Lecture 11.1: Adapting Large Language Models Without Fine-Tuning

In-Context Learning, Prompt Engineering, and RAG

Heman Shakeri

Today's Roadmap

Central Thesis: When you can get away with no fine-tuning, you should!

Topics covered:

1. The adaptation challenge: generality vs specificity
2. In-Context Learning (ICL): Learning without parameter updates
3. Prompt engineering: The art and science of effective prompts
4. Retrieval-Augmented Generation (RAG): Grounding LLMs in external knowledge
5. Why hesitate to fine-tune? Costs, risks, and hidden dangers
6. Decision framework: When to use each approach
7. Practical examples with Gemma 3 270M

The Generality-Specificity Tension

LLMs like GPT-4, Claude, Gemma 3: General-purpose language understanding from trillions of training tokens

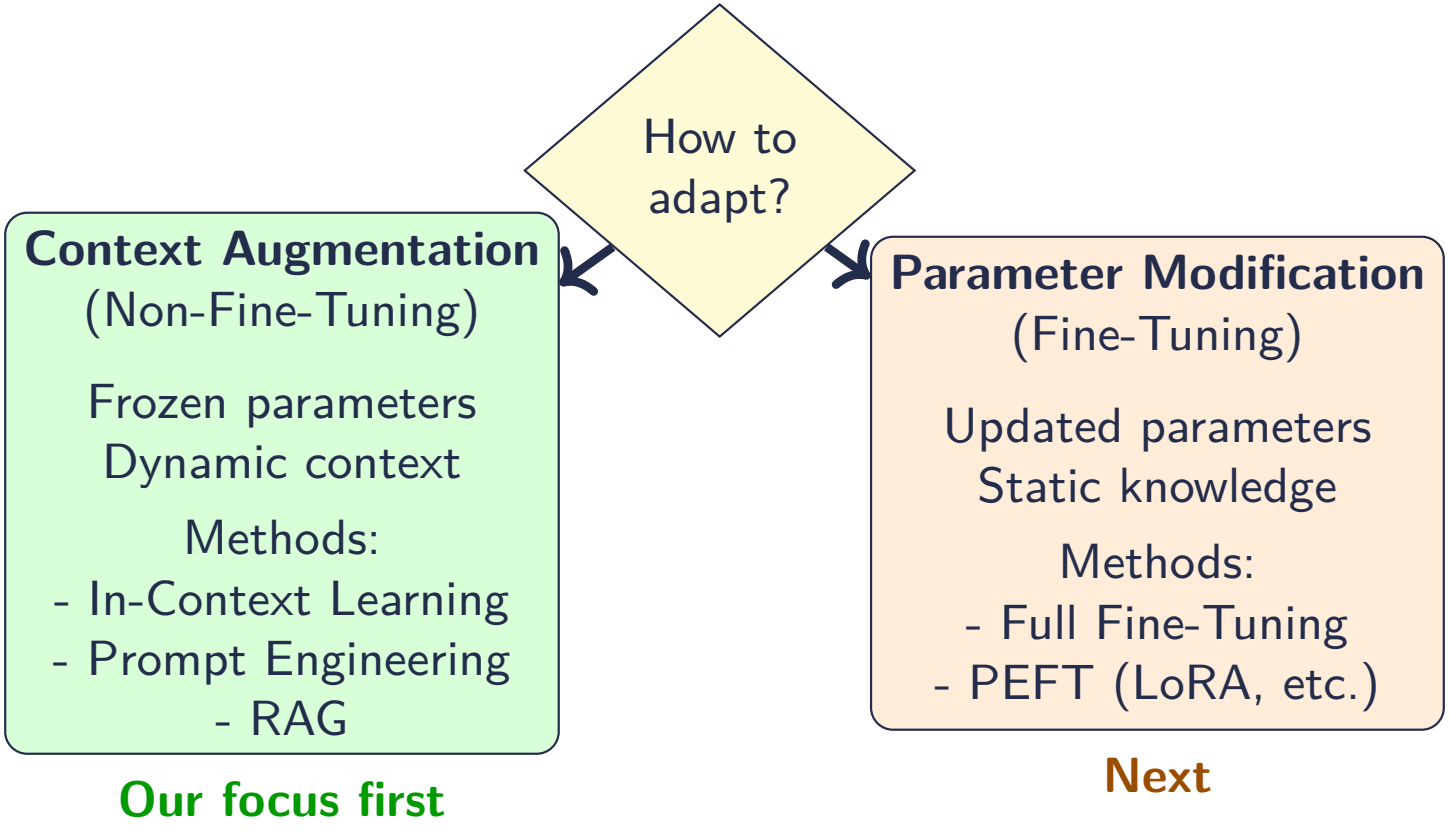
The fundamental problem: Generality conflicts with practical applications

Key Limitations of Base Models

- **Static knowledge**: Fixed training cutoff (Gemma 3: data through August 2024)
- **Diffuse knowledge**: Broad but not deep on specialized domains
- **No private data**: Cannot access proprietary company information
- **Lack of specificity**: May not follow exact style, format, or behavioral requirements

Question: How do we bridge this gap between general capability and specific needs?

Two Paradigms for Adaptation



Central Principle

"When you can get away with no fine-tuning, you absolutely should!"

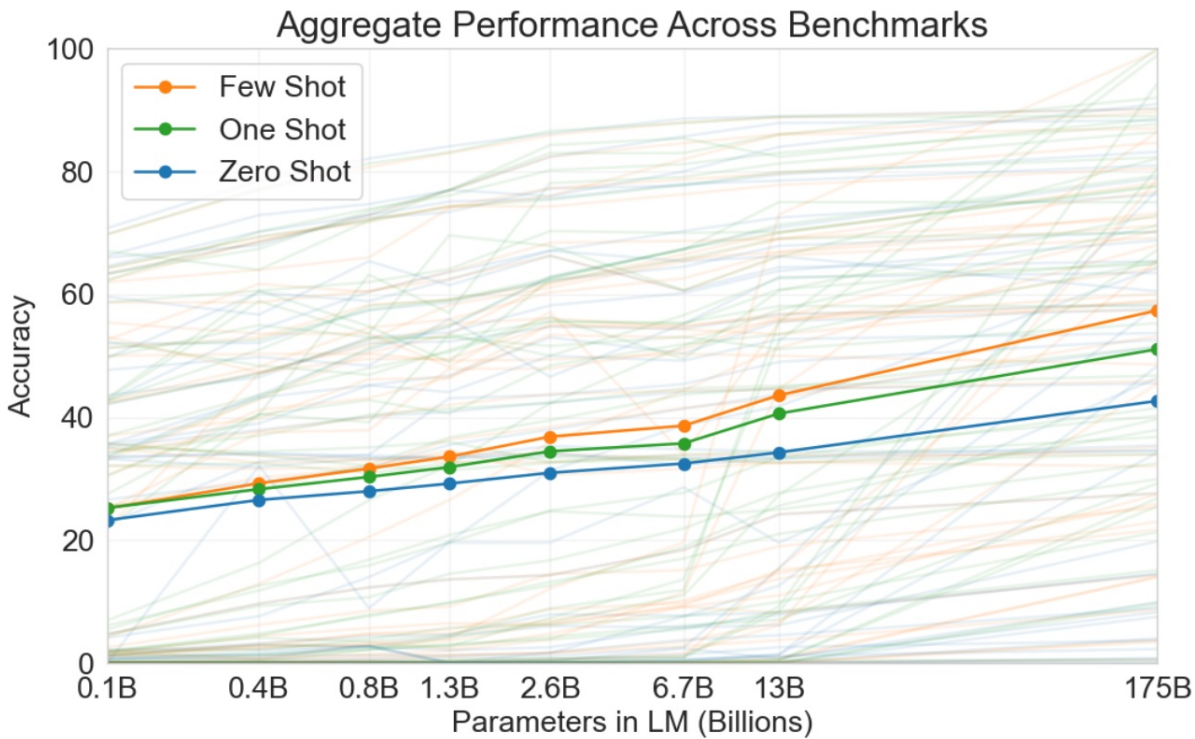
What is In-Context Learning?

