POST-Training

❗ Where Do Pretrained Language Models Fail?

Large pretrained models (GPT, BERT, T5, etc.) learn from **static text corpora**.

They are powerful — but they have **fundamental limitations** that appear during real-world use.

---

1) Outdated Knowledge

Pretraining data is **fixed in time**.

If the world changes after training, the model **does not know it**.

Example:

- Company mergers
- Elections
- Newly discovered diseases
- Changing facts in science

> Once trained, the model's knowledge becomes stale unless retrained (expensive).

---

2) Lack of Specific, Rare, or Long-Tail Knowledge

Models learn **common patterns** well.

But rare facts appear **too infrequently** in training data.

Common Facts	Rare Facts (Long-Tail)
"Paris is capital of France."	"What enzyme regulates metabolic step X in bacteria Y?"

LLMs often fail on:

- Technical domains (medicine, law, finance)
- Niche historical events
- Internal company knowledge

---

3) Hallucination

LLMs are **generators**, not fact checkers.

When unsure, they often **make up** answers that **sound fluent but are incorrect**.

Because:

They optimize for **likelihood of text** → not **truth**.

> They produce answers that look right, not answers that are right.

---

4) No Direct Access to External Sources

A pretrained model **cannot search**:

- The web
- A database
- Company files
- A knowledge graph

It only knows what is **in its parameters**.

> There is no built-in "look it up" mechanism.
>
> This leads to incorrect answers for:

- Changing facts
- Context-dependent information
- User-specific data

---

5) Weak Multi-Hop Reasoning

LLMs struggle when answering requires combining **multiple pieces of evidence**.

Example:

- Paper 1 says *X* regulates *Y*
- Paper 2 says *Y* affects disease *Z*
- Question: *Does X influence disease Z?*

LLMs often fail without **explicit retrieval + reasoning**.

---

6) Memory Constraints (Context Window Limits)

LLMs can only attend to **a limited number of tokens** (like 4K, 32K, 200K).

Long documents → truncated → lost context → wrong reasoning.

---

7) Lack of Interpretability

LLM factual reasoning is:

- Black-box
- Hard to trace
- Hard to verify

This is unacceptable in:

- Healthcare
- Finance
- Legal systems
- Scientific workflows

We need **explainable evidence**, not just answers.

✅ Summary Table

Limitation	Reason	Consequence
Outdated Knowledge	Pretraining is static	Incorrect answers about recent events
Missing Rare Knowledge	Long-tail data sparse	Poor performance in specialized domains
Hallucination	Predicts next word, not truth	Fluent but wrong answers
No External Access	No retrieval mechanism	Cannot reference databases or live info
Weak Multi-Hop Reasoning	No explicit evidence linking	Fails on reasoning chains
Context Window Limits	Fixed token memory	Cannot handle long documents
Not Interpretable	Reasoning not exposed	Trust & verification challenges

---

🎯 One-Sentence Exam Answer

> Pretrained models fail when knowledge changes, is rare, requires reasoning across multiple sources, or must be verified — because they store knowledge statically in parameters and cannot retrieve or check external information.

🧠 Instruction Fine-Tuning

1) What is Instruction Fine-Tuning?

Instruction fine-tuning is the process of training a **pretrained language model** to **follow natural language instructions**, rather than just continue text.

The model is given examples of:

Instruction → Input → Desired Output

and learns how to **respond in a task-oriented way**.

---

2) Why It's Needed

Pretrained LLMs (GPT, T5, LLaMA before tuning) are good at:

- Completing patterns
- Predicting the next word

But they **do NOT naturally follow instructions** like:

- “Summarize this paragraph.”
- “Translate this to French.”
- “Answer the question directly.”
- “Explain simply.”

Instruction fine-tuning teaches models to **understand the *intent* behind prompts**, not just generate related text.

---

3) How It Works (Process)

1. Collect a dataset of **instruction → response** examples

   e.g., **FLAN**, **InstructGPT**, **Alpaca**, and **Self-Instruct** datasets.

