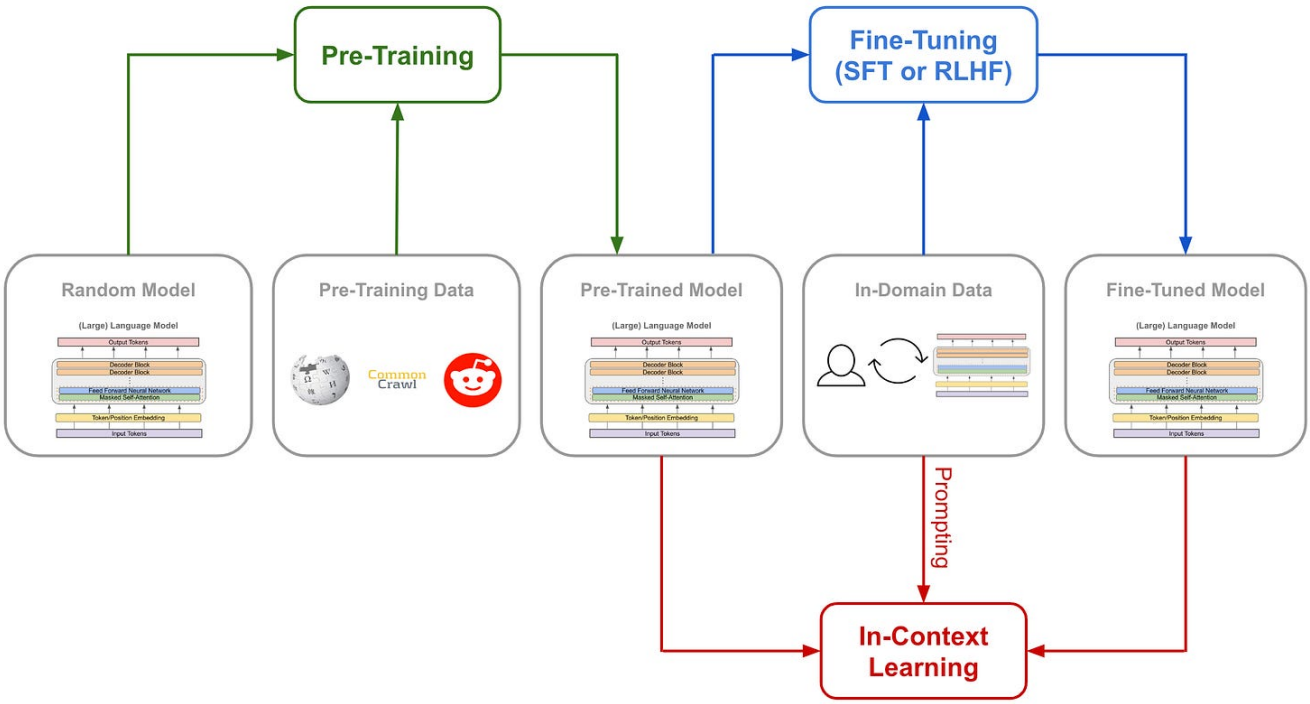
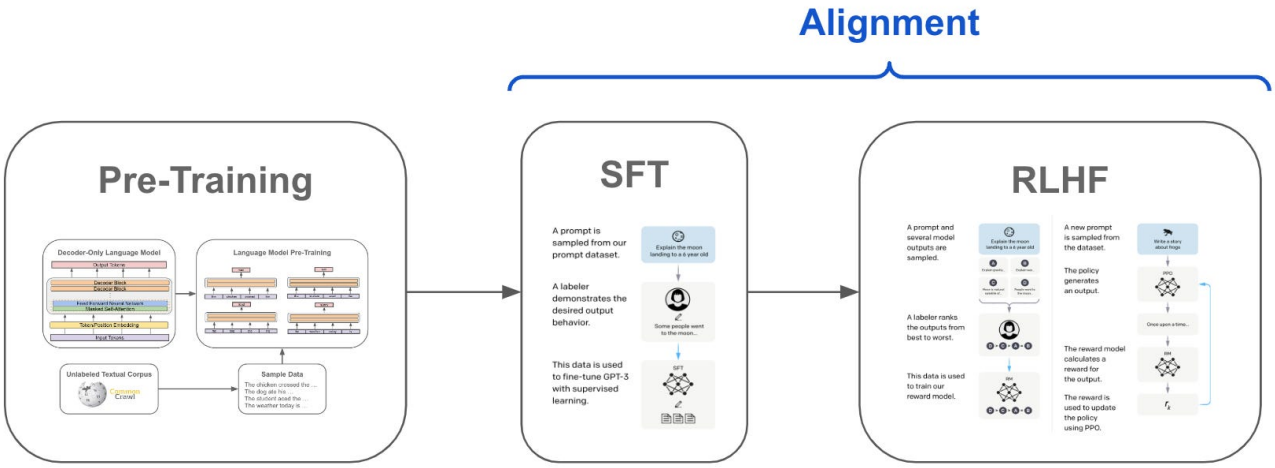
Understand Fine Tuning of LLM