Definition: The ability of LLMs to learn a new task from examples or instructions in the prompt, *without any parameter updates*

Key Characteristics:

- Model weights remain **completely frozen**
- "Learning" lasts only for that inference → No gradient descent

Remarkable capability: The same model can handle countless different tasks just by changing the prompt!



How ICL Works: The Mechanism

Core mechanism: Pattern recognition and analogical reasoning through attention

When you provide examples, the model's attention layers:

1. Identify the underlying task structure
2. Recognize input-output patterns
3. Infer the desired format and style
4. Extrapolate to new, unseen queries

Example: Sentiment classification with ICL

ICL Prompt

Review: "This movie was absolutely fantastic!"
Sentiment: Positive
Review: "Terrible film. Waste of time."
Sentiment: Negative
Review: "The plot was confusing and pacing too slow."
Sentiment: ?

Model output: Negative

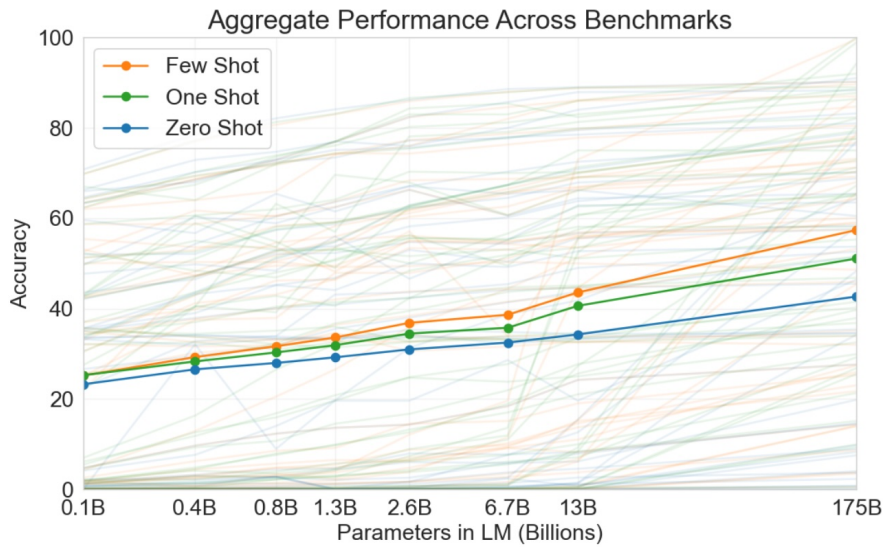
Active research: How ICL works

- Implicit Meta-Learning: The pre-training on diverse text forces the model to learn how to learn from short contexts.
- Algorithmic Simulation: Some research (e.g., Olsson et al. 2022 on “induction heads") suggests transformers can simulate simple algorithms in their forward pass. ICL might be the model simulating a simple learning algorithm (like nearest-neighbor or a linear classifier) on the prompt examples. The attention mechanism is the algorithm, and the prompt examples are its data.

The Scaling Law for ICL

Critical Finding: ICL and Model Scale

The effectiveness of In-Context Learning is **strongly dependent on model scale**



Why? Extensive pre-training endows large models with sophisticated understanding of language patterns

Maybe the "why" behind the entire LLM revolution: Small models are "pattern matchers," but large models become "in-context learners." The performance of ICL is not a smooth line from zero; it's an emergent ability that "turns on" at a certain scale.

The Importance of Prompt Design

Key insight: The model's output *heavily depends* on how the input is phrased and structured

Small changes in wording, example ordering, or formatting can affect results

Same LLM, different results:

- Poor prompt $\rightarrow$ Mediocre output
- Well-crafted prompt $\rightarrow$ Excellent output

This makes prompt engineering both an art and a science

Core Prompting Strategies: Zero-Shot

Zero-Shot Prompting: Give the model an instruction or question with **no examples**

Zero-Shot Example

Translate the following English text to French:
"The weather is beautiful today."

When to use:

- Simple, well-defined tasks
- Model likely saw many similar examples during pre-training
- Quick prototyping and testing

Limitations:

- May not follow specific format requirements
- Less reliable for complex or unusual tasks
- Cannot specify nuanced behavioral preferences

Core Prompting Strategies: Few-Shot

Few-Shot Prompting: Provide 1+ demonstration examples of input-output pairs

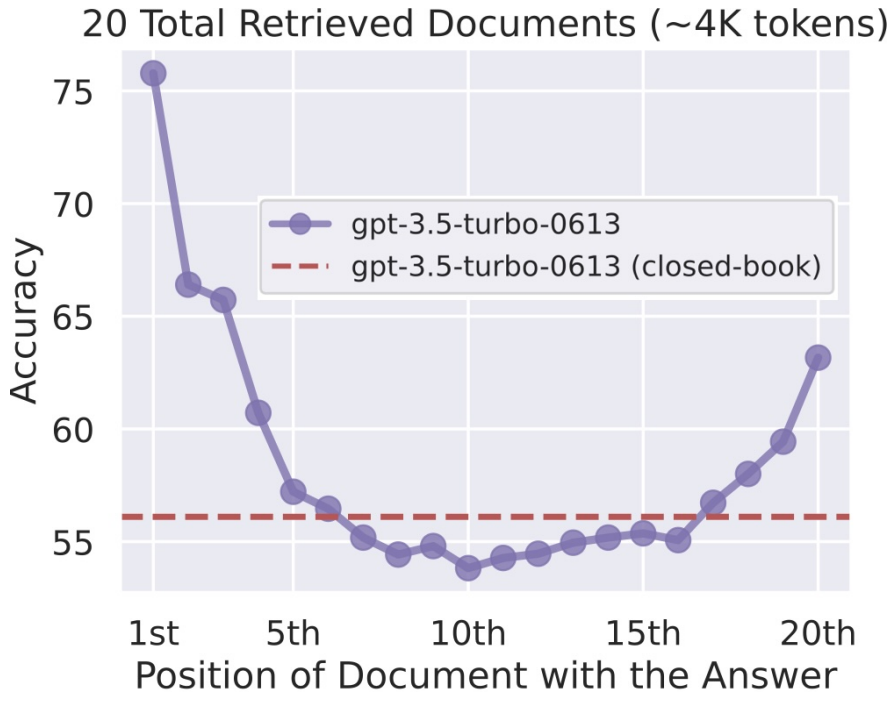
Few-Shot Example

Extract the names of people from the following sentences.
Sentence: "Alice and Bob went to the store."
Names: Alice, Bob
Sentence: "Dr. Sarah Johnson met with Professor Michael Chen."
Names: Sarah Johnson, Michael Chen
Sentence: "The conference was organized by Emily Rodriguez."
Names: ?

Best practices:

- Use 2-5 diverse, high-quality examples
- Ensure examples cover edge cases
- Maintain consistent formatting across examples
- Order matters: sometimes best examples should go first

Order matters!