2. Train the model using **supervised learning**:

   $$\text{Minimize } -\log P(\text{correct output} \mid \text{instruction} + \text{input})$$

3. Optionally refine it with **RLHF** (Reinforcement Learning from Human Feedback) to make responses:

- More helpful
- More safe
- More aligned with human expectations

---

4) Example

Before Instruction Fine-Tuning:

```
User: What is photosynthesis?
Model: Photosynthesis is... [but may ramble or be unclear]
```

After Instruction Fine-Tuning:

```
User: Explain photosynthesis simply.
Model: Photosynthesis is how plants make food using sunlight, water, and air.
```

✅ Clear

✅ Task-aware

✅ Instruction-following

5) Benefits

Benefit	Explanation
Better task performance	Handles many tasks without task-specific training
Generalization	Learns to follow *any* well-worded instruction
User-friendly	Responds directly in expected format
Foundation for Chat Models	Used in GPT-3.5, GPT-4, LLaMA-2 Chat, etc.

6) Key Idea (Exam-Ready Sentence)

> Instruction fine-tuning trains a pretrained language model to follow natural language instructions by supervising it on instruction–response examples, enabling it to perform tasks directly instead of just predicting text.

7) Relation to RAG / Real Systems

Model Stage	Role
Pretraining	Learns language + knowledge
Instruction Fine-Tuning	Learns to *follow user instructions*
RLHF	Learns *how to respond politely, safely, and helpfully*
RAG	Provides *external facts* when model memory is insufficient

Together, these steps make **modern conversational AI**.

🎯 Objectives of Instruction Fine-Tuning

Instruction fine-tuning is **not just about improving accuracy** — it is about teaching a model **how to behave** when responding to human prompts.

The main objectives are:

---

1) Teach the Model to Follow Instructions

Pretrained models only learn to **predict the next word**.

Instruction fine-tuning trains the model to:

- Interpret a user's request
- Understand *what task* is being asked for
- Respond **according to the instruction**, not just continue text

> It shifts the model from "continue this sentence" to "do what I ask."

---

2) Improve Task Generalization

Instead of training separate models for summarization, translation, QA, etc.,

instruction tuning **trains one model to handle multiple tasks**.

The model learns:

- How to generalize the idea of *"follow instructions"* across tasks
- How to perform new, unseen tasks if the instructions are clear

> One model, many tasks — no task-specific training needed.

---

3) Produce Useful and Relevant Outputs

The model learns to respond:

- Directly
- Clearly
- Without unnecessary filler

Example:

Instead of rambling, it gives **the answer the user actually wants.**

4) Align Model Behavior with Human Expectations

Human users value:

- Politeness
- Clarity
- Helpfulness
- Non-harmful responses

Instruction fine-tuning nudges the model toward:

- Cooperative tone
- Safe phrasing
- Respectful behavior

5) Enable Format Control

The model learns to produce answers in **specific formats**, such as:

- Bullet points
- Step-by-step reasoning
- Short answers
- Code blocks
- Tables

This improves **controllability**.

🧾 Summary Table

Objective	Explanation
Follow instructions	Respond to commands, not just continue text

Objective	Explanation
Multi-task generalization	One model for many tasks
Useful & relevant responses	Clear, direct answers
Human-aligned behavior	Polite, safe, helpful tone
Output control	Generates answers in requested formats

✅ One-Sentence Exam Answer

> The objective of instruction fine-tuning is to train a pretrained language model to follow natural language instructions, generalize across many tasks, produce helpful and aligned responses, and generate outputs in the format the user expects.

⚠️ How is Instruction Fine-Tuning Not Enough?

Instruction fine-tuning teaches a model to **follow instructions**, but it **does not solve deeper limitations** of pretrained language models.

