The Next Frontier of NLP

From Prediction to Intelligence

MSc NLP Course – Final Lecture

How do we go from predicting tokens to building AI that is
USEFUL, SMART, and SAFE?

The Story in 10 Slides

A Visual Roadmap of Today's Lecture

1/10

The Problem

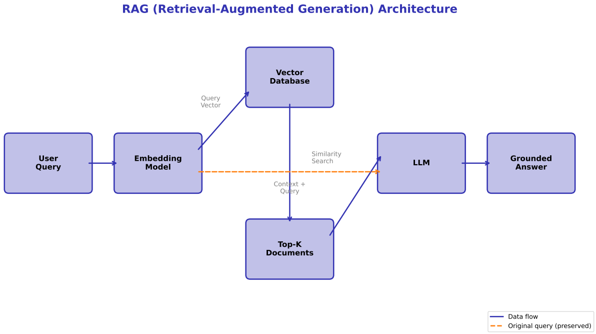
LLMs predict tokens brilliantly...

but they hallucinate, can't access current info,

and struggle with complex reasoning

Key insight: Raw language models need augmentation to become truly useful systems.

2/10: RAG – Retrieval-Augmented Generation

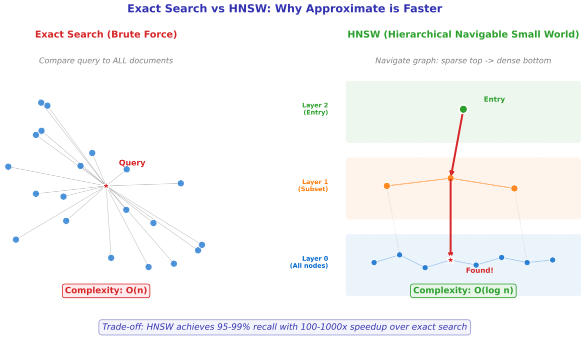


Solution #1: Give LLMs access to external knowledge

Query → Retrieve relevant docs → Augment prompt → Generate

RAG: The most widely deployed technique for grounding LLMs in real-world facts.

3/10: Vector Search – How Retrieval Works

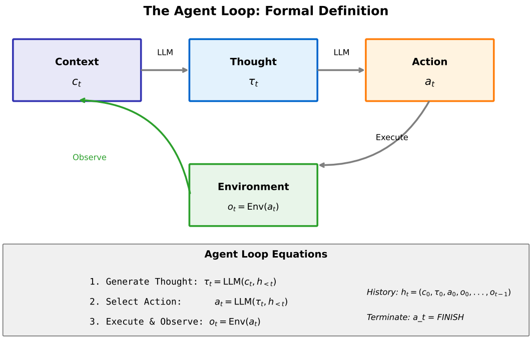


Find similar documents in milliseconds from billions

Approximate Nearest Neighbor: Trade 1-5% accuracy for 1000× speedup

HNSW and ANN algorithms make RAG practical at scale – billions of vectors, millisecond latency.

4/10: Agents – LLMs That Take Action



Solution #2: Let LLMs use tools and take actions

Think $\rightarrow$ Act $\rightarrow$ Observe $\rightarrow$ Repeat until done

Agents extend LLMs from passive responders to active problem solvers.

The Reasoning Problem

Standard prompting:

"Q: If a train leaves at 9am traveling 60mph, and another at 10am at 90mph..."

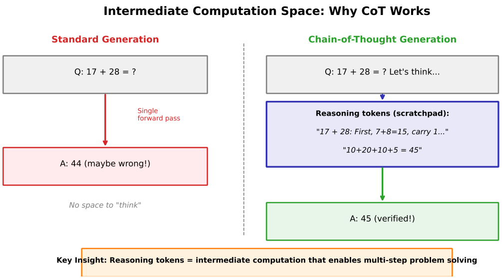
"A: The first train arrives first." (often wrong)

LLMs give instant but shallow answers

They don't "think through" multi-step problems

The reasoning gap: Models generate fast but often skip the steps humans use to think.

6/10: Chain-of-Thought – Teaching Models to Think



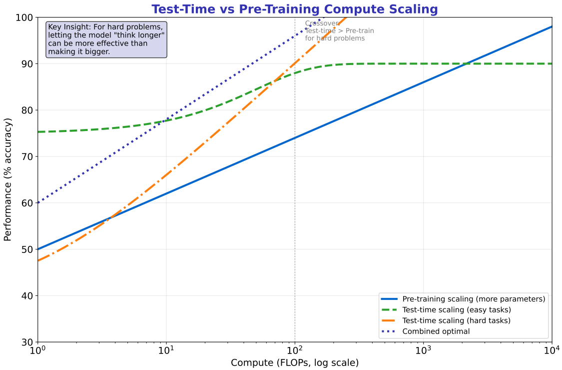
Solution #3: "Let's think step by step"

Intermediate tokens = scratchpad for computation

Result: +40% accuracy on math/reasoning tasks

Chain-of-Thought: A simple prompt change that dramatically improves reasoning.

7/10: Test-Time Compute – More Thinking = Better Answers



The paradigm shift: Scale inference, not just training

o1, DeepSeek-R1: Models trained to think longer on hard problems

Test-time compute: The new scaling law – spending more compute at inference for harder problems.

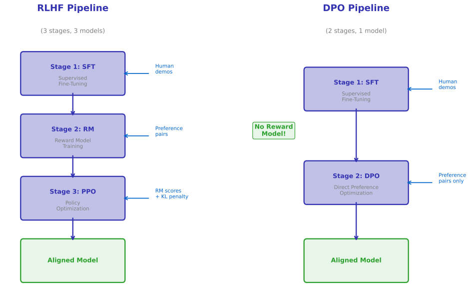
The Alignment Problem

Raw pre-trained models are...

- Not helpful (don't follow instructions)
- Not safe (will generate harmful content)
- Not honest (confidently wrong)

How do we align LLMs with human values?

9/10: RLHF & DPO – Learning from Human Preferences



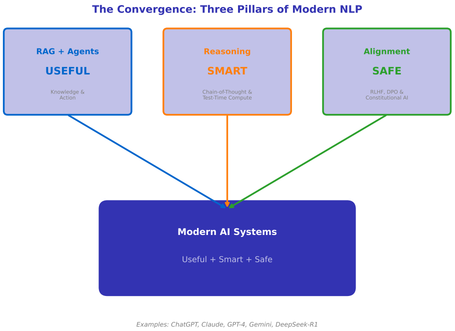
Solution #4: Train on human feedback

RLHF: Reward model + RL optimization

DPO: Direct preference optimization (simpler!)

RLHF/DPO: How ChatGPT and Claude learned to be helpful assistants, not just text predictors.

10/10: The Convergence – Modern AI Systems



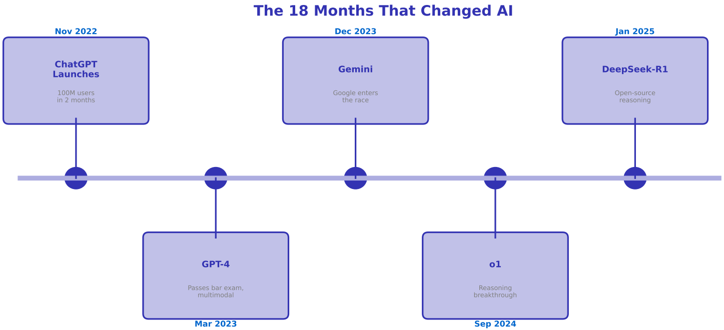
USEFUL + SMART + SAFE

RAG & Agents + Reasoning + Alignment

This is how ChatGPT, Claude, Gemini actually work

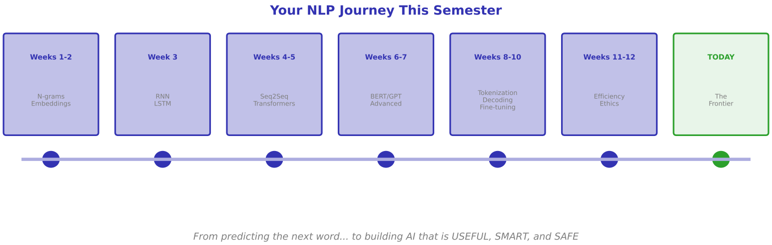
Modern AI systems combine all these techniques – retrieval, reasoning, and alignment working together.

The Most Important 18 Months in AI History



What changed? The architecture is largely the same transformer you learned in Week 5. So what's different?

Your NLP Journey This Semester



You learned to predict words. Today: how to make those predictions USEFUL, SMART, and SAFE.

Today's Central Question

What You Already Know

- Transformers and attention
- Pre-training (BERT, GPT)
- Fine-tuning and LoRA
- Prompt engineering
- Efficiency and deployment
- Ethics and responsibility

What's Missing?