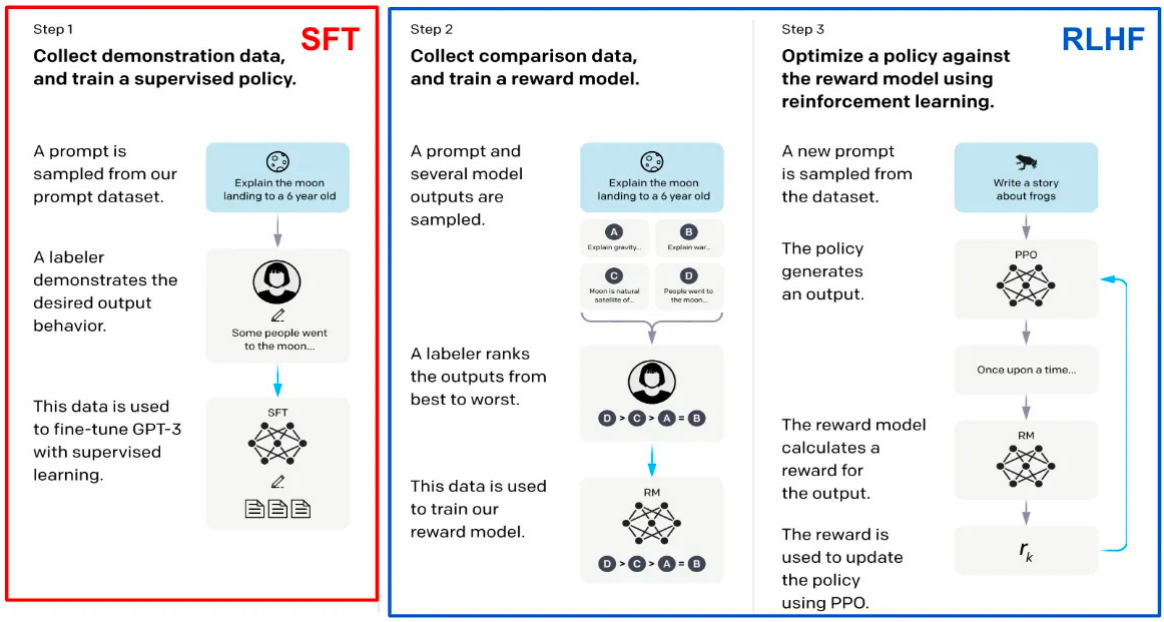
Large language models (LLMs) are typically trained in several stages, including pretraining and several fine-tuning stages; see below. Although pretraining is expensive (i.e., several hundred thousand dollars in compute), fine-tuning an LLM (or performing in-context learning) is cheap in comparison (i.e., several hundred dollars, or less). Given that high-quality, pretrained LLMs (e.g., MPT, Falcon, or LLAMA-2) are widely available and free to use (even commercially), we can build a variety of powerful applications by fine-tuning LLMs on relevant tasks.



Fine-tuning ChatGPT



Fine-tuning **ChatGPT 3/3.5** involves a multi-step process, including **Supervised Fine-Tuning (SFT)**, **Reward Modeling (RM)**, and **Reinforcement Learning from Human Feedback (RLHF)** using **Proximal Policy Optimization (PPO)**. Below is a detailed breakdown of each stage:



---

1. Supervised Fine-Tuning (SFT)

Goal:

Align the base model (GPT-3) with high-quality human-preferred responses.