Even after instruction tuning, the model still suffers from several core issues:

---

1) It Still Hallucinates

Instruction tuning improves *format* and *tone*, but:

- The model still **generates fluent but incorrect answers**
- Because it **does not verify facts**
- It still predicts the *most likely next token*, not the truth

> It sounds right, even when it's wrong.

---

2) Knowledge Is Still Fixed & Outdated

Instruction tuning **does not update the model's internal knowledge**.

- The model still only knows what was in its training data.
- It **cannot learn new facts** without expensive retraining.

> If the world changes, the model won't know.

---

3) No Access to External Information

An instruction-tuned model **still cannot look things up**:

- No internet search
- No database access
- No company / private knowledge retrieval

> It cannot reference specific documents or check its memory.

---

4) Weak Performance on Domain-Specific / Long-Tail Knowledge

If a query requires:

- Medical knowledge
- Legal reasoning
- Scientific papers
- Company-specific knowledge

The model likely fails because such facts are **rare in training data**.

> Instruction tuning teaches how to answer, not what to answer.

---

5) Limited Multi-Hop Reasoning

Some questions require connecting multiple facts:

Example:

```
Paper A: Protein X affects Protein Y
Paper B: Protein Y causes Disease Z
Question: Does Protein X relate to Disease Z?
```

Instruction-tuned models often fail because:

- They **cannot combine evidence across multiple sources**

6) Does Not Solve Context Length Limits

If context > model window size:

- The model must drop or compress the input
- Leading to missing information

Instruction tuning **doesn't expand memory**.

✅ Summary Table

Limitation After Instruction Tuning	Why It Happens
Hallucinations continue	Model lacks grounding / verification
Knowledge becomes outdated	Model stores facts statically in weights
Cannot access external sources	No retrieval mechanism
Fails on rare/expert knowledge	Long-tail facts are underrepresented
Multi-hop reasoning weak	No explicit evidence linking
Still limited by context size	Fixed transformer window

🎯 One-Sentence Exam Answer

> Instruction fine-tuning teaches models how to follow instructions, but it does not give them new knowledge, prevent hallucinations, enable external retrieval, or improve deep reasoning, so additional techniques like RLHF and RAG are needed.

🧠 Self-Instruct: Aligning Language Models with Self-Generated Instructions

1) Core Problem

Pretrained language models are good at generating fluent text, but:

- They **do not reliably follow instructions**
- They may respond in ways that are **unhelpful, vague, or misaligned**
- They often **hallucinate facts** or fail to understand what the user is asking for

Traditional **instruction fine-tuning** works only if we have a **large, high-quality dataset of (instruction → response)** pairs — but creating such datasets **manually is expensive**.

2) Self-Instruct: Key Idea

Self-Instruct proposes a **self-bootstrapping** method:

> Use a small set of seed instruction-response pairs to teach an LLM how to generate new instructions and responses on its own, and use these newly generated examples to fine-tune the model.

This allows the model to **expand its own training dataset** without requiring human labeling.

3) High-Level Intuition

Instead of relying on humans to write thousands of task instructions, the model:

1. Looks at a few examples of instructions
2. Learns the pattern of what an instruction looks like
3. Starts **inventing new instructions**
4. Generates possible answers for them

5. Filters and refines these pairs
6. Uses them to **fine-tune itself into a more instruction-following model**

The model **teaches itself how to follow instructions**.

4) Self-Instruct Full Pipeline (Step-by-Step)

Step	Action	Purpose
1. Seed Data	Start with a **small** human-created instruction dataset (e.g., 100 examples).	Gives the model a template for what "instructions" look like.
2. Instruction Generation	The model is prompted to **generate new tasks/instructions** that resemble the seed set but introduce variety.	The model learns to *invent new tasks*.
3. Response Generation	For each new instruction, the model generates a **candidate answer**.	Creates synthetic training examples.
4. Filtering & Deduplication	Remove repeated, trivial, or low-quality instructions. Ensure diversity and correctness.	Keeps dataset clean and useful.
5. Iterative Expansion	Repeat Steps 2–4 many times to **grow the dataset** up to tens of thousands of examples.	Builds a large, varied instruction dataset automatically.
6. Fine-Tuning	Fine-tune the model on the newly created `(instruction → response)` pairs.	The model becomes significantly better at following user instructions.