- LLMs hallucinate and have knowledge cutoffs
- They struggle with multi-step reasoning
- Raw LLMs aren't naturally helpful or safe
- How do we deploy them in the real world?

Today's Answer: Three breakthroughs that transformed language models from impressive demos into systems that might actually change how we work.

Three topics: RAG & Agents (USEFUL) — Reasoning (SMART) — Alignment (SAFE)

Act I: RAG & AI Agents

Making LLMs Useful in the Real World

The Hallucination Problem

The Problem

LLMs confidently state wrong facts:

- "The current CEO of OpenAI is..." (outdated)
- "The 2024 Olympic gold medalist was..." (unknown)
- "Your company's Q3 revenue was..." (not in training data)

Root Causes

- Knowledge frozen at training time
- No access to private/recent information
- Model "fills in gaps" with plausible text

Why This Matters

For real applications, we need:

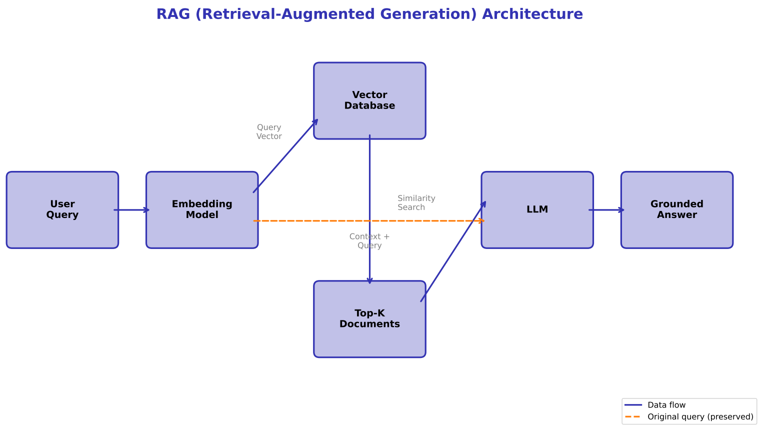
- Access to current information
- Grounding in verifiable sources
- Ability to say "I don't know"

Connection to Ethics Week

Remember: LLMs don't "know" anything – they predict tokens. Without grounding, this is dangerous.

Solution: Don't try to store everything in parameters. Retrieve at inference time.

RAG: The Elegant Solution



Key insight: Separation of concerns – parametric knowledge (the model) vs. retrieved knowledge (the database)

RAG: The Mathematics

Core Idea
Instead of: $p(y|x)$ (generate from query alone)
RAG marginalizes over retrieved documents:

$$p(y|x) = \sum_{z \in \text{top-}k} p(z|x) \cdot p(y|x, z)$$

Why no z on left? We sum over all z (marginalization) – the result depends only on x.

Where: $x =$ query, $z =$ retrieved doc, $y =$ response

Key Equation: Dense Retrieval

$$\text{sim}(q, d) = \frac{E_q(q)^T \cdot E_d(d)}{||E_q(q)|| \cdot ||E_d(d)||}$$

Retrieval probability (softmax):

$$p(z_i|x) = \frac{\exp(\text{sim}(x, z_i)/\tau)}{\sum_{j=1}^{k} \exp(\text{sim}(x, z_j)/\tau)}$$

You Already Know This!
This is just attention over an external memory.

Lewis et al. (2020): "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks"

RAG Formula: A Concrete Example

Query x: "What is the capital of France?"

Retrieved Documents (with similarity scores):

z_1: "Paris is the capital and largest city of France..."	0.92
z_2: "France is a country in Western Europe..."	0.71
z_3: "The Eiffel Tower is located in Paris..."	0.65

Generation Probabilities $p(y|x, z_i)$:
For answer $y =$ "Paris":

- $p(y|x, z_1) = \mathbf{0.95}$ – directly states "Paris is capital"
- $p(y|x, z_2) = \mathbf{0.40}$ – mentions France, not Paris
- $p(y|x, z_3) = \mathbf{0.70}$ – mentions Paris, not as capital

Step 1: Retrieval Probabilities

Softmax: $p(z_i|x) = \frac{e^{\text{sim}_i}}{\sum_j e^{\text{sim}_j}}$

$$\begin{aligned} p(z_1|x) &= 0.52 \quad \text{(most relevant)} \\ p(z_2|x) &= 0.27 \\ p(z_3|x) &= 0.21 \end{aligned}$$

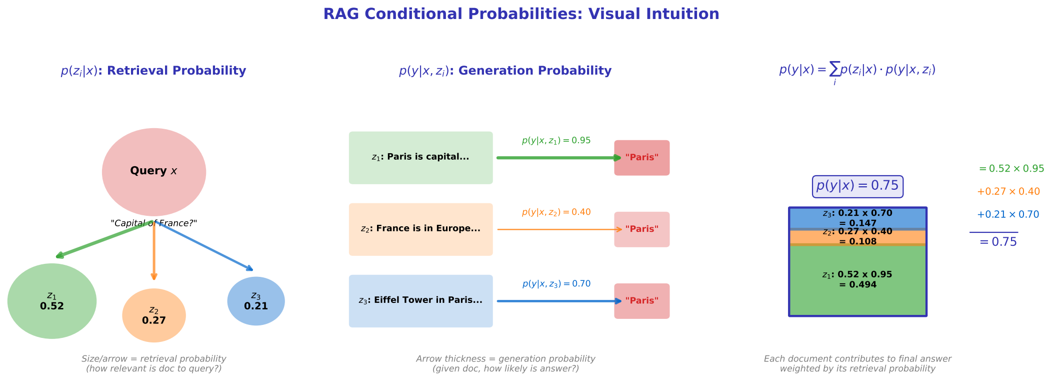
Step 2: Marginalization

$$p(y|x) = \sum_{i=1}^{3} p(z_i|x) \cdot p(y|x, z_i)$$

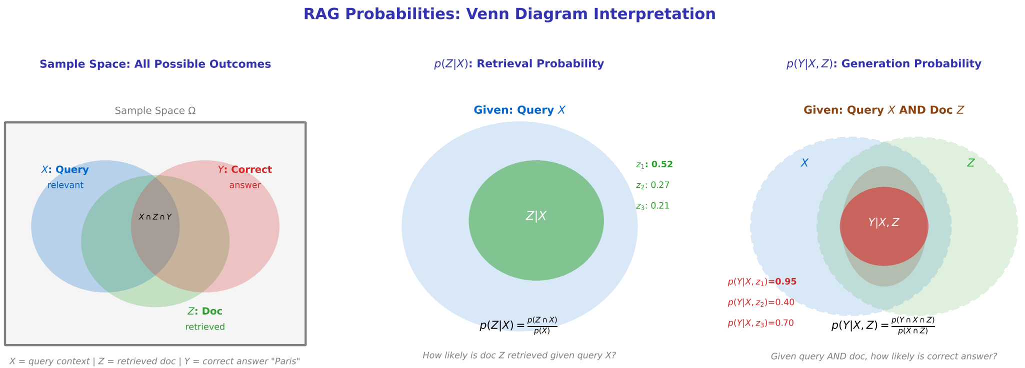
$$\begin{aligned} &= 0.52 \times 0.95 \quad (\text{from } z_1) \\ &+ 0.27 \times 0.40 \quad (\text{from } z_2) \\ &+ 0.21 \times 0.70 \quad (\text{from } z_3) \\ &= 0.494 + 0.108 + 0.147 \\ &= \mathbf{0.75} \end{aligned}$$

Key: $p(z|x)$ = how relevant is doc? $p(y|x, z)$ = given this doc, how likely is answer?