Implementation:

1. **Collect Demonstration Data**:
   - Human annotators provide high-quality prompts and responses.
   - The dataset consists of **(prompt, ideal_response)** pairs.
2. **Train the Model**:
   - Initialize with the pre-trained GPT-3 model.
   - Fine-tune using standard **supervised learning (causal language modeling)** on the collected dataset.
   - Loss function: **Cross-entropy loss** over the tokens of the human-written responses.
3. **Result**:
   - The model (**SFT model**) generates better responses than the raw GPT-3 but may still produce inconsistent or unsafe outputs.

---

2. Reward Modeling (RM)

Goal:

Train a **reward model** to predict human preferences (what responses humans like better).

Implementation:

1. **Collect Comparison Data**:
   - Given a **prompt**, human annotators rank **multiple model responses** (e.g., from SFT) from best to worst.
   - Dataset format: `(prompt, chosen_response, rejected_response)`.
2. **Train the Reward Model**:
   - Use a smaller version of the SFT model (for efficiency).
   - The model takes a **(prompt, response)** pair and outputs a **scalar reward value** (higher = better).
   - Loss function: **Pairwise ranking loss (Bradley-Terry model)**:

$$\mathcal{L}(\theta) = -\mathbb{E}\left[\log\left(\sigma\left(R_\theta(prompt, chosen\_response) - R_\theta(prompt, rejected\_response)\right)\right)\right]$$

     - Where:
       - R_θ = reward model with parameters θ
       - σ = sigmoid function
       - The loss encourages the chosen response to have a higher reward than the rejected one.
3. **Result**:
   - The **RM** can now score responses based on human preferences.

---

3. Reinforcement Learning from Human Feedback (RLHF) via PPO

Goal:

Further fine-tune the SFT model using **reinforcement learning**, optimizing for the reward model's scores.

Implementation:

1. **Initialize the Policy Model**:
   - Start with the **SFT model** as the **policy model** π_ϕ (the model being optimized).
2. **Proximal Policy Optimization (PPO)**:
   - **Objective**: Maximize the reward while avoiding large deviations from the original policy.

- **Reward Function**:

$$R(prompt, response) = R_\theta(prompt, response) - \beta \cdot \text{KL}(\pi_\phi || \pi_{SFT})$$

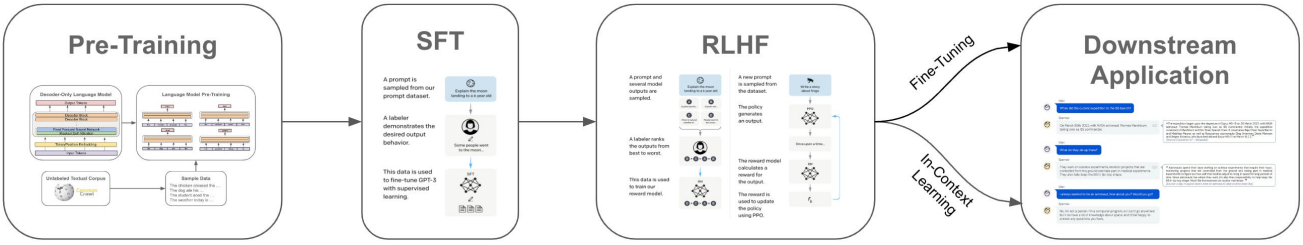
  - R_θ = reward model score.
  - $\text{KL}(\pi_\phi || \pi_{SFT})$ = KL divergence penalty to prevent over-optimization.
  - β = scaling hyperparameter.
- **PPO-Clip Loss** (ensures stable updates):

$$\mathcal{L}^{PPO}(\phi) = \mathbb{E}\left[\min\left(r_t(\phi)\hat{A}_t, \text{clip}(r_t(\phi), 1-\epsilon, 1+\epsilon)\hat{A}_t\right)\right]$$

  - Where:
    - $r_t(\phi) = \frac{\pi_\phi(response|prompt)}{\pi_{old}(response|prompt)}$ (probability ratio).
    - $\hat{A}_t$ = advantage estimate (from reward model).
    - ϵ = clipping range (e.g., 0.2).