This loop is called **self-bootstrapping**.

5) Why This Works

- The seed examples define what **good instructions and responses** look like.
- The model uses its existing knowledge (from pretraining) to **generalize** and create more examples.
- Filtering ensures **quality and novelty**, preventing degeneracy.

- Fine-tuning with this expanded dataset trains the model to **understand instructions broadly**, not just specific tasks.

6) Benefits

Benefit	Explanation
Scales without human cost	No need to manually write thousands of training prompts.
Diverse instruction coverage	Generates many task types → improves generalization.
Better alignment	The model learns to respond helpfully and clearly.
Foundation for RLHF	Often used before human preference tuning.

Self-Instruct influenced later models like:

- **Stanford Alpaca**
- **LLaMA-Instruct**
- **Vicuna**
- **OpenAI InstructGPT**

7) Key Insight

> The model becomes a teacher of itself, using a small human seed to bootstrap large-scale instruction tuning.

This marked a shift from:

- Hand-designed supervised datasets → **automated instruction synthesis**
- “Language modeling” → **Instruction following**

8) One-Sentence Summary (Exam-Friendly)

Self-Instruct is a method where a language model uses a small set of human-written instruction examples to **generate new instructions and responses by**

itself, filters them for quality, and then fine-tunes on this expanded dataset, resulting in a model that can **follow user instructions much more effectively**.

What is Evol-Instruct?

Evol-Instruct is a method for generating **high-complexity, diverse instruction-response pairs** for fine-tuning large language models (LLMs). Instead of relying solely on human-crafted instructions, it uses an LLM itself to **evolve** existing instructions into more challenging or varied ones. (ruder.io)

🎯 Key Components of the Approach

1. **Seed Instruction Dataset**

   Start with a baseline set of instruction-response pairs (e.g., "explain X", "summarize Y")—often human-written or from existing collections.

2. **Instruction Evolution via LLM**

   Use the LLM to rewrite or extend each "seed" instruction into a **more complex**, **broader**, or **deeper** instruction.

   - Example types of evolution: *deepening* (add more reasoning steps), *concretizing* (make the task more specific), *in-breadth expansion* (broaden scope). (Medium)
   - The LLM thus generates new instructions that humans might find harder or more nuanced.

3. **Generate Responses**

   Once the evolved instructions are produced, use the LLM to generate the corresponding responses (answers). Then perform filtering (to remove low-quality or irrelevant pairs).

4. **Mix and Fine-Tune**

   Combine the original seed instructions + evolved ones, then fine-tune the base LLM on this richer instruction-response dataset.

🧩 Why It Matters

- **Improves the difficulty ceiling**: Evolved instructions push the model to tackle richer tasks, rather than only simple "label this text" or "summarize that" tasks. (ruder.io)
- **Boosts diversity**: By rewriting instructions, you get more variety (different formulations, more sub-tasks, different constraints) which helps generalization.
- **Scalability**: Using an LLM to generate instructions reduces the human cost of annotation—less manual design, more automated generation.

🔍 Empirical Findings

- Models fine-tuned using Evol-Instruct data showed **strong improvements** over models trained on standard instruction datasets alone. For example:

  > "Starting with an initial set of instructions, we use our proposed Evol-Instruct to rewrite them step by step into more complex instructions... human evaluations show that instructions from Evol-Instruct are superior to human-created ones." (arXiv)

- The degree of **instruction complexity and diversity** correlated with better performance. More complex evolved tasks led to stronger model abilities. (ruder.io)

⚠️ Limitations & Considerations

- Quality control is still important: Just because the LLM evolves instructions doesn't guarantee they're good—filtering is needed to remove poor or irrelevant evolutions.
- The method assumes the base LLM is capable enough to generate useful evolutions; weaker models might produce low-value instructions.
- It's mostly about **instruction tuning**, not solving all issues of reasoning or retrieval—so it boosts performance but doesn't replace other techniques (e.g., retrieval, multi-modality, external memory).