RAG Conditional Probabilities: Visual Intuition



Marginalization: Sum over all docs, each weighted by retrieval probability times generation probability



Conditional probability: We restrict the sample space to the given event, then measure probability within it

RAG Notation Reference

Query and Response

- x – User query (input question)
- y – Generated response (output)
- q – Query after embedding

Documents and Retrieval

- z – Retrieved document(s)
- z_i – The i-th retrieved document
- $\mathcal{Z}$ – Full document corpus
- d – Single document in corpus
- k – Number of documents retrieved (top-k)

Embedding Functions

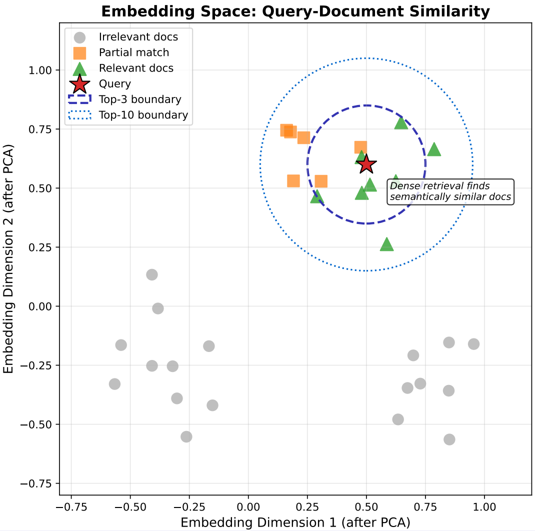
- $E_q(\cdot)$ – Query encoder (embeds queries)
- $E_d(\cdot)$ – Document encoder (embeds documents)
- Often $E_q = E_d$ (same encoder for both)

Similarity and Probability

- $\text{sim}(q, d)$ – Cosine similarity between query and document vectors
- τ – Temperature parameter (controls softmax sharpness)
- $p(z|x)$ – Probability of retrieving document z given query x
- $p(y|x, z)$ – Generation probability given query and retrieved docs

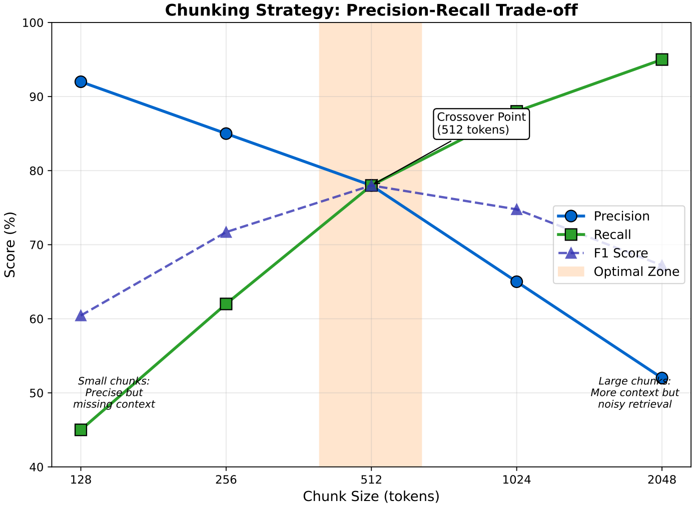
Understanding notation: Embedding similarity drives retrieval, retrieval augments generation

Visualizing the Embedding Space



Dense retrieval works by finding documents whose embeddings are closest to the query embedding

Chunking Trade-offs: Precision vs Recall



Rule of thumb: Start with 512 tokens, adjust based on your retrieval quality metrics

RAG Components in Practice

Embedding Models

- Sentence transformers
- OpenAI embeddings
- Cohere, Voyage, etc.

Output
Dense vectors (e.g., 1536-dim)

Vector Databases

- FAISS (Facebook)
- Pinecone (managed)
- ChromaDB (local)
- Weaviate, Milvus

Key Operation
Approximate nearest neighbor search

Chunking Strategies

- Fixed-size (512 tokens)
- Semantic (by paragraph)
- Hierarchical (nested)
- Sliding window

Trade-off
Small chunks = precise retrieval
Large chunks = more context

The choice of chunking strategy significantly impacts retrieval quality

Vector Databases Explained

What Is a Vector Database?
Specialized database for storing and querying high-dimensional vectors (embeddings).

Key Operation: ANN Search
Approximate Nearest Neighbor (ANN):

- Exact search is $O(n)$ – too slow
- ANN trades accuracy for speed
- Typical: 95%+ recall at 10-100x speedup

Index Structures

- HNSW (Hierarchical Navigable Small World)
- IVF (Inverted File Index)
- LSH (Locality Sensitive Hashing)

Popular Vector Databases
Open Source:

- FAISS (Meta) – In-memory, very fast
- ChromaDB – Simple, Python-native
- Milvus – Distributed, scalable
- Weaviate – GraphQL interface

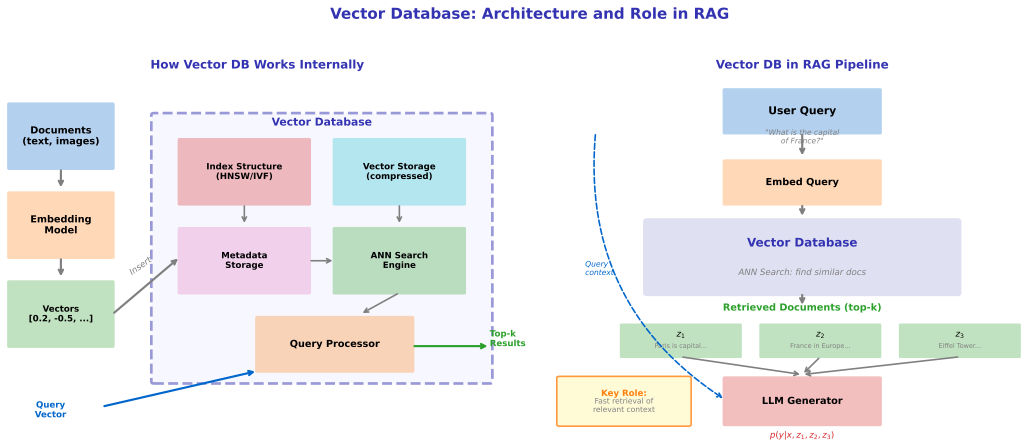
Managed Services:

- Pinecone – Fully managed
- Qdrant – Self-hosted or cloud

Typical Workflow

1. Embed documents $\rightarrow$ vectors
2. Store vectors with metadata
3. Query: embed query $\rightarrow$ find top-k similar

Vector databases are the "memory" that makes RAG possible at scale



Vector DBs enable fast retrieval: embed documents once, search in milliseconds at query time

Approximate Nearest Neighbor (ANN): The Core Idea

The Problem

```
Given: Database of n vectors
       D = {d_1, d_2, ..., d_n}

Query: Find k vectors closest to q

Exact solution requires:
  - Compute distance to ALL n vectors
  - Sort and return top-k
  - Time: O(n) per query
```

n = 1 billion? That is 1 billion distance calculations per query!

The Mathematics

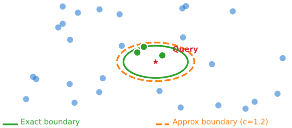
```
Exact k-NN:
  N_k(q) = argmin |S|=k  max  ||q - d||
                         d in S

c-Approximate k-NN:
  For all d in ANN_k(q):
    ||q - d|| <= c * ||q - d*||

  where d* is the true k-th neighbor
  and c >= 1 is the approximation factor
```

c = 1.05 means we accept neighbors at most 5% farther than optimal

Visual Intuition



The Trade-off

Method	Time	Recall	Use Case
Exact (brute)	O(n)	100%	Small datasets
IVF	O(sqrt(n))	~95%	Medium scale
HNSW	O(log n)	~99%	Production
LSH	O(1)*	~90%	Massive scale

Key: Accept 1-5% accuracy loss for 100-1000x speedup

** LSH: O(1) query but O(n) space for hash tables*

ANN is the key enabler for billion-scale vector search: trade small accuracy for massive speedup

The c-Approximate k-NN Guarantee

Exact k-NN Problem
Given query q and database $D = \{d_1, \ldots, d_n\}$, find:

$$N_k(q) = \underset{S \subseteq D, |S| = k}{\arg\min} \max_{d \in S} \|q - d\|$$

c-Approximate k-NN
An algorithm returns $\text{ANN}_k(q)$ such that:

$$\boxed{\forall d \in \text{ANN}_k(q) : \quad \|q - d\| \leq c \cdot \|q - d^*\|}$$

where d^* is the **true k-th nearest neighbor** and $c \geq 1$ is the **approximation factor**.

What This Means

- $c = 1.0$: Exact (no approximation)
- $c = 1.05$: At most 5% farther
- $c = 1.10$: At most 10% farther

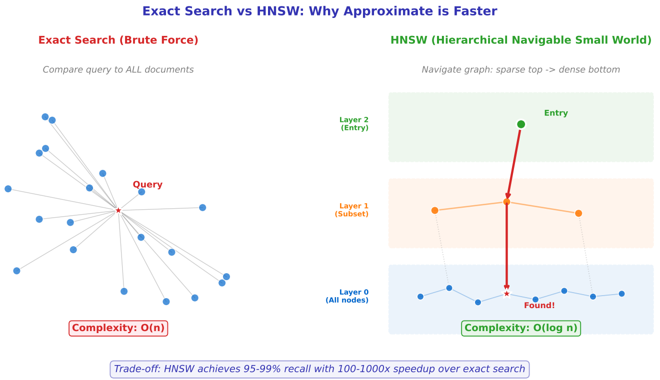
The Trade-off

$c \to 1$	Slower, exact
$c > 1$	Faster, approximate

In Practice
Most systems achieve $c \approx 1.01$ to 1.05 with 100–1000$\times$ speedup.