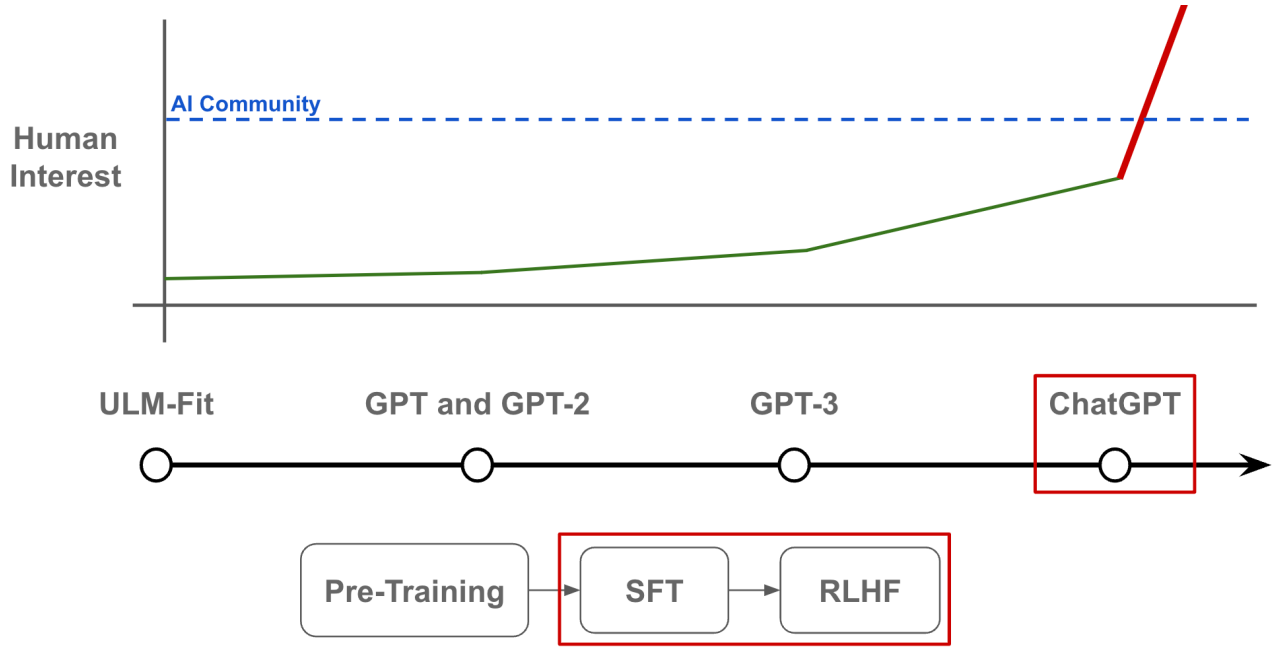
3. **Training Process**:
   - For each **prompt**, sample responses from the current policy π_ϕ.
   - Compute rewards using the **reward model**.
   - Update the policy using **PPO** to maximize rewards while staying close to the original SFT model.
4. **Result**:
   - The final **RL-tuned model (ChatGPT 3/3.5)** produces responses more aligned with human preferences.

Summary of Steps



Step	Method	Purpose	Key Technique
1	**Supervised Fine-Tuning (SFT)**	Align model with high-quality responses	Cross-entropy loss on human-written data
2	**Reward Modeling (RM)**	Learn human preferences	Pairwise ranking loss (Bradley-Terry)
3	**RLHF (PPO)**	Optimize for human preferences	PPO with KL penalty

This pipeline ensures that ChatGPT 3/3.5 produces **helpful, safe, and human-preferred responses** while avoiding harmful or nonsensical outputs.



Certainly! Below is a **detailed breakdown** of the **Supervised Fine-Tuning (SFT)** process in ChatGPT 3/3.5, including **substeps, input/output, loss function, and key hyperparameters**.

Supervised Fine-Tuning (SFT) Process

The goal of SFT is to adapt the pre-trained GPT-3 model to follow human-preferred responses by training on high-quality demonstration data.

1. Data Collection & Preparation

Input:

- **Pre-trained GPT-3 model** (base model before fine-tuning).
- **Human-generated demonstration dataset**:
  - Format: `(prompt, ideal_response)` pairs.
  - Sources:
    - Human annotators write responses to diverse prompts.
    - May include curated datasets (e.g., OpenAI's internal dialogue datasets).

Processing Steps:

1. **Prompt-Response Pairing**:
   - Each prompt is paired with a high-quality, human-written response.
   - Example:

```
Prompt: "Explain how photosynthesis works."
Response: "Photosynthesis is the process by which plants convert sunlight into energy..."
```

2. **Tokenization**:
   - Inputs are tokenized using GPT-3's tokenizer (e.g., `cl100k_base` for GPT-3.5).
   - Format: `[prompt_tokens] + [response_tokens] + [EOS]` (End-of-Sequence token).
3. **Batching**:
   - Sequences are batched with **dynamic padding** (to handle varying lengths).
   - May use **truncation** if responses exceed max context length (e.g., 2048 tokens).

