RAG and AI Agents

What a language model can do beyond its weights

CSCI 567 — Machine Learning • Discussion 10 • Nov 13, 2026

Two limits of a model that answers from its weights alone

Lecture 10 studied what a language model's weights hold and how that grows with scale. Two of its numbers show what the weights alone do not give.

Facts. GPT-3 answers 64.3% of TriviaQA questions zero-shot. Whatever is not in the weights, or has changed since training, it cannot know.

Retrieval-augmented generation: look the facts up at inference time and put them in the prompt.

Exact computation. GPT-4 gets 59% of 3-digit by 3-digit products right zero-shot (Dziri et al., 2023): small per-step errors compound.

Tool use: hand the sub-task to a program that does it exactly, and read back the result.

Both remedies leave θ untouched and change the *input*: they are in-context learning (Lecture 10, Section 4) with the context filled by a search index or a tool instead of by hand.

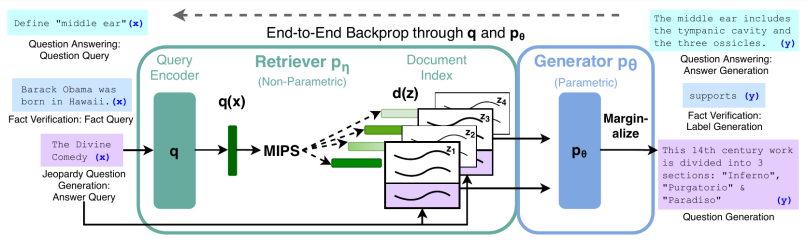
Parametric and non-parametric memory

	Parametric memory	**Non-parametric memory**
Where knowledge lives	in the weights θ	in an external store: documents, a database, the web
When it is written	at training time	whenever the store is edited
Updating one fact	retrain or fine-tune	edit one document
Can cite a source	no	yes, the retrieved passage
Cost at inference	one forward pass	a search, then a forward pass on a longer prompt
Failure mode	confident answer with no source	wrong or irrelevant passage retrieved

The terms are from Lewis et al. (2020), who introduce retrieval-augmented generation (RAG) as "models which combine pre-trained parametric and non-parametric memory for language generation".

A RAG system uses both at once. The weights supply language and reasoning; the store supplies the facts. The store can change without touching the weights, which matters because Lecture 10 showed what retraining at scale costs.

RAG (Lewis et al., 2020): a retriever in front of a generator



Lewis et al. (2020), Figure 1.

Reading the figure, left to right. A query x (a question, a claim to verify, a Jeopardy answer) is encoded as a vector $q(x)$. Maximum inner product search (MIPS) finds the stored documents $z_1, \ldots, z_K$ whose vectors $d(z)$ are closest to it. The generator p_θ reads the query together with each document and produces the output y; the per-document predictions are then combined ("marginalize").

Green block: the non-parametric memory. Blue block: the parametric one. Both are pretrained, then fine-tuned together (dashed arrow).

RAG as a probabilistic model: sum over the retrieved documents

Input text x, output text y, retrieved document z (plain letters: these are strings).

Retriever. A query encoder q and a document encoder d, both BERT, map text to vectors, and a document is likely when its vector has a large inner product with the query's:

$$p_\eta(z \mid x) \;\propto\; \exp\big(d(z)^\top q(x)\big).$$

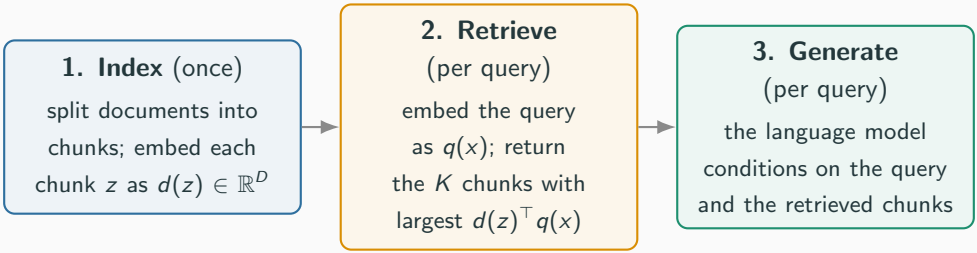
Generator. A sequence-to-sequence Transformer $p_\theta(y \mid x, z)$ reads the query and one document, concatenated.

Combine. Treat z as a hidden variable and sum it out over the top K documents:

$$p(y \mid x) \;\approx \sum_{z \in \text{top-}K(p_\eta(\cdot \mid x))} p_\eta(z \mid x)\, p_\theta(y \mid x, z).$$

Sizes in the paper. Store: the December 2018 Wikipedia dump in disjoint 100-word chunks, 21 million of them. Search: approximate MIPS with a FAISS index, sub-linear in the store size; K is 5 or 10. Generator: BART-large, 400M parameters.