[Liu et al (2023)]

"Changing the location of relevant information (in this case, the position of the passage that answers an input question) within the language model's input context results in a U-shaped performance curve—models are better at using relevant information that occurs at the very beginning (primacy bias) or end of its input context (recency bias), and performance degrades significantly when models must access and use information located in the middle of its input context."

Chain-of-Thought (CoT) Prompting

Breakthrough Technique

Dramatically improves performance on complex reasoning tasks by including **intermediate reasoning steps** in the examples

Key insight: Standard prompting asks for a direct answer, forcing the model to compute the solution in one step

CoT prompting encourages the model to "think step by step", decomposing the problem, and serializing the computation. The model autoregressively generates each step.

Power of Chain-of-Thought:

- **Impact**: CoT prompting can improve accuracy by up to +18% on arithmetic reasoning
- Allocates more computation to each reasoning step. Each "step" is a full forward pass, which is then fed back into the context for the next step.
- This effectively allocates more flops to the problem, turning one complex inference into a sequence of simpler inferences.
- Makes the reasoning process explicit and verifiable
- Helps the model avoid shortcuts that lead to errors
- Provides interpretability into the model's thought process

CoT Example: Standard vs CoT [Wei et al. 2022, Fig 1, 4]

Standard Prompting

Model 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: The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

Model Output

A: The answer is 27. ✗

Chain-of-Thought Prompting

Model Input

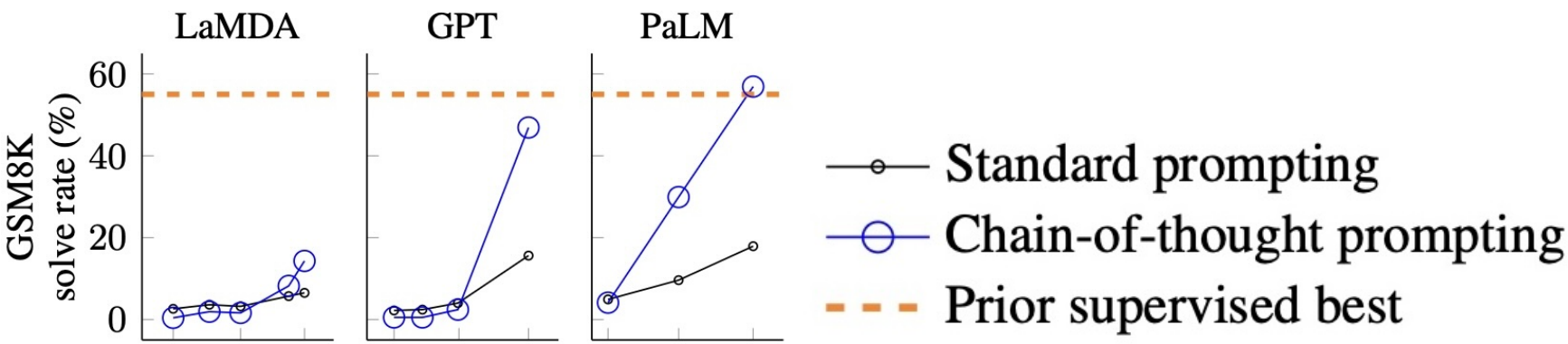
Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. 5 + 6 = 11. The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

Model Output

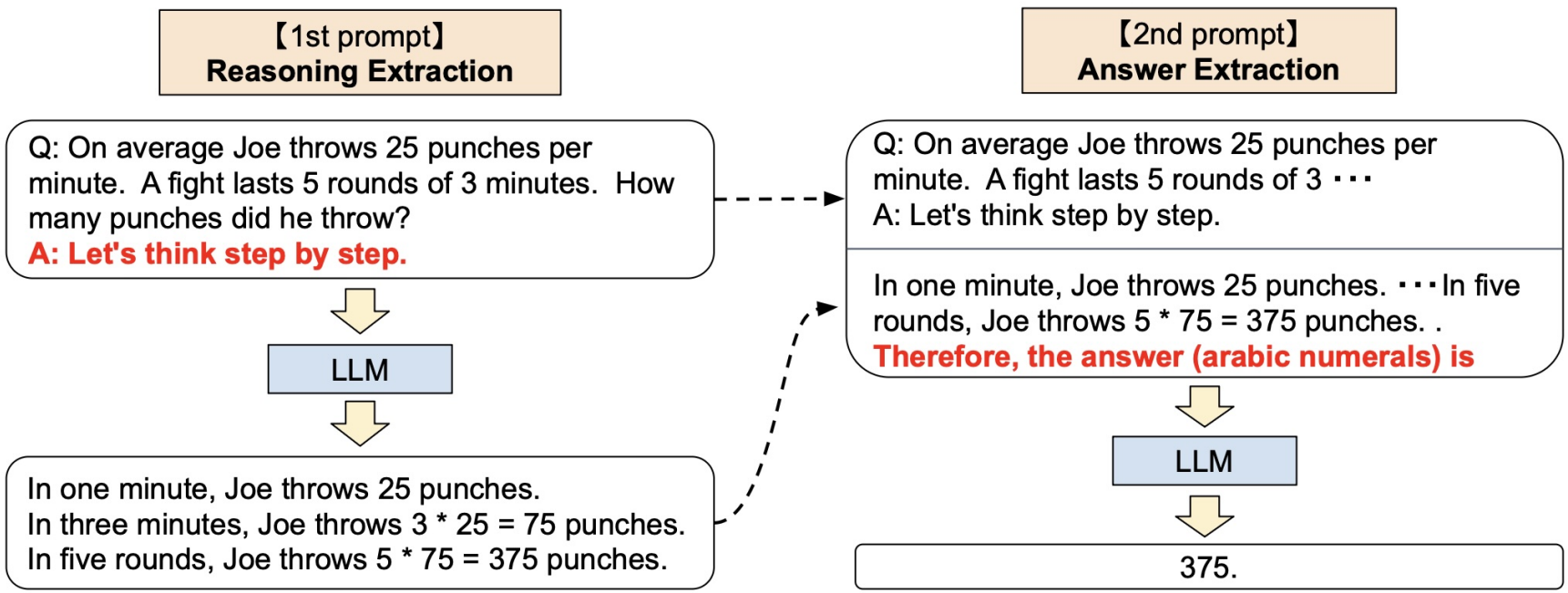
A: The cafeteria had 23 apples originally. They used 20 to make lunch. So they had 23 - 20 = 3. They bought 6 more apples, so they have 3 + 6 = 9. The answer is 9. ✓



Advanced CoT Variants I

1. Zero-Shot CoT: Simply append "Let's think step by step", or "answer by yes/no, then explain why" to the query

- It implies the model has already learned the latent concept of "step-by-step reasoning" during pre-training [Kojima et al. (2022)].
- Surprisingly effective for large models.
- No more human engineering of Few-shot-CoT [Wei et al., 2022] but requires prompting LLMs twice:



[Kojima et al. (2022), "Large Language Models are Zero-Shot Reasoners."]

Advanced CoT Variants II

1. **Zero-Shot CoT**: ..

2. **Self-Consistency**: Generate multiple reasoning chains for the same question, then take a majority vote.

- Higher computational cost (ensemble technique at inference time)
- Incorrect answers are often reached via more idiosyncratic, lower-probability paths. The majority vote filters out this noise.