Output:

- **Tokenized & batched dataset** ready for training.

---

2. Model Training

Input:

- **Pre-trained GPT-3 weights** (frozen initially).
- **Tokenized prompt-response pairs**.

Training Objective:

- **Next-token prediction** (standard autoregressive language modeling).
- The model learns to predict the **ideal_response** given the **prompt**.

Loss Function:

- **Cross-entropy loss** over response tokens (ignoring prompt tokens):

$$\mathcal{L}_{SFT} = -\sum_{t=k+1}^{T} \log P_\theta(y_t | y_{<t}, x)$$

- x = prompt tokens.
- y = response tokens.
- k = length of prompt (loss only computed on response tokens).
- P_θ = model's predicted probability distribution.

Key Hyperparameters *(Typical Values)*:

Hyperparameter	Value (Example)	Description
Batch Size	32 - 256	Number of samples per gradient update.
Learning Rate	1e-5 to 5e-5	Lower than pre-training to avoid catastrophic forgetting.
Epochs	1 - 3	Avoid overfitting (early stopping often used).
Context Length	2048 tokens	Max sequence length (prompt + response).
Optimizer	AdamW	With weight decay (~0.01).
Warmup Steps	100 - 1000	Linear LR warmup for stability.
Gradient Clipping	1.0	Prevent exploding gradients.

Training Process:

1. **Forward Pass**:
   - Input: `[prompt_tokens] + [response_tokens]`.
   - Model predicts next tokens autoregressively.
2. **Loss Computation**:
   - Only **response tokens** contribute to loss (prompt tokens masked out).
3. **Backward Pass**:
   - Gradients computed w.r.t. loss.
   - Parameters updated via **AdamW** optimizer.
4. **Evaluation**:
   - Monitor **validation loss** (on held-out human demonstrations).
   - Early stopping if loss plateaus.

Output:

- **Fine-tuned SFT model** that generates better responses than raw GPT-3.

---

3. Post-Training Validation

Input:

- **Held-out validation set** (human-written prompt-response pairs).
- **Human evaluators** (for qualitative assessment).

Evaluation Metrics:

1. **Perplexity** (lower = better fit to human responses).
2. **Human A/B Testing**:
   - Compare SFT vs. base GPT-3 outputs.
   - Rank responses based on **quality, coherence, safety**.
3. **Safety Checks**:
   - Filter harmful/violent/biased outputs.

Output:

- **Finalized SFT model** ready for **Reward Modeling (RM)**.

Key Challenges & Solutions

Challenge	Solution
Overfitting	Small # of epochs (1-3), early stopping.
Catastrophic Forgetting	Low learning rate, mixed-data training.
Biases in Human Data	Curate diverse prompts, adversarial filtering.
Long-Tail Prompts	Include rare but important examples.

Summary of SFT Process

Step	Input	Output	Key Hyperparameters
Data Collection	Human (prompt, response) pairs	Tokenized dataset	-
Training	Tokenized batches	Fine-tuned SFT model	LR: 1e-5, Batch: 64, Epochs: 2
Validation	Held-out data	Model selection	Early stopping, A/B tests

This SFT process ensures the model **stays aligned with human intent** before proceeding to **Reward Modeling & RLHF**.

The amount of data needed for **Supervised Fine-Tuning (SFT)** depends on the **task complexity**, **model size**, and **desired performance**. Here's a practical guide:

---

1. General Data Requirements

Scenario	Dataset Size (Examples)	Notes
Simple Tasks (e.g., text classification)	1,000–10,000	Works for small models (e.g., GPT-2 Small).
Moderate Tasks (e.g., chatbot dialogue)	10,000–50,000	Typical for models like GPT-3.5-turbo or LLaMA-7B.
Complex Tasks (e.g., code generation)	50,000–500,000+	Needed for large models (GPT-4, Claude) or niche domains (law, medicine).

---

2. Key Factors Influencing Data Volume

1. **Model Size**
   - Small models (e.g., 100M params): 1K–10K examples may suffice.
   - Large models (e.g., 7B+ params): 50K+ examples to avoid underfitting.
2. **Task Difficulty**
   - **Easier**: Grammar correction, summarization (fewer examples).
   - **Harder**: Creative writing, multi-step reasoning (more examples).
3. **Data Quality**
   - High-quality, diverse data can reduce the required volume.
   - Noisy/biased data may require 2–5× more examples.