The pipeline in three steps, and what each step is for



What each step is for. Indexing makes 21 million chunks searchable without reading them at query time. Retrieval cuts them to the few that fit in a prompt. Generation turns passages into an answer in the form the question asked for.

Chunking strategy, the choice of embedding model, and index tuning are engineering topics for the companion Deep Learning and Applications course. Here the object of study is the model $p(y \mid x)$ above.

Worked example: retrieve by inner product, then marginalize

Setup ($D = 2$ so it can be checked by hand). Query $q = (4, 3)$; three indexed chunks $a = (3, 4)$, $b = (-4, 3)$, $c = (0, 5)$. All four vectors have norm 5.

chunk	$q^\top v$	$\cos(q, v) = q^\top v / (\|q\| \, \|v\|)$	**rank**
$a = (3, 4)$	$4 \cdot 3 + 3 \cdot 4 = 24$	$24/25 = 0.96$	1
$c = (0, 5)$	$4 \cdot 0 + 3 \cdot 5 = 15$	$15/25 = 0.60$	2
$b = (-4, 3)$	$4 \cdot (-4) + 3 \cdot 3 = -7$	$-7/25 = -0.28$	3

Retrieve with $K = 2$: chunks a and c. Take the retriever's weights as a softmax over the cosine scores of the retrieved pair: $p_\eta(a \mid x) = e^{0.96}/(e^{0.96} + e^{0.60}) = 0.59$ and $p_\eta(c \mid x) = 0.41$.

Generate. Suppose the generator gives the correct answer y probability 0.9 when it reads a and 0.2 when it reads c. Then

$$p(y \mid x) = 0.59 \times 0.9 + 0.41 \times 0.2 = 0.53 + 0.08 = 0.61.$$

Evidence: retrieval beats a larger closed-book model; the index can be swapped

Natural Questions, exact match		
T5-11B, closed book	11B parameters	34.5
RAG-Sequence	400M generator + index	44.5

Lewis et al. (2020), Table 1.

A generator about 27× smaller, given the right passages, answers 10 points more questions than a model that must hold every fact in its weights.

"Who is {position}?", 82 world leaders	**2016 index**	**2018 index**
answers correct for 2016	70%	12%
answers correct for 2018	4%	68%

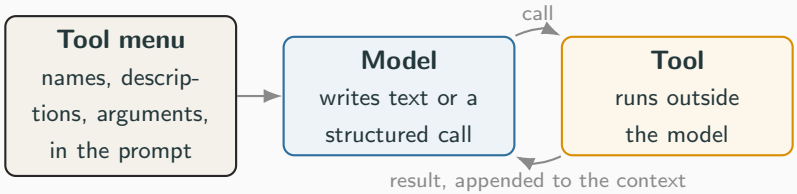
Lewis et al. (2020), index hot-swapping experiment.

Same weights, two Wikipedia snapshots. The model's answers follow the index: its knowledge of who holds an office was updated by replacing the store.

What retrieval does not fix. The answer is only as good as the passages: the 4% and 12% cells are the model following a wrong index. Retrieval adds a search to every query, and the generator can still ignore or misread what it was given.

Tool use: the model writes a call, a program runs it, the result returns as text

RAG gives the model *read* access to a store; an **agent** can also *act*: calculate, run code, call an API.



Function calling in one turn: "What is $\sqrt{2024} \times 17$, to two decimals?"

1. The model writes `calculator(expression="sqrt(2024)*17")` instead of guessing digits.
2. The harness runs it: $\sqrt{2024} = 44.9889$, times 17 is 764.81.
3. The result is appended to the context; the model reads it and answers "about 764.81".

What a tool fixes, and how tool use fails

Tool	What the weights alone do badly	Example call
calculator	exact arithmetic over many digits	`calculator("sqrt(2024)*17")`
search or retrieval	facts after the training cutoff, with a source	`search("population of Chicago")`
code interpreter	long exact procedures: sort, simulate, parse	`python("sorted(xs)[:10]")`
actions with side effects	changing the world, not describing it	`send_email(to, subject, body)`

Failure modes.

Wrong arguments. The right tool with a malformed or wrong-unit input (Fahrenheit where Celsius was meant): the tool computes exactly the wrong thing.

Unnecessary calls. A call for something the weights already had adds latency and a place to fail.