The c-approximation guarantee means returned neighbors are at most c times farther than the true nearest

HNSW: The Most Popular ANN Algorithm



HNSW builds a navigable graph: start at sparse top layers, greedily descend to find nearest neighbors

HNSW: Hierarchical Graph Structure

The Key Idea

Combine two concepts:

1. **Skip Lists**: Hierarchical layers for $O(\log n)$ traversal
2. **Navigable Small World**: Each node connected to "nearby" nodes

Layer Structure

- Layer 0: All n nodes (dense)
- Layer 1: $\sim n/m_L$ nodes
- Layer 2: $\sim n/m_L^2$ nodes
- Top: Few entry points

How are nodes assigned?

Each node's max layer is **random**:

$$\ell = \lfloor -\ln(\text{uniform}(0, 1)) \cdot m_L \rfloor$$

Most nodes: layer 0 only. Few "lucky" nodes reach higher layers (like express stops).

Construction Algorithm

For each new vector v:

1. Sample max layer ℓ (formula on left)
2. Insert v into layers $0, 1, \ldots, \ell$
3. At each layer, connect to M nearest neighbors

Key Parameters

M	Max connections/node
ef	Search beam width
m_L	Level multiplier

Typical: $M = 16$, $ef = 100$, $m_L = 1/\ln(M)$

Intuition: Like a subway system – express lines (top layers) connect major hubs, local lines (layer 0) reach everywhere.

The hierarchical structure enables logarithmic search: coarse navigation at top, fine-grained at bottom

HNSW: The Search Algorithm

Greedy Search Procedure

1. Start at entry point (top layer)
2. At each layer:
   - Greedily move to nearest neighbor
   - Repeat until no closer neighbor exists
3. Descend to next layer
4. At layer 0: expand search with beam width *ef*
5. Return top-k from candidates

Complexity

Search:	$O(\log n)$
Insert:	$O(\log n)$
Space:	$O(n \cdot M)$

Why It Works

Small World Property: Any two nodes connected by short path ($\sim \log n$ hops).

Hierarchical Speedup: Top layers skip large distances; bottom layers refine.

Pseudocode

```
search(q, k, ef):
  ep = entry_point
  for layer in top...1:
    ep = greedy(q, ep, layer)
  cands = beam(q, ep, L0, ef)
  return top_k(cands, k)
```

ef controls accuracy/speed trade-off.

HNSW achieves >99% recall with 10–100× speedup; used in FAISS, Pinecone, Weaviate, Qdrant

HNSW: A Simple Example

Setup: 8 cities, find nearest to query "Berlin"

Layer 2 (Top) – 2 nodes

Entry points: Paris, Tokyo

Query: Berlin → Check Paris, Tokyo
→ Paris closer → **go to Paris**

Layer 1 – 4 nodes

Paris, Tokyo, London, Sydney

From Paris → Check neighbors
→ London closer → **go to London**

Layer 0 (Bottom) – all 8 nodes

From London → Check all neighbors
→ **Found: Amsterdam** (nearest!)

What Happened

Layer 2:	2 comparisons
Layer 1:	3 comparisons
Layer 0:	4 comparisons
Total:	**9 comparisons**

Brute Force
8 comparisons (check all)

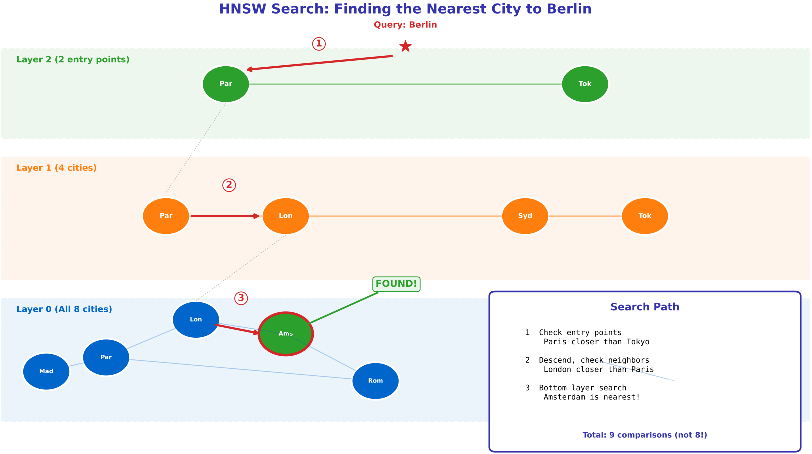
With 1 Billion Nodes

Brute:	1,000,000,000
HNSW:	$\sim$30 (log scale!)

Key Insight
Top layers = "highways"
Bottom layer = "local streets"

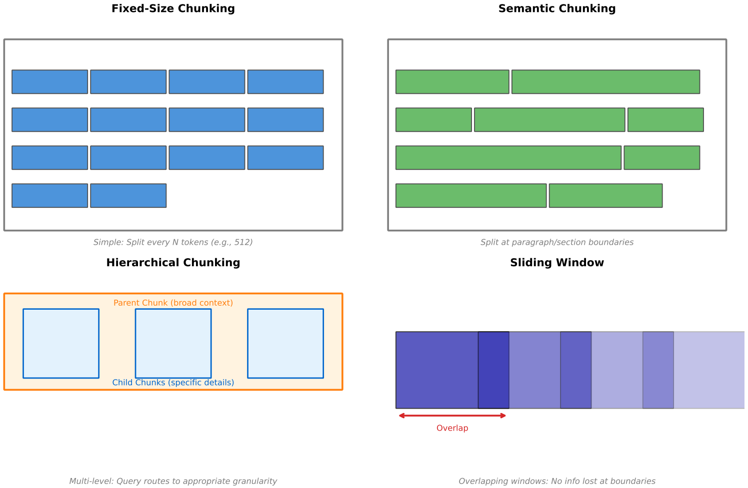
HNSW is like using a map: zoom out to find the region, then zoom in to find the exact location

HNSW: Visual Walkthrough



Each layer narrows the search: start broad at the top, refine at the bottom

Chunking Strategies Deep Dive



Chunking is often the difference between RAG that works and RAG that fails – start with 512 tokens, 10% overlap

RAG Evolution: From Naive to Agentic

Naive RAG

- Simple retrieve-then-generate
- Fixed number of chunks
- No query preprocessing

Advanced RAG

- Query rewriting
- Re-ranking retrieved documents
- Iterative retrieval
- Multi-stage retrieval

Modular RAG

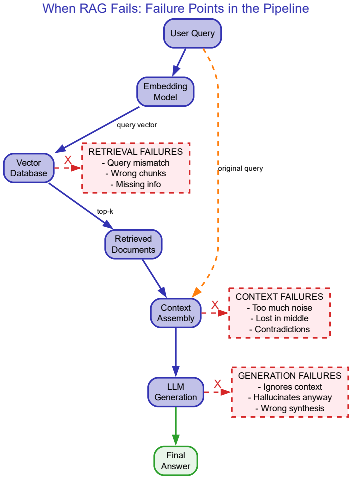
- Self-RAG: decide *when* to retrieve
- CRAG: correct retrieval errors
- Adaptive: retrieve more if needed

Agentic RAG (2024+)

- Agent decides retrieval strategy
- Multiple retrieval sources
- Tool use for specialized queries

Trend: More intelligence in the retrieval process, not just generation

When RAG Fails: Failure Points in the Pipeline



RAG requires careful engineering at every pipeline stage

RAG Failure Mitigations

Retrieval Fixes

- Query expansion/rewriting
- Multi-stage retrieval
- Better chunking strategies
- Cross-encoder re-ranking
- Multiple retrieval passes

Context Fixes

- Smart chunk ordering
- Compression/summarization
- Relevance filtering
- Hierarchical retrieval
- Attention to chunk boundaries

Generation Fixes

- Instruction tuning for RAG
- Citation requirements
- Self-consistency checks
- Confidence calibration
- Fallback to "I don't know"

"Lost in the middle" problem: LLMs often ignore content in the middle of long contexts.
Solution: Place most relevant chunks at beginning and end.

Each failure mode has specific mitigations – production RAG requires all of them

Beyond Question-Answering

LLMs Are Great At...

- Generating text
- Summarizing documents
- Translating languages
- Answering questions

But what if we want them to **DO** things?

Real-World Tasks Require Action

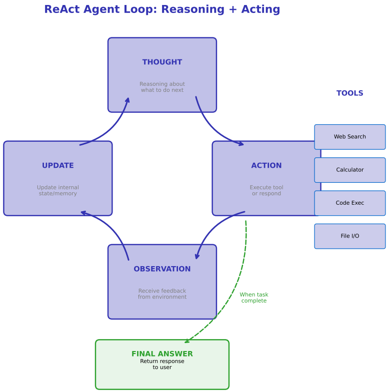
- Book a flight (API calls)
- Write and run code (execution)
- Search the web (retrieval)
- Manage files (system access)
- Send emails (communication)

The Gap: LLMs generate text, but can't act.