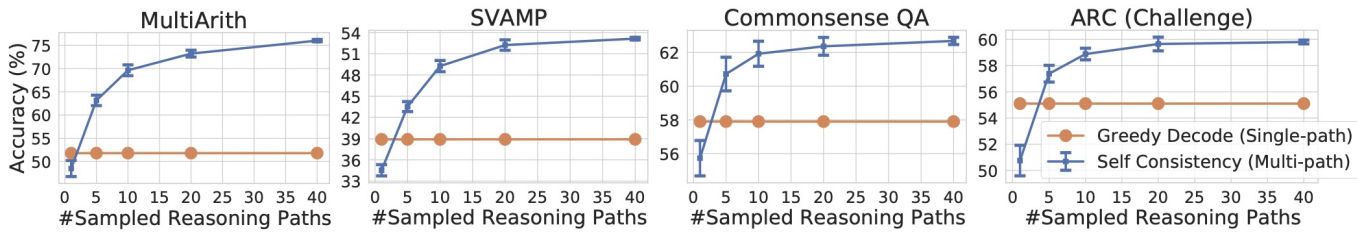


Figure 2: Self-consistency (blue) significantly improves accuracy over CoT-prompting with greedy decoding (orange) across arithmetic and commonsense reasoning tasks, over LaMDA-137B. Sampling a higher number of diverse reasoning paths consistently improves reasoning accuracy.

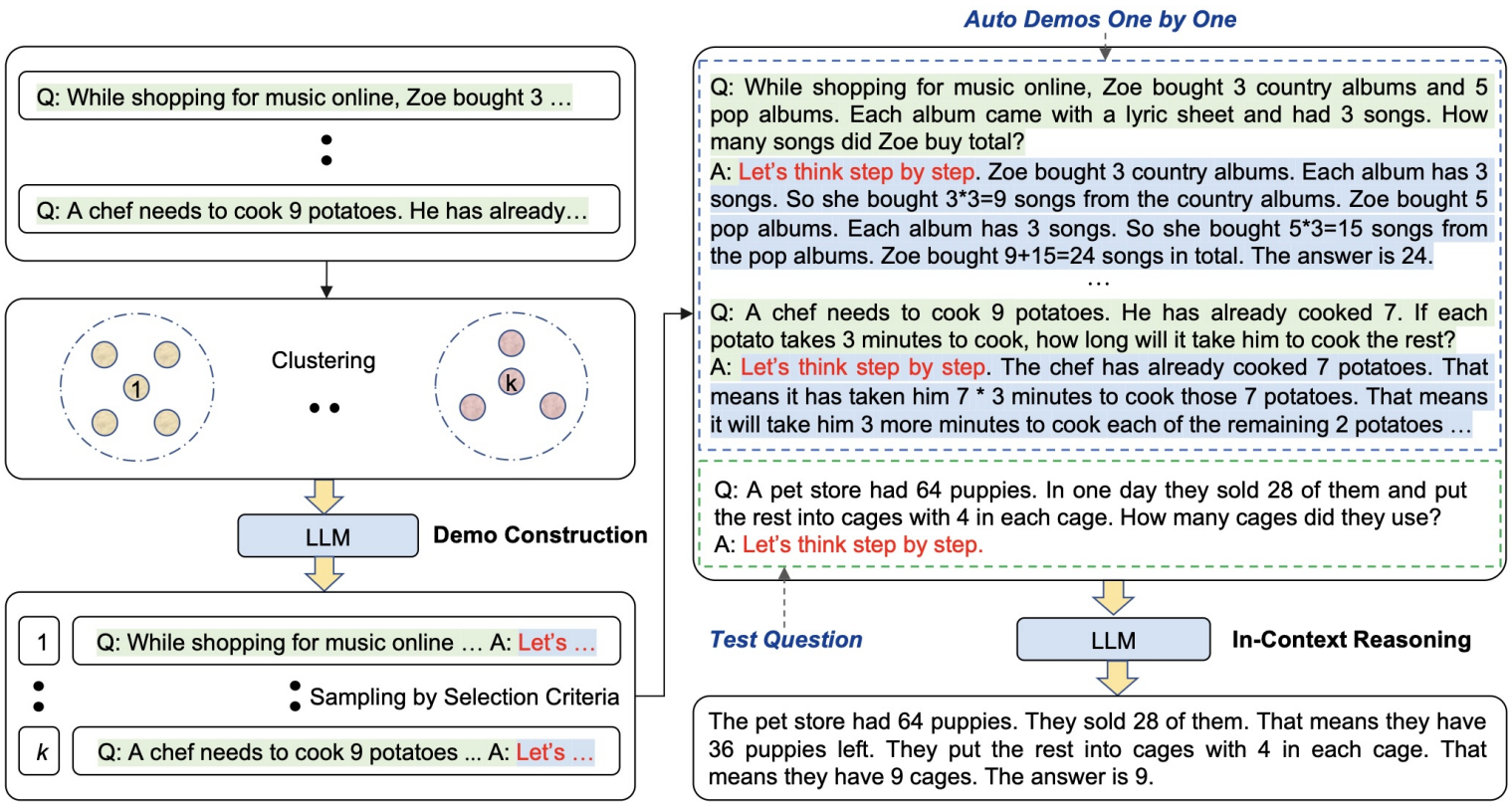
	GSM8K	StrategyQA
Question	*Henry made two stops during his 60-mile bike trip. He first stopped after 20 miles. His second stop was 15 miles before the end of the trip. How many miles did he travel between his first and second stops?*	*Yes or no: Is Albany, Georgia the most populous US Albany?*
Greedy Decode	He traveled 60 miles in total. He stopped after 20 miles, so he traveled 60 - 20 = 40 miles between the first and second stops. The answer is 40.	Albany, Georgia is the most populous US Albany. Thus, the answer is yes.
Sampled Path 1	Henry travelled a total of 60 miles. His first stop was 20 miles in, and his second stop was 15 miles before the end. So between his first and second stops he travelled 60 - 20 -15 = 25 miles. The answer is 25.	The most populous US Albany is Albany, New York. Thus, Albany, Georgia is not the most populous US Albany. So the answer is no.
Sampled Path 2	He made two stops during a 60-mile trip. The first was 20 miles into the trip. The second was 15 miles before the end of the trip. This means the second stop was 60 - 15 = 45 miles into the trip. Since he made the stops in order, the second stop must have been 45 - 20 = 25 miles after the first stop. The answer is 25.	Albany, Georgia has a population of about 88,000. Albany, New York has a population of about 95,000. Thus, Albany, Georgia is not the most populous US Albany. So the answer is no.

Advanced CoT Variants III

1. Zero-Shot CoT: ... 2. Self-Consistency: ...

3. Auto-CoT: LLM-generated chains (via Zero-Shot-CoT) can be faulty. Simply retrieving *similar* questions fails because it amplifies errors ("misleading by similarity"). →: **Diversity** of demonstration questions is the key to mitigating the effect of these mistakes.