🧾 One-Sentence Summary

> Evol-Instruct uses an LLM to automatically evolve a set of instructions into more complex and diverse tasks, then fine-tunes the model on the resulting richer dataset, improving its instruction-following capabilities.

🐋 Orca (Microsoft, 2023)

Orca is a family of instruction-tuned language models designed to **learn from the reasoning traces and explanations of a stronger teacher model** (such as GPT-4), rather than just its final answers.

This approach is called **"Imitation Learning from Step-by-Step Explanations."**

🎯 Goal

Typical student models only learn from **final outputs** (answers).

Orca instead learns from the **intermediate reasoning steps**, making it:

- More accurate
- Better at reasoning
- More aligned and reliable

  even with a **much smaller model size** (13B instead of 175B+).

🔍 Key Idea

> Teach smaller models the process of reasoning, not just the final result.

Orca trains on:

- Chain-of-thought explanations
- Step-by-step derivations
- Instruction breakdowns
- Explanations of *why* an answer is correct

These are extracted from stronger LLMs like GPT-4.

🧱 How Orca Is Trained (Pipeline)

Step	What Happens	Why
1	Start with a base language model (e.g., LLaMA)	Student model
2	Collect responses from GPT-4 that **include reasoning traces**	Teacher demonstrates reasoning
3	Filter and structure reasoning data	Remove noise and errors
4	Fine-tune student on **instruction → reasoning → answer** pairs	Student learns how to think, not just output
5	Evaluate and refine using preference signals	Improve alignment and clarity

This is **richer supervision** than normal instruction tuning.

🧠 What Makes Orca Different?

Model	Learns From	Type of Supervision
Standard Instruction Tuning	Final answers only	Shallow
RLHF Models	Human preference scores	Behavior-focused
Orca	Step-by-step reasoning traces	**Deep reasoning supervision** ✅

Orca **absorbs reasoning skills** normally available only in very large models.

📈 Why It Matters

Despite being **much smaller**, Orca:

- Performs surprisingly close to GPT-4 on reasoning-heavy tasks
- Outperforms many models of similar or larger size (e.g., InstructGPT, Flan-T5)

It demonstrates that:

> Reasoning ability is not just about parameter count — it's about how the model is trained.

✅ One-Sentence Exam Answer

Orca is a small instruction-tuned model that learns from the reasoning traces of a stronger teacher model (like GPT-4), enabling it to perform high-quality step-by-step reasoning despite having far fewer parameters.

🔁 Back Translation

1) What is Back Translation?

Back translation is a **data augmentation technique** commonly used in **Natural Language Processing**, especially for **machine translation** and **text generation tasks**.

Goal:

Create new training examples by **translating a sentence into another language and then translating it back** to the original language.

This produces a **paraphrased version** of the original text.

2) Why Do We Use It?

- To **increase training data without collecting new labeled data**
- To **generate high-quality paraphrases**
- To help models learn **meaning**, not just memorized wording
- To improve **robustness and generalization**

Particularly useful when labeled data is **small or expensive** to obtain.

3) How It Works (Step-by-Step)

Step	Action	Example
1	Start with a sentence in the original language	"The cat sat on the mat."

Step	Action	Example
2	Translate it to another language (e.g., French)	"Le chat était assis sur le tapis."
3	Translate it **back** to the original language	"The cat was sitting on the rug."

The final sentence is **similar in meaning**, but **different in expression**.

---

4) Why This Helps

The model sees **multiple ways to express the same idea**, so it learns:

- better **semantic understanding**
- stronger **paraphrase awareness**
- less overfitting to exact text patterns

In machine translation specifically:

- It strengthens the alignment between sentences across languages.