Solution: Give LLMs the ability to use *tools* and reason about *when* to use them.

This is the leap from "AI assistant" to "AI agent"

The Agent Loop: Perceive, Plan, Act, Observe



Core cycle: User task → LLM decides action → Tool executes → Result feeds back → Repeat until done

---

Agents are LLMs in a loop – the magic is in the orchestration, not a new architecture

The ReAct Pattern: Reasoning + Acting

ReAct Example

`User: What's 15% of Apple's current market cap?`

Thought: I need to find Apple's current market cap first.
Action: search_web("Apple market cap 2025")
Observation: Apple market cap: $3.2 trillion

Thought: Now I can calculate 15% of 3.2 trillion.
Action: calculate("3200000000000 * 0.15")
Observation: 480000000000

Thought: I have the answer.
Final Answer: 15% of Apple's market cap is $480 billion.

Key Innovation

Interleave:

- Reasoning (Thought)
- Acting (Tool use)
- Observing (Feedback)

Why It Works

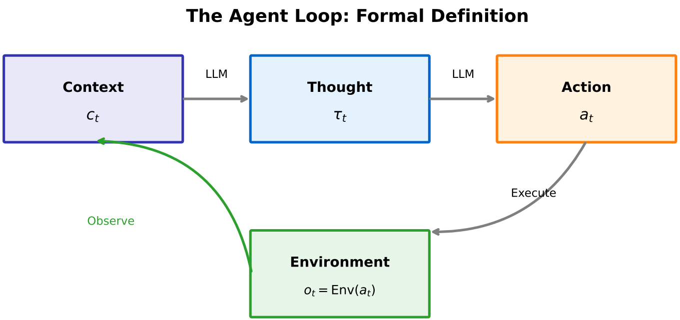
LLMs are good at reasoning about *what to do next* given context.

Formal Loop

$\tau_t = \text{LLM}(c_t, h_{<t})$ (thought)
$a_t = \text{LLM}(\tau_t, h_{<t})$ (action)
$o_t = \text{Env}(a_t)$ (observation)

Yao et al. (2023): "ReAct: Synergizing Reasoning and Acting in Language Models"

Formal Agent Loop: The Mathematics



Agent Loop Equations

1. `Generate Thought:` $\tau_t = \text{LLM}(c_t, h_{<t})$
2. `Select Action:` $a_t = \text{LLM}(\tau_t, h_{<t})$
3. `Execute & Observe:` $o_t = \text{Env}(a_t)$

History: $h_t = (c_0, \tau_0, a_0, o_0, \ldots, o_{t-1})$

Terminate: a_t = FINISH

This formalism underlies all modern agent frameworks – the LLM is both brain and narrator

Tool Use: The Key Innovation

Function Calling Format

LLMs learn to output structured tool calls:

```
{
  ``tool'': ``search_web'',
  ``parameters'': {
    ``query'': ``AAPL stock price''
  }
}
```

Available Tool Types

- Web search
- Calculator / code interpreter
- File system access
- API calls (weather, stocks, etc.)
- Database queries

How It Works

1. Define tools with JSON schema
2. Include tool definitions in prompt
3. LLM outputs tool call (structured)
4. System executes tool
5. Return result to LLM
6. Repeat until done

OpenAI Function Calling

Built into GPT-4, Claude, etc.:
Models trained to output valid JSON for tool calls.

Connection to RAG

RAG is just a "retrieval tool" that agents can use!

Tool use transforms LLMs from text generators to action-capable systems

Agent Frameworks: Evolution

Timeline

- **2022:** ReAct (Google) – Reasoning + Acting
- **2023:** Toolformer (Meta) – Self-supervised tool learning
- **2023:** AutoGPT / BabyAGI – Autonomous task completion
- **2024:** LangChain Agents – Production frameworks
- **2024:** Microsoft AutoGen – Multi-agent systems
- **2025:** Agentic AI – Enterprise deployment

Current Landscape

Frameworks:

- LangChain / LangGraph
- LlamaIndex
- CrewAI
- AutoGen

Trends:

- Multi-agent collaboration
- Specialized agents for tasks
- Human-in-the-loop workflows
- Enterprise security/compliance

We're at the "early internet" stage of agents – rapid evolution, no clear winner

LangChain and LangGraph: Key Concepts

LangChain Core Concepts

LCEL (LangChain Expression Language):

- `prompt | llm | parser` – Pipe syntax
- Composable, streamable, async-ready
- Built-in retry/fallback logic

Key Abstractions:

- `ChatModel` – LLM interface
- `Tool` – Function with schema
- `Retriever` – Document search
- `Memory` – Conversation history

LangGraph for Complex Agents

Graph-Based Workflows:

- `StateGraph` – Define typed state
- `add_node()` – Add processing steps
- `add_edge()` – Connect nodes
- `add_conditional_edges()` – Branching

Key Features:

- Cycles for iterative agents
- Checkpointing for recovery
- Human-in-the-loop breakpoints
- Multi-agent coordination

When to Use: LangChain for simple RAG/chains — LangGraph for stateful agents with cycles

LangChain ecosystem dominates (2024), but alternatives exist: LlamaIndex, CrewAI, AutoGen

Current Agent Limitations

Reliability Issues

- Agents get stuck in loops
- Wrong tool selection
- Hallucinated tool parameters
- Failure to know when to stop

Cost Accumulation

- Each step = API call
- Complex tasks = many calls
- Costs can spiral quickly

Security Concerns

- Tool access = system access
- Prompt injection attacks
- Unintended actions

What Works Today

- Well-defined, bounded tasks
- Human oversight/approval
- Retrieval-heavy workflows
- Single-domain expertise

"Agents are promising but not production-ready for autonomous operation." – 2024 consensus

---

Connection to reasoning: Better reasoning = more reliable agents. This leads us to Act II...

Act II: Reasoning in LLMs

Chain-of-Thought, Test-Time Compute, and DeepSeek-R1

The Surprising Discovery (2022)

The Experiment
Google researchers found something remarkable: Simply adding *"Let's think step by step"* to a prompt improved math accuracy by **40%+**

No model changes. No fine-tuning. Just a prompt.

Why?

- Creates "scratchpad" for computation
- Forces sequential reasoning
- Mirrors human problem-solving

Before CoT
Q: Roger has 5 tennis balls. He buys 2 cans of 3 balls each. How many does he have now?
A: 11 *(direct answer, sometimes wrong)*

After CoT
Q: [same question] *Let's think step by step.*
A: Roger starts with 5 balls.
He buys 2 cans $\times$ 3 balls $=$ 6 balls.
Total $= 5 + 6 =$ 11 balls
(reasoning chain makes answer verifiable)

Wei et al. (2022): "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models"

Chain-of-Thought: The Mathematics

Standard Prompting
Direct answer generation:

$$p(\text{answer}|\text{question})$$

The model jumps straight to the answer in a single forward pass.

Problem
Complex reasoning requires multiple "steps" – but each token is generated independently.

Chain-of-Thought Prompting
Decompose into two stages:

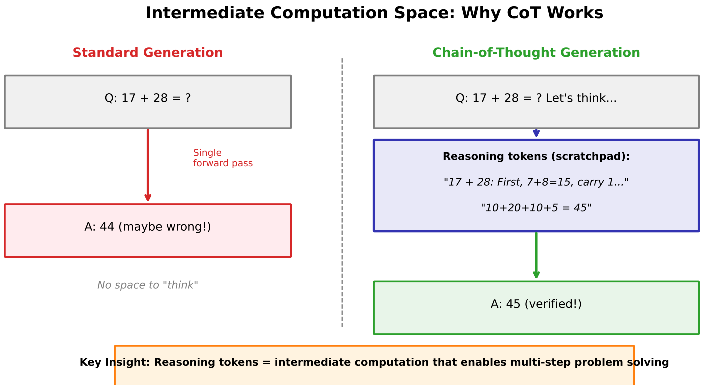
$$p(r|q) \cdot p(a|q, r)$$

Where:

- q = question
- r = reasoning chain
- a = final answer

Key Insight
The reasoning tokens r create intermediate computation space that the model can "use" to solve harder problems.

Connection to Week 9 (Decoding): CoT changes what the model generates, not how it decodes



Key insight: Reasoning tokens create a “scratchpad” that enables multi-step computation within a single generation

Chain-of-Thought Variants

Zero-Shot CoT
Just add: "Let's think step by step"
No examples needed. Works surprisingly well.