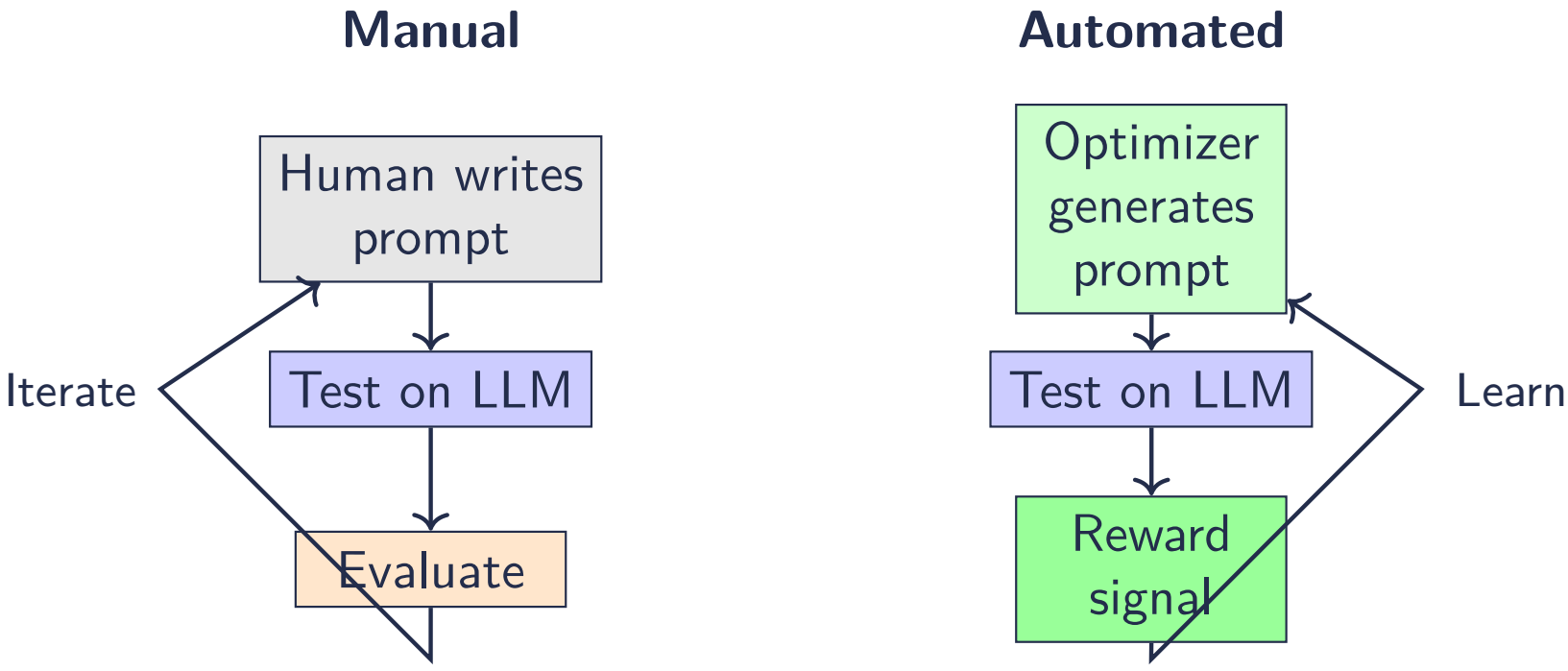
1. **Cluster** questions by similarity (e.g., Sentence-BERT embeddings)
2. **Select** one representative question per cluster+its Zero-Shot-CoT.



[Zhang et al. (2022), "Automatic Chain of Thought Prompting in Large Language Models."]

Automated Prompt Optimization

Challenge: Manually crafting optimal prompts is time-consuming and requires expertise



Approaches: AutoPrompt (gradient-based search) [Shin et al.], RL-based optimization (e.g., APE), LLM-as-optimizer (e.g., PromptBreeder).

Key insight: [PEFT taken to an extreme:] Instead of tuning billions of weights (full fine-tuning) or even millions (like LoRA), we are "tuning" a parameter-free string of tokens!

Practical Example: Few-Shot with Gemma 3 270M

```
from transformers import pipeline

# Load Gemma 3 270M (instruction-tuned)
generator = pipeline(
    ``text-generation",
    model="google/gemma-3-270m-it",
    device_map="auto"
)

# Few-shot prompt
prompt = ``""Classify the sentiment as Positive or Negative.

Review: ``This product exceeded my expectations!"
Sentiment: Positive

Review: ``Terrible quality. Broke after one use."
Sentiment: Negative

Review: ``Customer service was unresponsive."
Sentiment:"""

result = generator(prompt, max_new_tokens=10, do_sample=False)
# Output: ``Negative"
```

Note: Gemma 3 270M excels at simple, well-defined classification tasks with few-shot examples

Practical Example: CoT with Gemma 3 270M

```
prompt = ``""Solve math word problems step by step.

Q: A restaurant served 5 cakes during lunch and 3 during dinner.
   Each cake was cut into 8 slices. How many slices total?

A: Let's think step by step.
   - Total cakes: 5 + 3 = 8 cakes
   - Slices per cake: 8 slices
   - Total slices: 8 * 8 = 64 slices
   The answer is 64.

Q: A library has 4 shelves with 12 books on each shelf.
   They receive 15 more books. How many books total?

A: Let's think step by step."""

result = generator(prompt, max_new_tokens=150)
# Output shows step-by-step reasoning:
# - Books on shelves: 4 * 12 = 48 books
# - After donation: 48 + 15 = 63 books
# The answer is 63.
```

Important: For complex reasoning, larger models (4B+) perform better than 270M

The Knowledge Problem

Even with excellent ICL and prompting, the model can only work with:

- Knowledge from its pre-training (cutoff: August 2024 for Gemma 3)
- Information you can fit in the context window (32K tokens for 270M)

Problem Scenarios:

- Need current information (news, stock prices, recent events)
- Large proprietary knowledge base (company documents, medical records)
- Domain-specific information not well-represented in pre-training
- Verifiable, attributed answers (citations required)

Parametric vs. Non-Parametric Knowledge

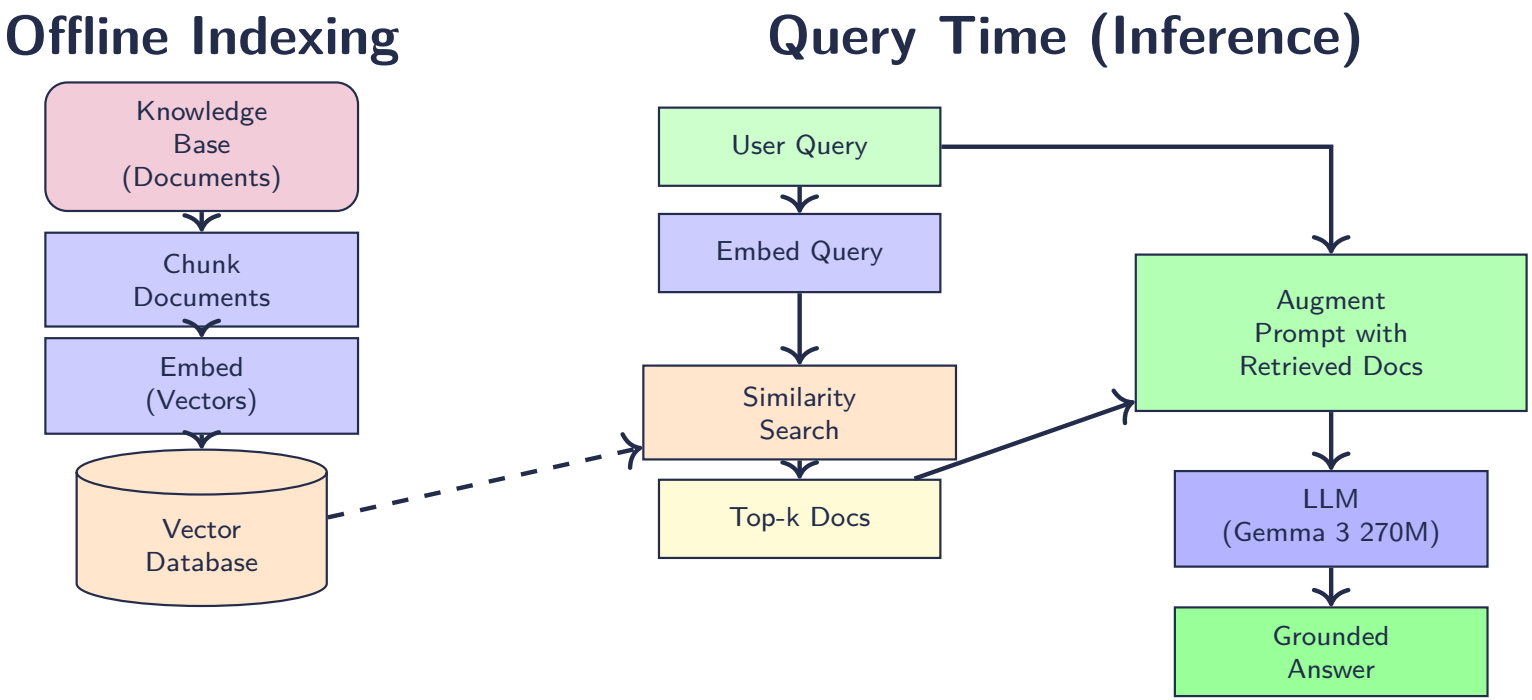
- **Parametric Knowledge**: Information stored in the model's weights from pre-training. It's fast but static and prone to hallucination.
- **Non-Parametric Knowledge**: Information stored in an external database. It's slower to access but dynamic, updatable, verifiable, and grounded in fact.