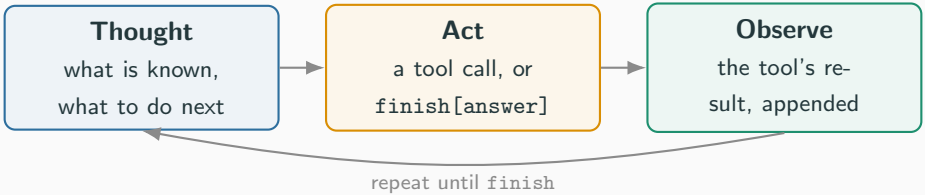
Blind trust. An empty or failed result is treated as ground truth. With side effects the cost is higher: a wrong search wastes a step, a wrong email cannot be unsent.

Deciding what to do *after* a result needs a reasoning step between calls: ReAct, next.

ReAct (Yao et al., 2023): interleave reasoning with acting

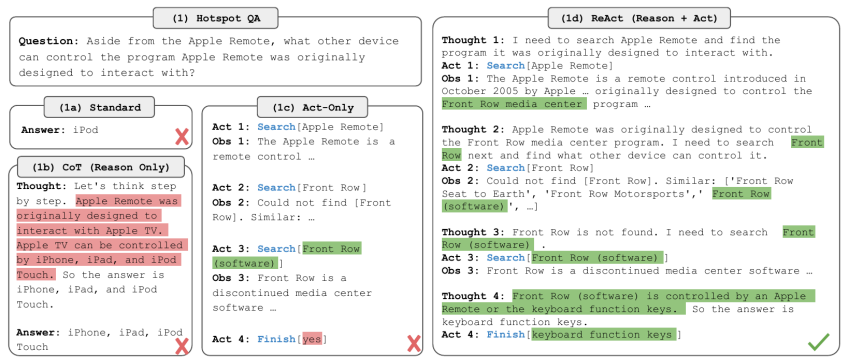
Setup. At step t the agent has a context $c_t = (o_1, a_1, \ldots, o_{t-1}, a_{t-1}, o_t)$ of past observations and actions (the prompt so far) and picks a_t from a set $\mathcal{A}$ of tool calls.

The idea. Enlarge the action set to $\hat{\mathcal{A}} = \mathcal{A} \cup \mathcal{L}$, where $\mathcal{L}$ is free text. A text action is a **thought**: it changes nothing outside and returns no observation; it is only appended to the context, $c_{t+1} = (c_t, \hat{a}_t)$.



No weights are trained: a frozen model is prompted with a few solved trajectories in this format (6 for HotpotQA, 3 for FEVER) and continues the pattern: few-shot in-context learning. We use only the loop; a formal decision-process treatment is outside this course.

The same question answered four ways



Yao et al. (2023), Figure 1 (top). One HotpotQA question under four prompting methods. Red: invented or wrong content. Green: content supported by an observation.

The question needs two hops: what the Apple Remote was originally designed to control, then what else controls that program.

Reading the four traces: where each method fails or succeeds

Method	What the model may do	What happens on this question
(1a) Standard	answer directly	wrong answer, no way to see why
(1b) Reasoning only (chain of thought)	write thoughts, no tools	the reasoning rests on a recalled "fact" that is false
(1c) Acting only	call search, no thoughts	finds the right pages, cannot combine them, answers "yes"
(1d) ReAct	thoughts and search, interleaved	reformulates a failed search, reads the answer off the result

What the thoughts are used for (the paper's list): breaking the goal into steps, pulling the relevant part out of an observation, tracking progress, and changing the plan when a result is unexpected. In (1d), Thought 3 is the last kind: "Front Row is not found. I need to search Front Row (software)."

Reasoning without acting has nothing to check its facts against. Acting without reasoning has nowhere to combine what it found. The thought step is where a plan is revised and where observations are turned into an answer.

Worked trace: a two-hop question in three tool calls

Question. "Is the population of the capital of France larger than that of Chicago?"

t	**Thought**	**Act**	**Observe**
1	I need the capital of France, then two populations.	`search("capital of France")`	Paris
2	The capital is Paris. I need its population.	`search("population of Paris")`	about 2.1 million (city proper)
3	Now Chicago's.	`search("population of Chicago")`	about 2.7 million
4	2.1M $<$ 2.7M, so the answer is no.	`finish("No, Chicago is larger")`	

- The second call could not have been written in advance: its argument, Paris, is the first call's observation. A fixed script of calls cannot do this; a loop can.
- Each thought records what has been established, so the last step compares two numbers that are both in the context and guesses neither from the weights.
- Retrieval is one of the tools: here the agent *chooses* when to search, where classic RAG retrieves once for every query.

The observations are illustrative search results written for this example.

