension specific) threshold to be considered an acceptable output)

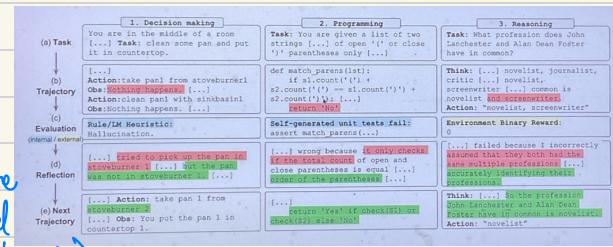
Task	GPT-3.5		ChatGPT		GPT-4	
	Base	+SELF-REFINE	Base	+SELF-REFINE	Base	+SELF-REFINE
Sentiment Reversal	8.8	30.4 (\uparrow 21.6)	11.4	43.2 (\uparrow 31.8)	3.8	36.2 (\uparrow 32.4)
Dialogue Response	36.4	63.6 (\uparrow 27.2)	40.1	59.9 (\uparrow 19.8)	25.4	74.6 (\uparrow 49.2)
Code Optimization	14.8	23.0 (\uparrow 8.2)	23.9	27.5 (\uparrow 3.6)	27.3	36.0 (\uparrow 8.7)
Code Readability	37.4	51.3 (\uparrow 13.9)	27.7	63.1 (\uparrow 35.4)	27.4	56.2 (\uparrow 28.8)
Math Reasoning	64.1	64.1 (0)	74.8	75.0 (\uparrow 0.2)	92.9	93.1 (\uparrow 0.2)
Acronym Generation	41.6	56.4 (\uparrow 14.8)	27.2	37.2 (\uparrow 10.0)	30.4	56.0 (\uparrow 25.6)
Constrained Generation	28.0	37.0 (\uparrow 9.0)	44.0	67.0 (\uparrow 23.0)	15.0	45.0 (\uparrow 30.0)

Summary

- iteratively improve initial results based on model feedback
- can be combined with ReAct

Reflexion

- * Can be thought of as an extension of self-refine.
- * It is a combination of self-refine with tools (typically self-refine would use the same LLM to give feedback)
- * The paper suggests not to be biased towards just LLMs but use different evaluation metrics. e.g.: for code generation use tools like syntax checker, code interpreter, etc. and if there is any error then feed it back to the system
- * The output of the system is fed into a tool (could be a heuristic tool, test cases, binary reward, etc.), reward is obtained & then we do the reflection on top of the output, the trajectory taken & the task. (So it unentangles the two tasks i.e., as opposed to self-refine where the evaluator & feedback generator are tightly coupled, here we decouple them so we can use any evaluation metric we want)



1. Decision making
2. Programming
3. Reasoning

Task: You are given a list of two strings [...] of open '(' or close ')' parentheses only [...]

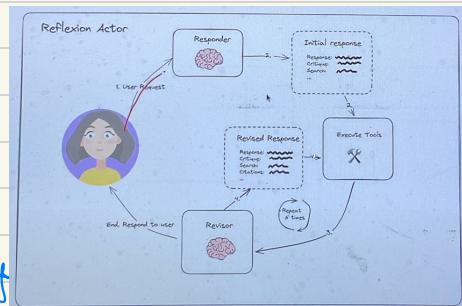
Task: What profession does John Leechester and Alan Dean Foster have in common?

Think: [...] novelist, journalist, critic, novelist, screenwriter, [...] common is novelist and screenwriter.
Action: "novelist, screenwriter"

Environment Binary Reward: 0

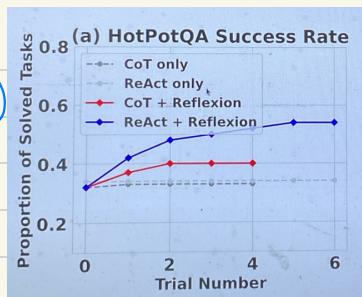
Task: [...] failed because I incorrectly assumed that they both had the same profession. I need to be more accurate identifying their professions.
Action: "The profession John Leechester and Alan Dean Foster have in common is novelist."

- Request goes in, LLM gives a response.
- Response is fed into the tool.
- The output of tool & the rest is again fed to the LLM to figure out the feedback & then create a new output.
- Repeat the process till you get a satisfactory response.



Results

- * The simple de-coupling (of evaluation & feedback module) gave a really good boost.
- * A big issue with REACT on HotpotQA was that it couldn't recover once it failed.
- * But REACT + Reflexion was able to recover (Remember, Reflexion is like a replacement for Self-refine & can be used with the REACT pipeline as the LLM) & give much better results.



eg:

Reflexion

Action 4: Finish!Rene Artois
Observation 4: Answer is INCORRECT
Reflection: I searched the wrong title for the show, "Allo 'Allo!", which resulted in no results. I should have searched the show's main character, Gordon Kaye, to find the role he was best known for in the show.