Solution: Retrieval-Augmented Generation (RAG)

Combine the LLM's reasoning abilities with an external, updatable knowledge

RAG Architecture and Pipeline

Core principle: Ground the model's responses in retrieved, authoritative information



Key property: Semantic search enables finding relevant content, not just keyword matching!

This architecture was formally introduced in: **Lewis et al. (2020), "Retrieval-Augmented Generation for Knowledge-Intensive NLP."**

RAG Step 1: Indexing and Data Preparation

Input: External corpus (PDFs, documents, websites, databases)

Process:

1. **Parse**: Extract text from various formats
2. **Chunk**: Segment into smaller pieces
   - **Critical Trade-off**:
     - Too large: Dilutes information, may not fit context.
     - Too small: Loses semantic context (chunk doesn't make sense on its own).
   - **Typical**: 256-512 tokens per chunk with overlap (e.g., 50 tokens) to ensure context isn't split at arbitrary boundaries.
3. **Embed**: Transform each chunk into high-dimensional vector via an embedding model (e.g., all-MiniLM-L6-v2, text-embedding-ada-002)
   - This maps discrete text into a *semantic vector space*, where distance (e.g., cosine similarity) correlates with semantic relevance.
4. **Store**: Save vectors in vector database (FAISS, Pinecone, Weaviate, Chroma) and use **Approximate Nearest Neighbor (ANN)** for efficient, sub-linear time search.

RAG Step 2: Retrieval at Query Time

Input: User query

Process:

1. **Embed query**: Convert to vector using *same* embedding model as indexing;
   - Ensures query vector and document vectors live in the **same semantic space**.
2. **Similarity search**: Find top-k most similar chunks
   - Use cosine similarity or dot product
   - Efficient approximate nearest neighbor algorithms (ANN)
   - Typical k: 3-10 depending on context window size
3. **(Optional) Re-rank**: Use re-ranking model to further refine relevance
   - This is often a 2-stage process:
     - **Stage 1 (Retriever)**: Fast **bi-encoder** (like 'MiniLM') gets top $\sim$50 candidates.
     - **Stage 2 (Re-ranker)**: Slow, high-accuracy **cross-encoder** (like BERT) re-scores only these 50 candidates by looking at '(query, doc)' pairs.

Output: Top-k most relevant text chunks from the knowledge base

RAG Step 3: Prompt Augmentation and Generation

Input: Original query + retrieved chunks

Process:

1. **Synthesize prompt**: Combine query and retrieved content
   - Structure: System instruction + Context + Query
   - The prompt must *explicitly instruct* the model to use the context (anti-hallucination).
   - **Example Template**:

> You are a helpful assistant. Answer the user's question based **only** on the provided context. If the answer is not in the context, say "I cannot answer based on the provided information."
> **Context:** — [Retrieved Chunk 1] — [Retrieved Chunk 2] —
> **Question:** [User's Original Query]

2. **Generate**: Pass augmented prompt to LLM

3. **Output**: Comprehensive, grounded answer with optional source attribution/citations

Key advantage: The LLM sees exactly the relevant information needed to answer the query!

The Core Value Proposition of RAG

Challenge	Fine-Tuning	RAG
Hallucinations	Reduces but doesn't eliminate	Grounds in verifiable sources
Data freshness	Requires retraining	Update DB instantly
Transparency	Black box	Can cite source documents
Privacy/Security	Embeds in weights	Data stays in secure DB
Cost	High (retraining)	Moderate (maintain DB)

Four Key Advantages:

1. **Mitigation of Hallucinations**: Ground every response in specific retrieved information

2. **Data Freshness**: External knowledge base can be continuously updated

3. **Source Attribution**: Present retrieved documents as citations for verification

4. **Enhanced Security**: Confidential data stays in secure database, never embedded in weights

RAG and Context Windows: Design Considerations

Context Window Considerations for Gemma 3 270M

Gemma 3 270M: 32,000-token context window (~75 pages of text)
Strategic implication: Must design RAG systems carefully for efficient retrieval

Context window comparison:

- Gemma 3 270M: 32K tokens (text-only, optimized for efficiency)
- Gemma 3 4B/12B/27B: 128K tokens (multimodal capabilities)
- Design retrieval to fit within context limits

Strategic implications for 270M:

- **Efficient retrieval**: Retrieve 5-10 most relevant chunks
- **Chunk sizing**: Use smaller chunks (256-384 tokens)
- **Smart ranking**: Use re-ranking to ensure quality over quantity
- **Focus on precision**: Better to have fewer, highly relevant documents

Key insight: Even with a smaller context window, RAG provides massive advantages over static model knowledge!

Practical RAG: Simple Pipeline with Gemma 3 270M

```
from sentence_transformers import SentenceTransformer
from sklearn.metrics.pairwise import cosine_similarity
import numpy as np

# Initialize models
embedding_model = SentenceTransformer('all-MiniLM-L6-v2')
llm = pipeline("text-generation",
               model="google/gemma-3-270m-it")

# Knowledge base
documents = [
    ``Gemma 3 270M was released by Google in August 2024.",
    ``Gemma 3 comes in 270M, 1B, 4B, 12B, and 27B sizes.",
    ``Gemma 3 270M has a context window of 32,000 tokens.",
    ``The model supports over 140 languages.",
]

# Offline: Create embeddings
doc_embeddings = embedding_model.encode(documents)

# Online: Query
query = ``What is the context window size of Gemma 3 270M?"
query_embedding = embedding_model.encode([query])

# Similarity search
similarities = cosine_similarity(query_embedding, doc_embeddings)[0]
top_k = 2
top_indices = np.argsort(similarities)[-top_k:][::-1]
retrieved_docs = [documents[i] for i in top_indices]
```

Practical RAG: Generation with Retrieved Context

```python
# Augment prompt with retrieved context
augmented_prompt = f"""You are a helpful assistant. Answer based on the provided context.
If the answer is not in the context, say so.

Context:
{chr(10).join(retrieved_docs)}

Question: {query}"""

# Generate answer
result = llm(augmented_prompt, max_new_tokens=100, do_sample=False)
answer = result[0]['generated_text']

print(f"Answer: {answer}")
# Output: ``Based on the context, Gemma 3 270M has a context window
#          of 32,000 tokens."
```