Evidence from the paper: fewer invented facts, more reasoning errors

Prompting method	HotpotQA (exact match)	FEVER (accuracy)
Standard	28.7	57.1
Chain of thought (CoT)	29.4	56.3
Act only	25.7	58.9
ReAct	27.4	60.9
Best ReAct + CoT hybrid	35.1	64.6

Yao et al. (2023), Table 1; PaLM-540B.

HotpotQA, share of failures due to	ReAct	CoT
hallucinated reasoning or facts	0%	56%
reasoning error (including repeated steps)	47%	16%
search returned nothing useful	23%	n/a

Yao et al. (2023), Table 2 (label-ambiguity row omitted).

Reading the tables. ReAct beats acting alone on both tasks, so the thoughts help. Against chain of thought it wins on FEVER and is slightly behind on HotpotQA. The failure table says why: grounding in observations removes invented facts, but the loop can get stuck repeating a step, and a bad search result leaves it with nothing to reason from. The best results combine the two and fall back from one to the other.

On the interactive benchmarks ALFWorld and WebShop, ReAct with one or two in-context examples beats imitation and reinforcement learning baselines by 34 and 10 points of absolute success rate.

Practice 1: inner product and cosine similarity can disagree

Problem. Query $q = (1, 0)$; chunks $a = (3, 4)$ and $b = (1, 0.5)$. Which chunk is returned by (i) maximum inner product search, (ii) cosine similarity? (iii) When do the two always agree?

Solution.

- (i) $q^\top a = 3$ and $q^\top b = 1$: inner product returns a.
- (ii) $\|a\| = 5$ and $\|b\| = \sqrt{1.25} \approx 1.118$, so $\cos(q, a) = 3/5 = 0.60$ and $\cos(q, b) = 1/1.118 \approx 0.89$: cosine returns b.
- (iii) $q^\top v = \|q\|\,\|v\| \cos(q, v)$. For a fixed query the two rankings differ only through $\|v\|$, so they agree whenever all chunk vectors have the same norm, as in the worked example where every norm was 5.

Inner product rewards long vectors: a is 53° from the query and wins because it is $4.5\times$ longer than b. With unit-norm embeddings the two rankings coincide.

Practice 2: errors compound over the steps of a trace

Problem. An agent solves a task in k steps; each step (thought, call, reading the result) is right with probability p, independently. (i) What is the probability the whole trace is right? (ii) Evaluate it for $p = 0.95$ with $k = 4$ and $k = 10$. (iii) For $k = 10$, what p gives a 90% success rate?

Solution.

- (i) All k steps must be right: p^k. This is Lecture 10's exact-match formula, with steps in place of tokens.
- (ii) $0.95^4 \approx 0.81$ and $0.95^{10} \approx 0.60$.
- (iii) $p^{10} = 0.9$ gives $p = 0.9^{1/10} = e^{\ln(0.9)/10} = e^{-0.0105} \approx 0.9895$: about one error in a hundred steps.

Long traces need per-step reliability near 1. A thought after each observation lets the agent repair a bad step, as Thought 3 did in the paper's example.

Summary

- **Two memories.** Parametric memory is written at training time and cannot cite; non-parametric memory is a store that can be edited and quoted, at the cost of a search.
- **RAG.** $p(y \mid x) \approx \sum_{z \in \text{top-}K} p_\eta(z \mid x)\, p_\theta(y \mid x, z)$ with $p_\eta(z \mid x) \propto \exp(d(z)^\top q(x))$. Natural Questions: 44.5 against 34.5 for a closed-book model 27$\times$ larger; swapping the index swaps the answers.
- **Tool use.** The model writes a structured call, a program runs it, the result returns as text. Failures: wrong arguments, needless calls, trusted bad results.
- **ReAct.** Thoughts are actions that only extend the context. Interleaved with tool calls they remove invented facts (0% against 56% of failures) and add reasoning loops (47% against 16%).
- **Compounding.** A k-step trace succeeds with probability about p^k: $0.95^{10} \approx 0.60$.

Coming up. HW4 is due Monday 11/30. Lecture 11 (11/20): reinforcement learning, from bandits to policy gradients, and its use in post-training language models (RLHF, RLVR).

References

- Lewis, Perez, Piktus, et al. *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks.* NeurIPS 2020; arXiv:2005.11401.
- Yao, Zhao, Yu, Du, Shafran, Narasimhan, Cao. *ReAct: Synergizing Reasoning and Acting in Language Models.* ICLR 2023; arXiv:2210.03629.
- Brown, Mann, Ryder, et al. *Language Models are Few-Shot Learners.* NeurIPS 2020; arXiv:2005.14165. (TriviaQA numbers.)
- Dziri, Lu, Sclar, et al. *Faith and Fate: Limits of Transformers on Compositionality.* NeurIPS 2023; arXiv:2305.18654. (Multiplication numbers.)