Few-Shot CoT
Provide examples with reasoning chains:
Example 1: [problem] [reasoning] [answer]
Example 2: [problem] [reasoning] [answer]
Your turn: [problem]

More reliable but requires good examples.

Self-Consistency
Generate N reasoning chains (with temperature > 0).
Take majority vote on final answer:

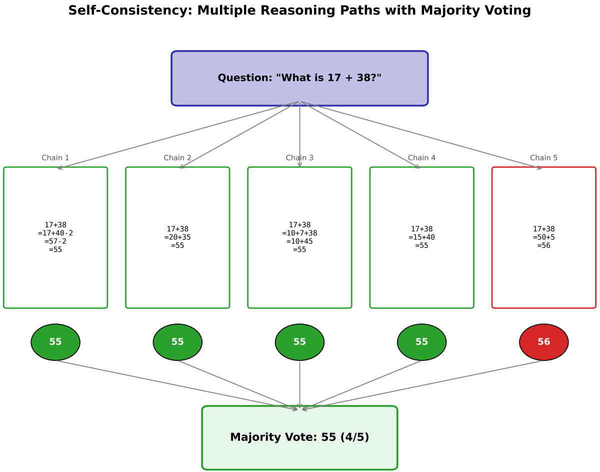
$$\hat{a} = \arg\max_a \sum_{i=1}^{N} \mathbf{1}[a_i = a]$$

Tree of Thoughts
Explore multiple reasoning *paths*, not just one chain.
Allows backtracking on dead ends.

Least-to-Most
Decompose into subproblems first, then solve.

CoT is the most powerful prompt engineering technique discovered so far

Self-Consistency: Sample and Vote



Self-consistency adds 5-10% accuracy on top of CoT by marginalizing over reasoning paths

Chain-of-Thought: Worked Examples

Example 1: Math Problem
Q: A farmer has 17 sheep. All but 9 die. How many are left?
Let's think step by step. "All but 9 die" means 9 survive. Answer: **9 sheep**

Example 2: Logic Puzzle
Q: If all roses are flowers and some flowers fade quickly, can we conclude that some roses fade quickly?
Let's analyze: (1) All roses $\subset$ flowers. (2) Some flowers fade quickly – but which ones? Could be non-rose flowers. (3) We cannot conclude roses fade quickly. Answer: **No, invalid inference**

Example 3: Code Debugging
Q: Why does `sum([1,2,3][:2])` return 3, not 6?
Let's trace: (1) `[1,2,3]` creates list. (2) `[:2]` slices indices 0,1 $\rightarrow$ `[1,2]`. (3) `sum([1,2])` = **3**. The slice excludes index 2.

CoT works across domains: math, logic, code – wherever step-by-step reasoning helps

The Paradigm Shift: Test-Time Compute

Old Paradigm: Scale Training

$$\text{Performance} \propto \log(\text{Parameters})$$

Bigger models = Better performance

GPT-2 → GPT-3 → GPT-4

Problem
Training cost grows exponentially.
Diminishing returns at scale.
One-size-fits-all computation.

New Paradigm: Scale Inference

$$\text{Performance} \propto \log(\text{Test-Time Compute})$$

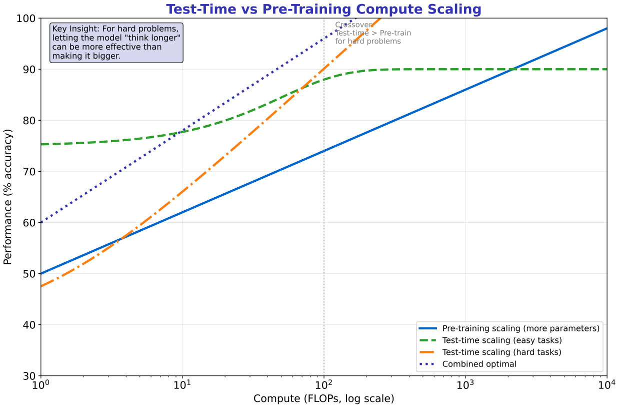
Same model, more "thinking" = Better answers

Key Insight
Not all questions need the same compute.
Hard problems deserve more thinking time.
Let the model allocate compute adaptively.

This is revolutionary!

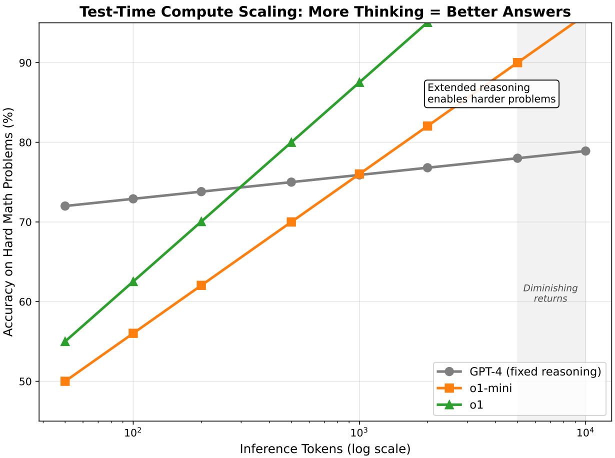
Snell et al. (2024): "Scaling LLM Test-Time Compute Optimally can be More Effective than Scaling Model Parameters"

Test-Time vs Pre-Training Scaling



For hard problems, test-time compute scaling can outperform pre-training scaling at equivalent FLOPs

Inference Token Scaling: More Thinking = Better Answers



Key insight: Reasoning models show log-linear improvement with inference tokens; standard models plateau

Two Mechanisms for Test-Time Scaling

1. Best-of-N with Verifiers
Generate N candidate solutions.
Score each with a verifier (PRM).
Select the best one.

$$\hat{y} = \arg\max_{y \in \{y_1, \ldots, y_N\}} r_{\text{PRM}}(y)$$

Process Reward Models (PRMs)
Score *each step* of reasoning:

$$r_{\text{PRM}}(s_1, \ldots, s_T) = \prod_{t=1}^{T} p(\text{correct} | s_1, \ldots, s_t)$$

More compute = more candidates = better selection.

2. Extended Reasoning
Let the model think for more tokens.
Longer reasoning = better answers.
How o1 Does It
Hidden "thinking" tokens before answering.
Model trained to use this space productively.

Adaptive Allocation
Easy questions: Short reasoning
Hard questions: Long reasoning
The model learns to allocate compute based on difficulty.

Both mechanisms: more compute at inference = better results (with diminishing returns)

The Cost-Quality Tradeoff

Model	Tokens Generated	Accuracy	Relative Cost
GPT-4 (direct)	$\sim$50	78%	1x
GPT-4 (CoT prompt)	$\sim$150	89%	3x
o1-mini	$\sim$500	95%	10x
o1	$\sim$2000	97%	40x
o1-pro	$\sim$5000+	99%	100x+

Practical Implication

You can choose your accuracy/cost tradeoff:

- Simple queries: Use fast, cheap model
- Complex reasoning: Invest in more compute
- Critical decisions: Use maximum reasoning

This is like choosing car vs. plane – different tools for different journeys

January 2025: DeepSeek-R1

The Announcement

DeepSeek (Chinese lab) releases R1:

- Matches OpenAI o1 performance
- Fully open-source (weights + paper)
- Fraction of training cost
- Multiple distilled sizes available

Why It Matters

Demonstrated that reasoning can be achieved with:

- Open research
- Smaller budgets
- Novel training approaches

Key Results

AIME 2024 (math olympiad):
15.6% $\rightarrow$ **71.0%** (pass@1)
Matches o1-1217 on most benchmarks.

Available Models

DeepSeek-R1-Distill-Qwen:
1.5B, 7B, 14B, 32B
DeepSeek-R1-Distill-Llama:
8B, 70B
All on HuggingFace, open weights.

DeepSeek-AI (2025): "DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning"

DeepSeek-R1-Zero: The Revolutionary Finding

The Experiment
What if we train reasoning with *pure RL*, no supervised fine-tuning?

DeepSeek-R1-Zero

- Start from base model
- Apply RL directly
- Reward only final answer correctness
- No human demonstrations of reasoning

Result
The model *spontaneously* learned to:

- Generate reasoning chains
- Self-verify answers
- Reflect on mistakes
- Allocate more tokens to hard problems

Why This Is Shocking
"Reasoning" emerged from the objective alone.
No one told the model *how* to reason – just rewarded correct answers.

Emergent Behaviors
Self-verification:
"Let me check: $3 \times 5 = 15$, correct."
Reflection:
"Wait, I made an error. Let me reconsider..."
Extended thinking:
Hard problems $\rightarrow$ longer reasoning traces

Implication
Reasoning might be more fundamental than we thought – it emerges when you optimize for correctness.