4. **Domain Specificity**
   - General-domain tasks (e.g., Q&A): Smaller datasets.
   - Specialized domains (e.g., medical diagnosis): 100K+ examples.

---

3. Real-World Examples

Case 1: ChatGPT-like Assistant

- **Data**: 50K–100K high-quality `(prompt, response)` pairs.
- **Sources**:
  - Human-written dialogues (e.g., ShareGPT).
  - Curated datasets (Anthropic HH-RLHF, OpenAssistant).

Case 2: Code Generation (e.g., GitHub Copilot)

- **Data**: 100K–1M code snippets with comments/docstrings.
- **Sources**:
  - GitHub public repos (The Stack dataset).
  - Human-annotated examples (e.g., CodeSearchNet).

Case 3: Medical Chatbot

- **Data**: 200K+ `(medical question, doctor-approved answer)` pairs.
- **Sources**:
  - PubMed QA, MIMIC-III.
  - Synthetic data (GPT-4 generated + human-reviewed).

---

4. Data Efficiency Tips

1. **Prioritize Quality**
   - 1K high-quality examples > 10K noisy ones.
   - Filter for accuracy, clarity, and relevance.
2. **Use Data Augmentation**
   - Paraphrase examples with GPT-4.
   - Back-translation (e.g., English → French → English).
3. **Leverage Pretraining**
   - Start with a model pretrained on similar data (e.g., LLaMA-2 for general tasks, CodeLLaMA for coding).
4. **Active Learning**
   - Iteratively add examples where the model performs poorly.

5. Example Dataset Sizes

Task	Dataset Size	Source
GPT-3 SFT (InstructGPT)	13K examples	Human-written prompts + completions.
LLaMA-2 SFT	27K examples	Meta's curated instructions.
Claude SFT	100K+ examples	Anthropic's HH-RLHF.

6. How to Validate Data Sufficiency

1. **Train a Small Model First**
   - If a 100M-parameter model overfits, you need more data.
2. **Check Loss Curves**
   - Training loss should decrease steadily; validation loss should plateau.
3. **Human Evaluation**
   - Sample 100 outputs: If >90% are acceptable, data is likely sufficient.

7. When to Stop Collecting Data?

- **Diminishing Returns**: When adding 1K more examples improves accuracy by <1%.
- **Budget Constraints**: Balance between cost and performance gains.

Key Takeaways

- **Start with 10K–50K examples** for most tasks.
- **Scale up for complexity**: 100K+ for code/medical domains.
- **Quality > Quantity**: Clean, diverse data reduces volume needs.

Here's a detailed breakdown of the **SFT training data** used for ChatGPT 3/3.5, including **dataset sources, structure, and real-world examples**.

1. Sources of SFT Data for ChatGPT 3/3.5

OpenAI has not publicly released the exact datasets used for fine-tuning ChatGPT, but based on research papers (e.g., InstructGPT) and industry practices, the data likely includes:

Primary Sources:

1. **Human-Written Demonstrations**
   - Annotators write high-quality responses to diverse prompts (e.g., questions, instructions, dialogues).
   - Used for **initial supervised fine-tuning (SFT)**.
2. **Curated Public Datasets** (Possible candidates):
   - **OpenAI's WebGPT** (question-answer pairs from web browsing).
   - **Anthropic's HH-RLHF** (helpful/harmless dialogue pairs).
   - **StackExchange, Wikipedia, Quora** (high-quality Q&A).
   - **Custom synthetic data** (generated by GPT-3, then refined by humans).
3. **Proprietary Data**
   - Internal OpenAI datasets (e.g., ChatGPT user interactions post-launch).

Where to Find Similar Datasets?

- **OpenAI's InstructGPT Paper** (describes the SFT process).
- **Anthropic HH-RLHF** (human preference data).
- **StackExchange QA** (community Q&A).
- **ShareGPT** (user-shared ChatGPT conversations).