Action 2: Search["Allo 'Allo"]
Observation 2: Could not find ["Allo 'Allo!"].
Similar: ["Allo 'Allo"], "List of 'Allo 'Allo' characters", "Gordon Kaye", "'Allo 'Allo' (series 2)", "Allo 'Allo", "Francesca Gonshaw", "'Allo 'Allo' (series 4)", "Kim Hartman", "'Allo 'Allo' (series 6)", "Jennette Lloyd".
Thought 1: I need to search the actor who was best known for a role on "Allo 'Allo!".
Action 1: Search[Gordon Kaye]
Observation 1: Grown-Ups is a 1980 British BBC television film devised and directed by Mike Leigh. It stars Lesley Manville, Philip Davis, Brenda Blethyn, Janine Davieski, Lindsay Duncan and Sam Kelly. It was edited by Robin Sales and produced by Louis Marks for the BBC, and originally shown on BBC 2 on 28 November 1980.
Thought 2: Grown-Ups starred Lesley Manville, Philip Davis, Brenda Blethyn, Janine Davieski, Lindsay Duncan and Sam Kelly. I need to search "Allo 'Allo!" and find which actor was best known for which role.

Action 4: Finish!Rene Artois

Reflexion

Action 4: Finish!Rene Artois
Observation 4: Answer is INCORRECT
Reflection: I searched the wrong title for the show, "Allo 'Allo!", which resulted in no results. I should have searched the show's main character, Gordon Kaye, to find the role he was best known for in the show.

Action 2: Search[Sam Kelly]
Observation 2: Roger Michael Kelly (19 December 1943 – 14 June 2014), known by the stage name Sam Kelly, was an English actor who appeared in film, television, radio and theatre. He is best known for his roles as Captain Hans Geering in "Allo 'Allo!", Warren Porridge, Sam in "On the Up", and Ted Liveridge in "Barbara".
Thought 3: Sam Kelly is best known for his role as Captain Hans Geering in "Allo 'Allo!", so the answer is Captain Hans Geering.
Action 3: Finish!Captain Hans Geering
Observation 3: Answer is CORRECT

→ Reflexion also performs well on coding tasks.

Except for I care it improves the accuracy on other benchmarks.

Benchmark + Language	Base	Reflexion
HumanEval (PY)	0.80	0.91
MBPP (PY)	0.80	0.77
HumanEval (RS)	0.60	0.68
MBPP (RS)	0.71	0.75

Overall accuracy for HumanEval and MBPP

Summary

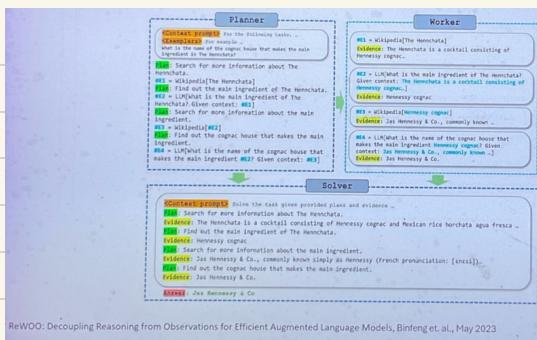
- iteratively improving initial results based on model feedback.
- use of tools & LLMs for reflection
- can be combined with REACT.

ReWOD

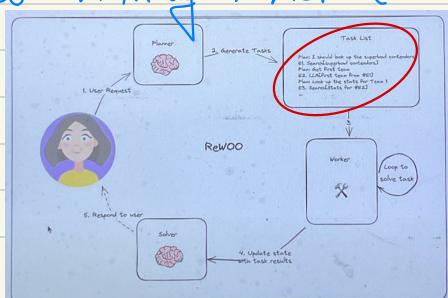
- * When we do REACT, Reflexion, etc. with something like SWE agent

we get a huge amount of logs and as we keep redoing they become even bigger & eventually, we might lose track of where we began with in the first place. It is also a very expensive process.

- * So people started thinking about how to reduce the context window.
- * Here, we first create a plan of what all needs to be done & then execute it separately.
- * But creating the plan together is the main difference (Because in ReACT we have an idea of what must be solved, but we always think through the 1st step, execute it, get the observation & then only do we go to the next step. So, the previous reasoning gets appended each time)
- * Here, we split the plan into multiple chunks & work through them.
- * Nowadays, people use a combination of the two in the real world applications mainly because there's a lot of cases where we use tools & LLMs to help plan well.
- * Here, we create a plan, execute it (in parallel, if needed), get all the outputs together & generate the response. So, we don't need to plugin the response of each one to generate the next and we end up saving token space & money.
- * So, the most obvious choice looks like mixing ReACT & ReWOO together.
- * The independently executable plan is what is most appealing because designing a ReACT system is a lot of effort but designing a planner is much simpler.