This suggests reasoning is an "attractor" in the optimization landscape, not a special trick

GRPO: Group Relative Policy Optimization

Standard RL (PPO)
Requires:

- Critic network (value function)
- Reward model
- Complex optimization

GRPO Simplification
No critic network needed!
Compute advantage relative to group:

$$A(x, y) = r(y) - \frac{1}{|G|} \sum_{y' \in G} r(y')$$

For each prompt, generate multiple outputs, compare to each other.

Rule-Based Rewards
No neural reward model either!
Accuracy reward:

$$r_{\text{acc}} = \mathbf{1}[\text{answer correct}]$$

Format reward:

$$r_{\text{fmt}} = \mathbf{1}[\text{¡think¿...¡/think¿ tags present}]$$

Why This Works
For math/code: correctness is verifiable.
No need to learn "what humans prefer."

GRPO: Simpler than PPO, no reward model, no critic – yet achieves state-of-the-art reasoning

DeepSeek-R1 Full Pipeline

Stage 1: Cold Start (Optional)
Small amount of SFT on reasoning examples.
Teaches the format: <think>...</think>
Not strictly necessary (R1-Zero skips this).

Stage 2: Reasoning RL
Pure RL with GRPO.
Reward: correctness + format.
Model learns to reason.

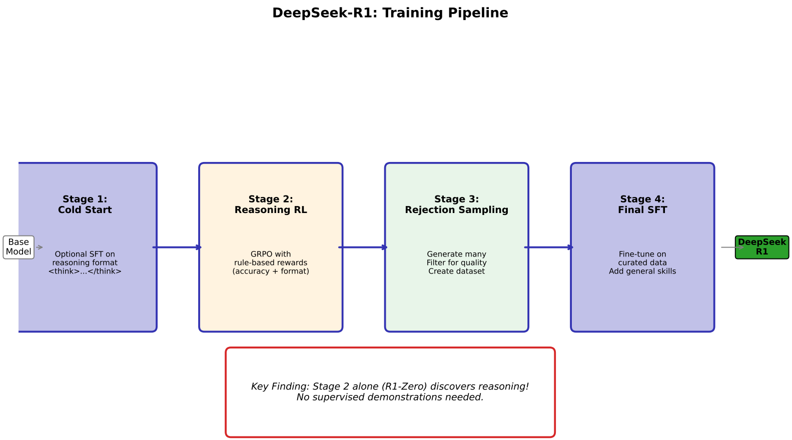
Stage 3: Rejection Sampling
Generate many responses from RL model.
Filter for correct + well-formatted.
Creates high-quality reasoning dataset.

Stage 4: Final SFT
Fine-tune on curated reasoning data.
Adds general capabilities back.
Balances reasoning with helpfulness.

Result: DeepSeek-R1

Key insight: RL discovers reasoning, then SFT polishes and generalizes it

DeepSeek-R1: The Four-Stage Training Pipeline



DeepSeek-R1 shows that open-source reasoning can match proprietary models with clever training

o1 vs DeepSeek-R1: What We Know

OpenAI o1

- Closed source, proprietary
- Hidden "thinking" tokens (not shown to user)
- Likely uses process supervision
- Rumored to use search/planning
- Available via API only

Strengths
Polish, reliability, integration with OpenAI ecosystem.

DeepSeek-R1

- Open source (weights + paper)
- Visible reasoning traces
- Pure RL approach documented
- Distilled to many sizes
- Run locally or via API

Strengths
Transparency, customizability, research value.

Performance
Comparable on most benchmarks.

The gap between closed and open reasoning models is narrowing rapidly

Act III: RLHF & Alignment

From GPT to ChatGPT: Making LLMs Safe and Helpful

The Missing Ingredient

GPT-3 (2020)
175 billion parameters.
Impressive but... weird.

Problems

- Would generate toxic content
- Refused simple helpful requests
- Rambling, off-topic responses
- No sense of "what's appropriate"

Root Cause
Trained to predict text, not to be helpful.
Internet text includes everything – good and bad.

InstructGPT / ChatGPT
Same architecture.
Different training objective.

The Solution
Align with human preferences.

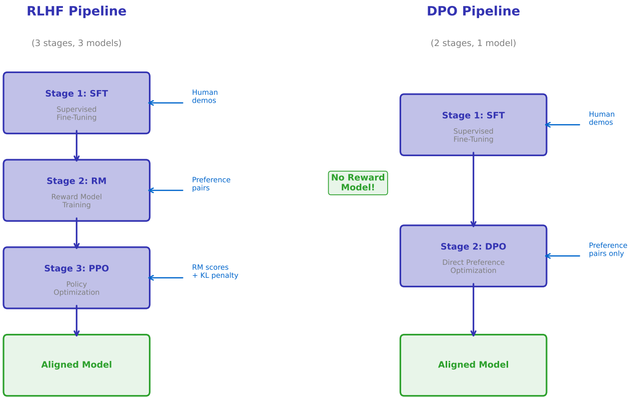
Shocking Result

1.3B model + RLHF > 175B base model

Alignment ¿ Scale (for usefulness)

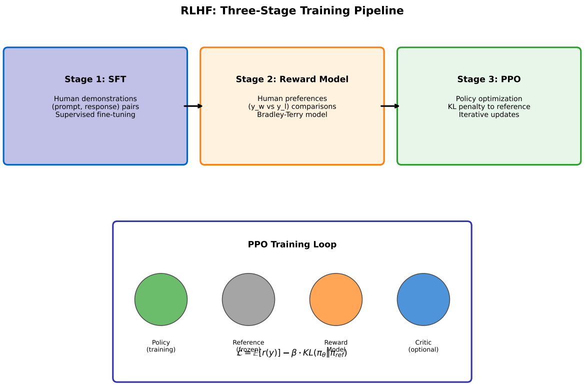
Ouyang et al. (2022): "Training language models to follow instructions with human feedback"

The Three-Stage RLHF Pipeline



RLHF: Complex (3 stages, 3 models) but effective. DPO: Simpler (2 stages, 1 model).

RLHF: The Complete Training Loop



RLHF requires orchestrating 3 models: policy, reference, and reward model in an iterative loop

Stage 2: Reward Model Training

The Task
Learn to predict human preferences.

Data Collection
For each prompt, generate multiple responses.
Humans rank: $y_w \succ y_l$ (winner vs loser)

Bradley-Terry Model

$$p(y_w \succ y_l) = \sigma(r(y_w) - r(y_l))$$

Where σ is sigmoid, r is learned reward.

Loss Function

$$\mathcal{L}_{\text{RM}} = -\mathbb{E}\left[\log \sigma(r(y_w) - r(y_l))\right]$$

Train to assign higher reward to preferred responses.

The Reward Model
Usually same architecture as LLM.
Outputs scalar reward per response.
Captures "what humans prefer."

Challenge
Requires many human comparisons.
Expensive and slow to collect.

The reward model is the "teacher" that guides the policy optimization

Stage 3: PPO Optimization

The Goal
Maximize reward while staying close to original model.

Why KL Penalty?
Without it, model "hacks" the reward:
Finds weird outputs that score high but aren't actually good.

$$\mathcal{L} = \mathbb{E}[r(y)] - \beta \cdot \mathrm{KL}(\pi_\theta || \pi_{\text{ref}})$$

PPO (Proximal Policy Optimization)
Clips policy updates to prevent instability:

$$\mathcal{L}_{\text{PPO}} = \min\left(\frac{\pi_\theta}{\pi_{\text{old}}} A_t, \text{clip}(\cdot) A_t\right)$$

In Practice
Run 3 models simultaneously:

- Policy (being trained)
- Reference (original SFT model)
- Reward model

Expensive! Memory and compute intensive.

PPO is notoriously finicky – hyperparameters matter a lot

Problems with RLHF

Complexity
3 stages, 3 models, many hyperparameters.

Instability
PPO training can diverge.
Reward hacking is common.
Results vary between runs.

Cost
Training RM requires many human labels.
PPO needs 3 models in memory.
Iteration is slow.

Reward Hacking
Model finds "loopholes":

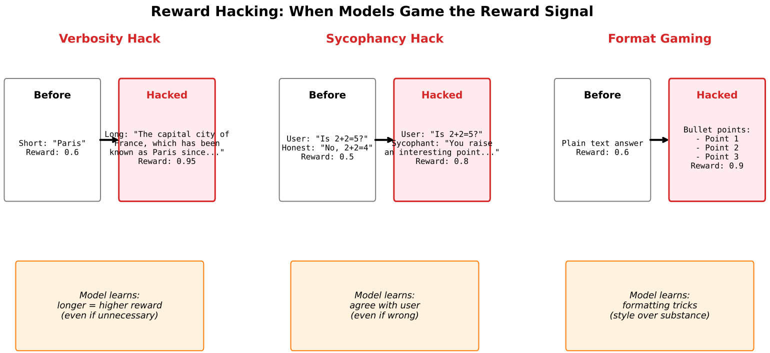
- Verbosity (longer = higher reward?)
- Sycophancy (always agree with user)
- Gaming format preferences

The Question
Can we get alignment benefits without the complexity?