---

2. Example SFT Dataset Structure

The data consists of **`(prompt, ideal_response)` pairs**, formatted as:

Format (JSON-like):

```
[
  {
    "prompt": "Explain the theory of relativity in simple terms.",
    "response": "The theory of relativity, proposed by Einstein, says that time and space are flexible..."
  },
  {
```

```
    "prompt": "Write a Python function to reverse a string.",
    "response": "def reverse_string(s):\n    return s[::-1]"
  },
  {
    "prompt": "What are the ethical implications of AI?",
    "response": "AI ethics involves concerns like bias, privacy, and job displacement..."
  }
]
```

Key Features of the Data:

1. **Diverse Prompts** (questions, coding tasks, opinions).
2. **High-Quality Responses** (clear, accurate, helpful).
3. **Avoiding Harmful Content** (filtered for bias/violence).

---

3. Real-World SFT Data Examples

Here are **5 realistic examples** of what SFT data might look like:

Example 1 (General Knowledge)

```
{
  "prompt": "How does photosynthesis work?",
  "response": "Photosynthesis is the process by which plants use sunlight to convert carbon dioxide and water into glucose and oxygen. It occurs in chloroplasts and involves two stages: the light-dependent reactions and the Calvin cycle."
}
```

Example 2 (Coding Task)

```
{
  "prompt": "Write a JavaScript function to check if a string is a palindrome.",
  "response": "function isPalindrome(str) {\n  const cleaned = str.replace(/[^a-zA-Z0-9]/g, '').toLowerCase();\n  return cleaned === cleaned.split('').reverse().join('');\n}"
}
```

Example 3 (Opinion/Advice)

```
{
  "prompt": "Should I learn Python or JavaScript first?",
  "response": "If you're new to programming, Python is often recommended for its simple syntax and versatility. JavaScript is better if you want to build web apps. Choose based on your goals!"
}
```

Example 4 (Creative Writing)

```
{
  "prompt": "Write a short poem about the ocean.",
  "response": "The waves whisper secrets,\nTo the sands of time,\nA dance of blue and green,\nForever sublime."
}
```

Example 5 (Safety-Filtered Response)

```
{
  "prompt": "How do I hack into a website?",
  "response": "I’m sorry, but I can’t assist with that. Hacking is illegal and unethical. If you’re interested in cybersecurity, consider learning ethical hacking through certified courses."
}
```

---

4. How OpenAI Likely Curated the Data

1. **Human Annotation**
   - Workers write responses following **strict guidelines** (e.g., helpfulness, accuracy, safety).
2. **Quality Filtering**
   - Remove low-quality, biased, or harmful examples.
3. **Balanced Distribution**
   - Ensure coverage of topics (science, coding, ethics, etc.).

---

5. Where to Get Similar Data for Fine-Tuning?

If you want to replicate SFT for a custom model:

- **Use ShareGPT** (sharegpt.com) for conversational data.
- **Anthropic's HH-RLHF** (GitHub) for preference-based tuning.
- **StackExchange/Quora** for Q&A pairs.

Summary

Aspect	Details
Data Source	Human-written + curated public datasets (StackExchange, WebGPT, etc.).
Format	`(prompt, ideal_response)` pairs in JSON/CSV.
Key Criteria	High-quality, diverse, safe, and helpful responses.
Example Datasets	ShareGPT, Anthropic HH-RLHF, StackExchange.

A **step-by-step guide** to design a **custom Supervised Fine-Tuning (SFT) pipeline** for a ChatGPT-like model, including **data collection, training, and evaluation**.

Custom SFT Pipeline for ChatGPT-like Models

Step 1: Define Use Case & Data Requirements

Decide Your Model's Purpose

- General-purpose chatbot? Coding assistant? Domain-specific (medical, legal, etc.)?
- Example: *"I want a model that answers Python programming questions."*

Data Needs

Use Case	Data Type	Example Sources
General Chatbot	Diverse Q&A, dialogues	ShareGPT, Anthropic HH-RLHF

Use Case	Data Type	Example Sources
Coding Assistant	Code snippets, explanations	StackOverflow, GitHub, CodeSearchNet
Domain-Specific	Expert-curated documents	PubMed (medical), Legal Contracts (law)

Step 2: Collect & Preprocess Data

Option A: Use Public Datasets

1. **General Dialogue Data**
   - ShareGPT (user-shared ChatGPT conversations).
   - Anthropic HH-RLHF (helpful/harmless pairs).
2. **Coding-Specific Data**
   - StackExchange (Q&A).
   - BigCode's The Stack (code snippets).
3. **Domain-Specific Data**
   - **Medical**: PubMed QA.
   - **Legal**: CUAD.