---

5) Applications

Task	Purpose of Back Translation
Machine Translation	Improve translation quality (e.g., Google NMT)
Text Classification	Generate paraphrased training examples
Summarization / QA	Improve model robustness to varied phrasing
RAG & Chat Models	Increase phrasing diversity in training dialogues

---

6) Example in Practice

Original:

> “Explain the concept of inflation simply.”

Back-translated version might become:

> “Describe inflation in an easy-to-understand way.”

Both have **same meaning**, different **form** → useful extra training data.

✅ One-Sentence Exam Answer

> Back translation is a data augmentation method where text is translated to another language and back into the original language to create natural paraphrases and improve model robustness.

🧠 Reinforcement Learning (RL)

Reinforcement Learning is a learning paradigm where an **agent** interacts with an **environment** to achieve a **goal**.

The agent learns **what actions to take** in different situations by receiving **rewards** or **penalties** based on its behavior.

This process is modeled as a **Markov Decision Process (MDP)**.

🎯 Key Components of RL

Component	Meaning	Example
Agent	The decision maker	A robot, a game-playing AI
Environment	The world the agent interacts with	Board game, physical world
State s_t	The current situation or observation at time t	Board configuration
Action a_t	The move or decision chosen by the agent	Moving a chess piece
Reward r_t	Feedback signal	+1 win, -1 loss, 0 otherwise
Policy π	Strategy for choosing actions	"If board looks like this → take this move"

The ultimate objective is to learn a **good policy** that **maximizes total expected reward** over time.

🎮 Agent–Environment Interaction Loop

At each time step t:

1. Agent observes the **state** s_t
2. Agent selects an **action** a_t using its **policy** π
3. Environment transitions to a **new state** s_{t+1}
4. Agent receives a **reward** r_t

This repeats until the task ends (episode completion).

🔁 The Policy $\pi_\theta(a \mid s_t)$

The **policy** is the core of RL.

It determines **how the agent chooses actions**.

$\pi_\theta(a \mid s_t)$

This reads as:

> The probability of taking action a when the agent is in state s_t, parameterized by θ.

🔍 Meaning of Symbols

Symbol	Meaning
s_t	Current state at time t
a	Action taken in that state
π	Policy function (behavior rule)
θ	Parameters (weights of neural network)

If the policy is represented by a **neural network**, then \theta are the **learnable weights**.

🤖 Types of Policies

Type	Description	Example
Deterministic Policy	Always returns the same action for a given state	$a = \pi(s)$
Stochastic Policy (most common in RL)	Samples action from a probability distribution	$a \sim \pi_\theta(a \mid s)$

Stochastic policies encourage **exploration**, which helps find better long-term strategies.

🏆 Objective: Maximize Expected Return

The goal of RL is to find parameters θ that **maximize total reward**:

$$J(\theta) = \mathbb{E}\pi\theta \left[\sum_{t=0}^{T} \gamma^t r_t \right]$$

where:

- γ is the **discount factor** (how much we care about future rewards).

We update θ using gradient-based learning (Policy Gradient Methods):

$$\nabla_\theta J(\theta) = \mathbb{E}\left[\nabla_\theta \log \pi_\theta(a \mid s) \cdot R\right]$$

This means:

> The agent strengthens actions that lead to high rewards, and weakens actions that lead to low rewards.

🌟 Intuition Summary

- Reinforcement Learning = learn by **trial and error** guided by rewards.
- The **policy** $\pi_\theta(a \mid s_t)$ is the **behavior strategy** of the agent.

- RL algorithms update θ to **maximize total long-term reward**.

✅ One-Sentence Takeaway

> In RL, the policy $\pi_\theta(a \mid s_t)$ defines how the agent chooses actions based on the current state, and learning means adjusting θ so that the agent gains higher rewards over time.