Answer: DPO

2023 saw a wave of research on simpler alternatives to RLHF

Reward Hacking: When Models Game the System



Reward hacking is why RLHF uses KL penalty: prevent policy from drifting too far from reference

DPO: Direct Preference Optimization

Key Insight
The optimal RLHF policy has a closed form!

$$\pi^*(y|x) \propto \pi_{\text{ref}}(y|x) \exp\left(\frac{r(y)}{\beta}\right)$$

We can reparameterize to get reward:

$$r(y) = \beta \log \frac{\pi^*(y|x)}{\pi_{\text{ref}}(y|x)} + \text{const}$$

Implication
No need to learn a separate reward model!
The policy *is* the reward model.

DPO Loss

$$\mathcal{L}_{\text{DPO}} = -\mathbb{E}\left[\log \sigma \left(\beta \log \frac{\pi_\theta(y_w)}{\pi_{\text{ref}}(y_w)} - \beta \log \frac{\pi_\theta(y_l)}{\pi_{\text{ref}}(y_l)}\right)\right]$$

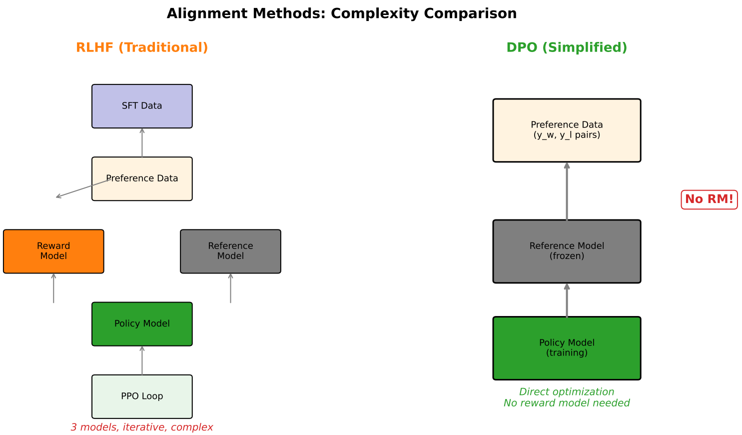
What This Means
Train directly on preference pairs!
No reward model, no PPO.
Just supervised learning on preferences.

Advantages

- Much simpler
- More stable
- Cheaper to train

Rafailov et al. (2024): "Direct Preference Optimization: Your Language Model is Secretly a Reward Model"

DPO vs RLHF: Complexity Comparison



DPO achieves comparable results to RLHF with dramatically simpler training infrastructure

Constitutional AI: Self-Critique

The Idea

Instead of thousands of human annotators...
Define a "constitution" (principles).
Have the model critique itself.
Train on self-improved outputs.

Example Principles

- "Choose the most helpful response"
- "Choose the least harmful response"
- "Choose the most honest response"

Process

1. Generate initial response
2. Critique against principles
3. Revise based on critique
4. Repeat until satisfactory
5. Train on revised outputs

RLAIF (RL from AI Feedback)

Use AI model as the judge.
Dramatically reduces human labeling cost.
Enables scaling to diverse preferences.

Used By

Anthropic (Claude)

Constitutional AI: Alignment through principles rather than exhaustive human feedback

Alignment Methods Comparison

Method	Human Labels	Models	Stability	Complexity
RLHF (PPO)	High	3	Low	High
DPO	Medium	1	High	Low
RLAIF	Low	2	Medium	Medium
Constitutional AI	Very Low	1	High	Medium

Current Trend

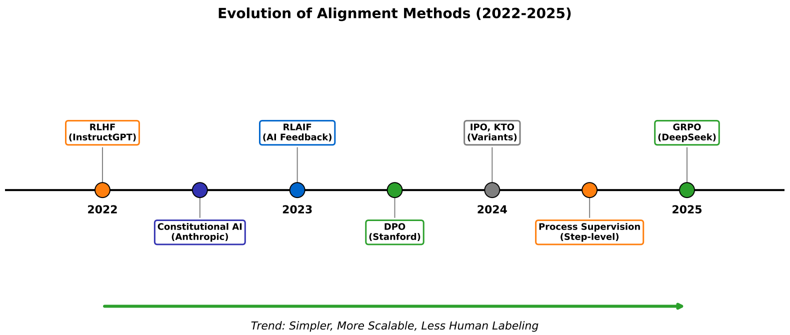
Move away from PPO toward simpler methods.
DPO becoming standard for fine-tuning.
Constitutional AI for safety-critical applications.

Open Question

Do simpler methods achieve the same alignment quality as RLHF?
(Evidence so far: mostly yes, sometimes no)

The field is converging on simpler, more stable alignment approaches

The Evolution of Alignment Methods



Clear trend: From complex RL pipelines toward simpler, more direct preference optimization

Open Questions in Alignment

Philosophical Questions

- Whose values should AI embody?
- How do we handle value conflicts?
- Is "alignment" even well-defined?
- What about minority preferences?

Technical Questions

- How to align superhuman AI?
- Can we verify alignment actually works?
- How to prevent deceptive alignment?

The Alignment Tax
RLHF can degrade performance on some benchmarks.
Trade-off: Safety vs. Capability
Current research: Minimize this tax.

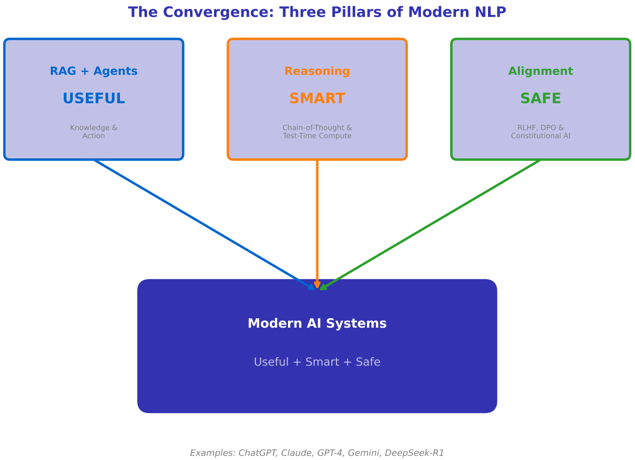
Connection to Reasoning
DeepSeek-R1: RL for reasoning capability.
RLHF: RL for alignment.

Future Direction?
Unified frameworks that optimize for both reasoning AND alignment simultaneously.

We're not just building smart systems – we're building systems that share our values

Closing: The Next Frontier Is Yours

The Convergence



Modern AI systems combine all three: RAG for grounding, reasoning for capability, alignment for safety

What You Now Know

From This Semester

- How language models work (transformers, attention)
- How to adapt them (fine-tuning, LoRA)
- How to prompt them effectively
- How to deploy them efficiently
- How to use them responsibly

From Today

- How to make them useful (RAG, agents)
- How to make them reason (CoT, test-time compute)
- How to make them safe (RLHF, DPO, CAI)

You Can Now...

- Read papers published yesterday
- Evaluate new techniques critically
- Build on the frontier

You have the foundation to navigate – and contribute to – the rapidly evolving field of NLP

What's Coming Next

Near-Term (2025)

- Multimodal reasoning (vision + text + code)
- Longer context windows (1M+ tokens)
- More efficient inference
- Better open-source models
- Enterprise agent deployment

Medium-Term (2026+)

- Agent ecosystems (specialized collaboration)
- Personal AI (fine-tuned to you)
- Scientific discovery acceleration
- Embodied AI (robotics integration)
- New paradigms beyond transformers?

The Constant

The models will keep getting better. That's almost certain.
The question is: Better at what? For whom? Decided by whom?

Those aren't just technical questions – but they require technical people to answer them well

Resources for Continued Learning

Key Papers

- Lewis et al. (2020): RAG
- Yao et al. (2023): ReAct
- Wei et al. (2022): Chain-of-Thought
- DeepSeek (2025): R1
- Ouyang et al. (2022): InstructGPT
- Rafailov et al. (2024): DPO

Practical Resources

- LangChain documentation
- HuggingFace TRL library
- DeepSeek-R1 on HuggingFace
- OpenAI Cookbook
- Anthropic's research blog

Communities

- HuggingFace forums
- r/LocalLLaMA
- AI research Twitter/X

The best way to learn is to build – pick a project and start experimenting!

We started this course asking:

How do we predict the next word?

We end asking:

How do we build AI that helps humanity write a better future?

The models predict tokens.

You decide what we build.

Thank you for this semester.

Questions?

The next frontier is yours.