ReWOO: Decoupling Reasoning from Observations for Efficient Augmented Language Models, Binfeng et al., May 2023



(Because in ReAct we need a real life trajectory. So, we need a combination of real-life examples & simulated scenarios to create a good in-context example)

- * But a planner can be created by any LLM which can be trained for doing planning.
- * The loop on the worker indicates that if it fails, we do ReACT within the worker.

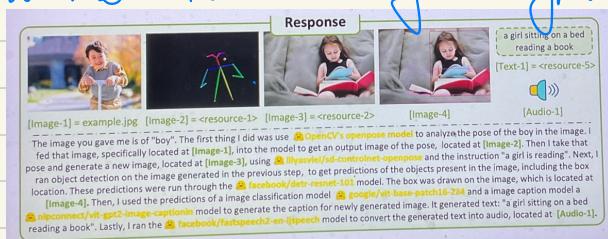
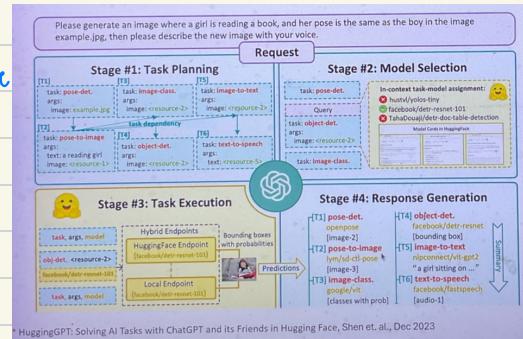
HuggingGPT

* There are a ton of models on Hugging Face & each one is good at something - some in STT, some in image captioning, some in generation, etc.

* If we want to do a task that requires using a lot of these then we can either wait for multi-modal LLMs to become that good or just stitch things together with what we have.

* In the above example, the input is an image 'example.jpg' along with the prompt:
* first we need to create a plan (same as ReWOO) - do pose detection → then pose-to-image → read the image of the girl → create a caption → use voice to convey the caption
* It also creates the dependency graph.
* Execute the tasks one at a time after selecting the right models.

* Then we generate the final response which is the audio.



Summary (ReWOO & Hugging GPT)

- decouples reasoning from observation
- reduce token consumption & execution

Multi-Agent Systems

*) Everything we studied till now falls under an individual agent setup - Get a task, choose an action, observe & reflect, iterate, etc.

*) The prompt that generates the input is different from the prompt that reflects & this helps us iteratively improve the response.

*) Here we take it to another level. e.g.: for SWE Agent creating a persona called Tester & a persona called Developer & treat each persona as an agent to create agentic solution for them.

*) Say we want to create an online painter. So, we need some inspiration for something say painting a beach. Search agent can help us understand what a beach looks like, what painting styles can be used for it, etc. i.e., it helps us generate ideas on how to paint the beach. The actual job of painting can be done by a painting agent.

*) Similarly, the other example is of Philosophers Chatting where each philosopher can be mimicked via an agent.

*) Such an approach of disentangling things can help us work with very clear & different instruction following models.

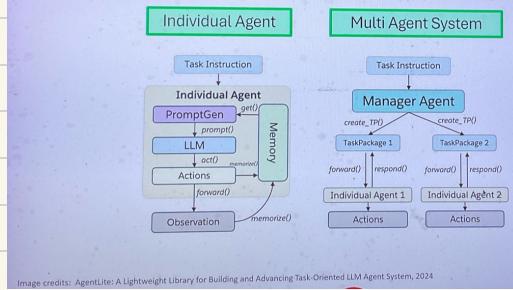
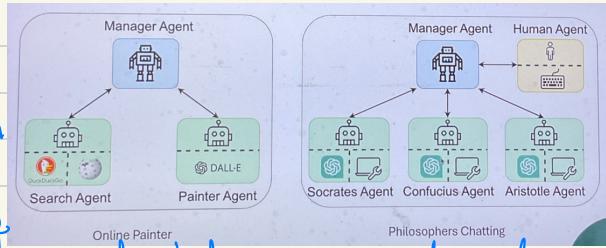


Image credits: AgentLIT: A Lightweight Library for Building and Advancing Task-Oriented LLM Agent System, 2024



Frontend Topic (as of 2025)