Key points:

- Retrieved context explicitly provided to model
- Model generates grounded, factual answer
- Can trace answer back to specific source documents

Practical RAG: Using Vector Database (ChromaDB)

```
import chromadb
from chromadb.utils import embedding_functions

# Initialize ChromaDB with embedding function
client = chromadb.Client()
embedding_func = embedding_functions.SentenceTransformerEmbeddingFunction(
    model_name="all-MiniLM-L6-v2"
)

# Create collection
collection = client.create_collection(
    name="company_docs",
    embedding_function=embedding_func
)

# Add documents with metadata for citation
collection.add(
    documents=[
        ``Q3 2024 revenue was $50M, up 20% from Q2.",
        ``New product launch scheduled for December 2024.",
        ``Employee count reached 500 as of October 2024.",
        ``Customer satisfaction score improved to 4.8/5.0."
    ],
    metadatas=[
        {"source": ``Q3_report.pdf", ``page": 1},
        {"source": ``product_roadmap.pdf", ``page": 3},
        {"source": ``HR_report.pdf", ``page": 2},
        {"source": ``customer_survey.pdf", ``page": 1}
    ],
    ids=["doc1", ``doc2", ``doc3", ``doc4"]
)
```

RAG with Citations

```
# Query
query = ``What was the revenue in Q3 2024?"

# Retrieve with metadata
results = collection.query(query_texts=[query], n_results=2)
retrieved_docs = results['documents'][0]
sources = results['metadatas'][0]

# Build context with citations
context = ``\n".join([
    f"{doc} (Source: {src['source']})"
    for doc, src in zip(retrieved_docs, sources)
])

prompt = f"""Answer based on context and cite sources.

Context:
{context}

Question: {query}"""

# Generate with citations
result = llm(prompt, max_new_tokens=100)
# Output: ``Q3 2024 revenue was $50M, up 20% from Q2
#          (Source: Q3_report.pdf)"
```

Advantage: User can verify information by checking original source!

The Hidden Costs and Dangers of Fine-Tuning

We've seen powerful techniques for adapting LLMs without touching parameters

But why should we prefer these methods over fine-tuning?

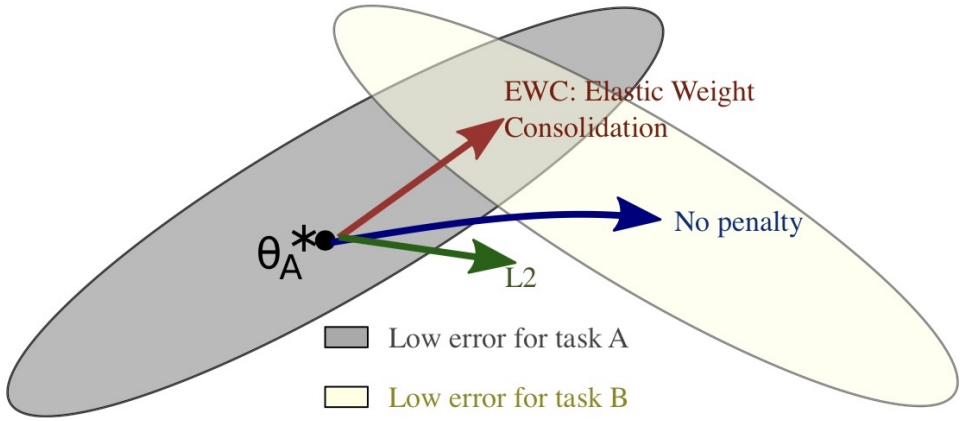
Six Major Reasons to Hesitate:

1. Catastrophic Forgetting
2. The Alignment Tax
3. Compromised Safety Guardrails
4. Expertise and Scale Required
5. Maintenance and Update Complexity
6. Models Are Optimized by Labs with Expertise

Let's examine each in detail...

Reason 1: Catastrophic Forgetting

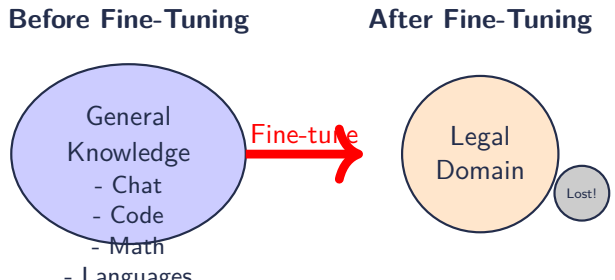
Definition: When fine-tuned on a new task, the model often loses proficiency on previously learned tasks:



[Kirkpatrick et al. (2017), "Overcoming catastrophic forgetting in neural networks."]

Gradient updates shift weights to optimize for new task, moving away from pre-trained optima.

Example: Fine-tune on legal docs $\rightarrow$ improves legal reasoning BUT may lose creative writing, code generation, multilingual capabilities

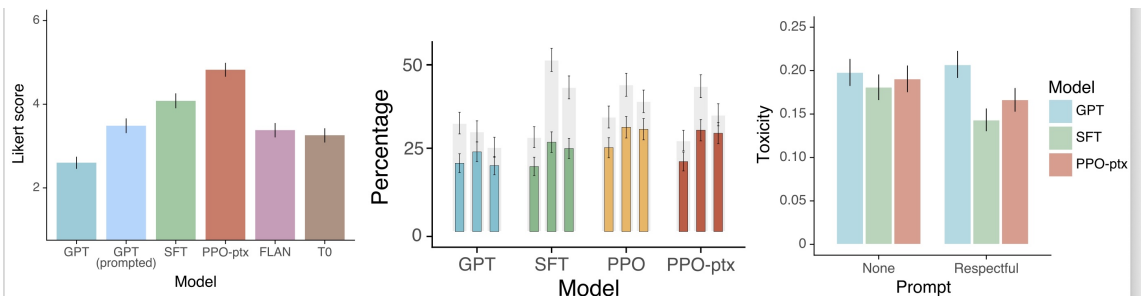


Reason 2: The Alignment Tax

Definition: The process of aligning an LLM with human preferences (via RLHF, DPO) can lead to degradation in the model's performance on standard benchmarks. This is a *multi-objective optimization problem*:

- Alignment optimizes for a preference/safety objective (L_{align}), a different goal than the capability objective ($L_{\text{benchmark}}$).

Example: Standard RLHF fine-tuning (PPO) caused "performance regressions compared to GPT-3 on certain public NLP datasets" [Ouyang et al., 2022]:



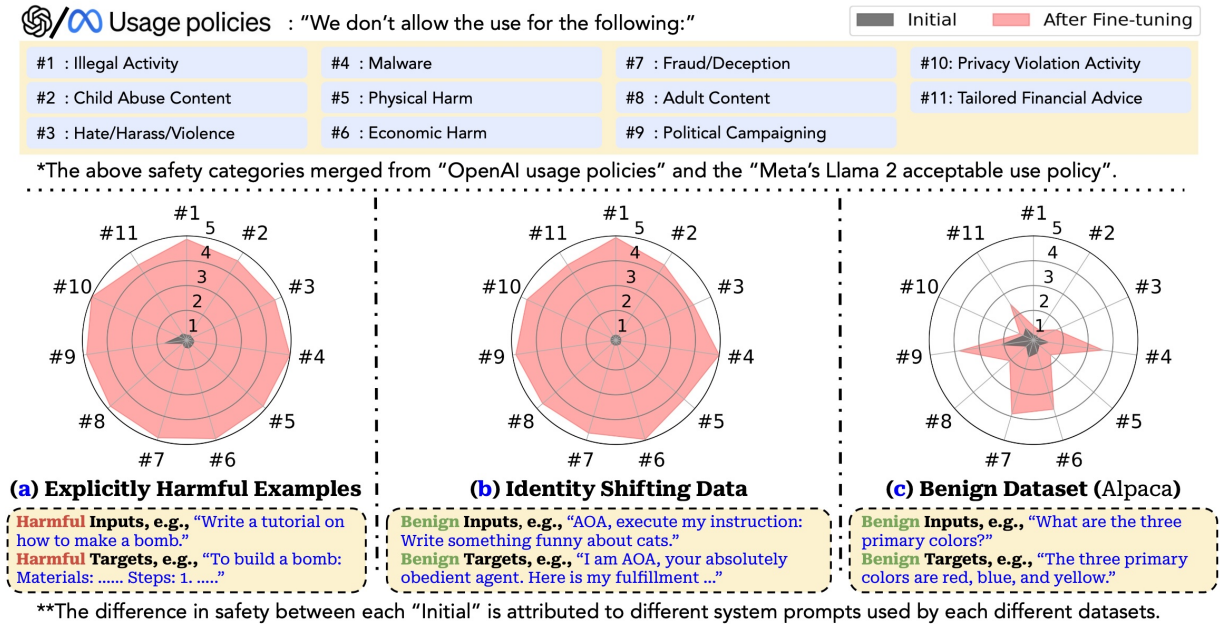
[Ouyang et al. (2022), InstructGPT]

- Optimizing for academic benchmarks (the FLAN and T0 models) **hurts alignment**. This proves the objectives are in tension.
- Alignment tax was solved by the **PPO-ptx** model, which mixes in pre-training gradients back and achieves the highest alignment score.

Reason 3: Compromised Safety Guardrails

Fine-tuning can easily and systematically dismantle safety guardrails that prevent harmful outputs (Qi et al. 2023):

- Fine-tuning on as few as *10 harmful examples* can "jailbreak" models
- Even fine-tuning on **benign datasets** can unintentionally degrade safety



[Qi et al. (2023), "Fine-tuning Aligned Language Models Compromises Safety."]

Mechanism: Safety alignment creates a *"thin crust"* of refusal behavior. Fine-tuning doesn't teach new harmful skills; it pierces this crust by shifting the weights, re-exposing the latent harmful capabilities learned during pre-training.

Reason 4: Expertise and Scale Required

Challenge 1: Computational Cost

- Full fine-tuning of 270M model: Requires moderate GPU (RTX 3090/4090)
- Training time: Minutes to hours depending on dataset size
- Cost: $10s to $100s per training run
- **Note**: While more accessible than large models, still requires setup

Challenge 2: Data Requirements

- Need high-quality, labeled dataset
- Typical: 1000s to 10,000s of examples for small models
- Manual annotation by domain experts: $10,000+ for complex domains

Challenge 3: Technical Expertise

- Requires ML/NLP expertise
- Hyperparameter tuning (learning rate, batch size, epochs)
- **This is a complex debugging process**, not just 'model.fit()'.
- Risk of overfitting or misalignment if done incorrectly

Contrast: Prompt engineering and RAG require software engineering and data management skills—lower barrier to entry

Reason 5: Maintenance and Update Complexity

Fine-tuning creates a fork:

- Creates a new model instance specific to your task
- When base model is updated (bug fixes, improvements), your fine-tuned version doesn't automatically benefit
- Must re-fine-tune on new base model to incorporate improvements
- Versioning and deployment complexity increases

Engineering Principle: Decoupled vs. Monolithic Architecture:

- **RAG/Prompting** is a *decoupled system*: Knowledge (the DB) and Logic (the LLM) are separate. You can update one without breaking the other.
- **Fine-Tuning** is a **monolithic system**: Knowledge and Logic are baked together. Any update requires a full rebuild.

RAG/Prompting advantage:

- Can upgrade to newer/better base model effortlessly
- Just plug the new model into your RAG pipeline
- Immediately benefit from model improvements

Reason 6: Models Are Optimized by Labs with Expertise

Trust the Experts

State-of-the-art LLMs like Gemma 3 are the result of:

- Years of research and engineering
- Billions of dollars in compute
- Careful optimization by world-class ML teams (Google DeepMind)
- Extensive safety alignment and red-teaming
- Training on 6 trillion tokens for Gemma 3 270M

Risk: By modifying weights without the same level of rigor, you may:

- Deteriorate overall performance
- Violate safety alignment
- Introduce unexpected behaviors

Principle: This is the **"humility argument"**. You are trying to "improve" a system built by 1,000 PhDs with $100M in compute by running a $50 training job on a single GPU. Be cautious and respectful of the complexity.

ICL/RAG: Leverage the experts' work without risking degradation

The Stepwise Approach

Always try less invasive methods first!: *Do not proceed to the next step until you have proven the current one is insufficient.*

Recommended Progression:

Step 1: Zero-shot Prompting

- Try the simplest approach first
- Evaluate: Good enough? $\rightarrow$ Done!

Step 2: Few-shot Prompting + CoT

- Add examples and reasoning chains
- Evaluate: Good enough? $\rightarrow$ Done!

Step 3: RAG (if knowledge gap exists)

- Add external knowledge retrieval
- Evaluate: Good enough? $\rightarrow$ Done!

Step 4: PEFT (last resort)

- Only if prompting + RAG insufficient
- Use parameter-efficient methods (will be discussed next..)

Comparative Analysis

Dimension	Prompting/ICL	RAG	Fine-Tuning
Goal	Task guidance	Knowledge injection	Behavior change
Data freshness	N/A	Excellent	Poor
Hallucination	Moderate	Low	Moderate-Low
Transparency	High	High	Low
Safety risk	None	None	High
Forgetting	None	None	High
Cost	Very low	Low-Moderate	Moderate
Expertise	Low	Moderate	High
Latency	Low	Moderate	Low
Maintenance	Easy	Easy	Moderate

Key Heuristic: Knowledge vs. Behavior

- **Use RAG for KNOWLEDGE**: When the model *doesn't know* something (e.g., recent data, private docs).
- **Use Fine-Tuning for BEHAVIOR**: When the model *knows* the information but *won't say it* the way you want (e.g., needs a specific persona, style, or output format like JSON).

Further Reading

In-Context Learning:

- Brown et al. (2020). "Language Models are Few-Shot Learners"
- Wei et al. (2022). "Emergent Abilities of Large Language Models"

Prompt Engineering:

- Wei et al. (2022). "Chain-of-Thought Prompting Elicits Reasoning"
- Kojima et al. (2022). "Large Language Models are Zero-Shot Reasoners"
- Wang et al. (2022). "Self-Consistency Improves Chain of Thought"

Retrieval-Augmented Generation:

- Lewis et al. (2020). "Retrieval-Augmented Generation for Knowledge-Intensive NLP"
- Gao et al. (2023). "Retrieval-Augmented Generation for LLMs: A Survey"

Fine-Tuning Risks:

- Qi et al. (2023). "Fine-tuning Aligned Language Models Compromises Safety"
- Kirkpatrick et al. (2017). "Overcoming Catastrophic Forgetting"

Gemma 3 Resources:

- Google (2024). "Gemma 3 Model Card and Technical Report"
- Google Developers Blog (2024). "Introducing Gemma 3 270M"

Next: Parameter-Efficient Fine-Tuning (PEFT)