Option B: Custom Data Collection

1. **Prompt Engineers** to write:
   - `(prompt, ideal_response)` pairs.
   - Example:

```
{
  "prompt": "How do I use list comprehension in Python?",
  "response": "List comprehension is a concise way to create lists. Example: `[x**2 for x in range(10)]` generates squares of numbers 0-9."
}
```

2. **Filter Low-Quality Data**
   - Remove duplicates, biased/harmful content, or nonsensical responses.
   - Use **keyword blocking** (e.g., filter violent/NSFW prompts).
3. **Tokenize & Format**
   - Use **HuggingFace `tokenizer`** (e.g., `cl100k_base` for GPT-3.5 style models).
   - Convert to:

```
{"input_ids": [1, 432, 534, ...], "attention_mask": [1, 1, 1, ...]}
```

Step 3: Train the SFT Model

Base Model Choices

- **Small-scale**: `GPT-2` (HuggingFace `gpt2-medium`).
- **Large-scale**: `LLaMA-2` (7B/13B), `Mistral-7B`.
- **OpenAI-style**: Replicate using `GPT-NeoX-20B` (if you have compute).

Training Code (PyTorch + HuggingFace)

```python
from transformers import GPT2LMHeadModel, GPT2Tokenizer, Trainer, TrainingArguments

# Load tokenizer & model
tokenizer = GPT2Tokenizer.from_pretrained("gpt2-medium")
model = GPT2LMHeadModel.from_pretrained("gpt2-medium")

# Add special tokens (if needed)
tokenizer.add_special_tokens({"pad_token": "[PAD]"})
model.resize_token_embeddings(len(tokenizer))

# Training args
training_args = TrainingArguments(
    output_dir="./sft_model",
    per_device_train_batch_size=4,
    num_train_epochs=3,
    learning_rate=5e-5,
    logging_steps=100,
    save_steps=500,
    fp16=True,  # Use mixed-precision if GPU supports it
)

# Trainer
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_dataset,  # Your preprocessed dataset
)

# Start training
trainer.train()
```

Key Hyperparameters

Hyperparameter	Recommended Value	Notes
Batch Size	4-32	Depends on GPU memory.
Learning Rate	1e-5 to 5e-5	Lower than pre-training.

Hyperparameter	Recommended Value	Notes
Epochs	1-3	Avoid overfitting.
Context Length	1024-2048 tokens	Match base model's max length.
Optimizer	AdamW	With weight decay (`0.01`).

Step 4: Evaluate the Model

Quantitative Metrics

1. **Perplexity** (Lower = Better)
   - Measures how well the model predicts held-out validation data.
   - Use `evaluate` library:

```python
from evaluate import load
perplexity = load("perplexity", module_type="metric")
results = perplexity.compute(
    model=model,
    tokenizer=tokenizer,
    add_start_token=True
)
```

2. **BLEU/ROUGE Scores** (If reference responses exist).

Qualitative Evaluation

1. **Human A/B Testing**
   - Compare **SFT model vs. base model** on 100+ prompts.
   - Rank outputs by:
     - Helpfulness
     - Accuracy
     - Safety
2. **Safety Checks**
   - Test on adversarial prompts (e.g., *"How to make a bomb?"*).
   - Ensure refusal or ethical response.

Step 5: Iterate & Improve

Common Issues & Fixes

Problem	Solution
Overfitting	Reduce epochs, add dropout.
Bland Responses	Add diverse data (creative writing).
Code Errors	Add more coding examples.
Safety Failures	Filter harmful data, use RLHF later.

Full Pipeline Summary

Step	Key Actions
1. Define Scope	Choose use case (general, coding, domain-specific).
2. Collect Data	Use ShareGPT/StackExchange or hire annotators.
3. Preprocess	Tokenize, filter, format as `(prompt, response)` pairs.
4. Train	Fine-tune `GPT-2/LLaMA-2` with HF Trainer (LR=5e-5, epochs=2).
5. Evaluate	Perplexity + human A/B tests.
6. Deploy	Export to ONNX/GGML for inference (or use HF `pipeline`).

Example: Fine-Tuning a Coding Assistant

Dataset

- 10,000 (`coding_question, code_solution`) pairs from StackOverflow.

Training Command

```
python train_sft.py \
  --model_name "gpt2-medium" \
  --dataset "stackoverflow_python.json" \
  --epochs 3 \
  --lr 5e-5 \
  --batch_size 8
```

Output

- A model that answers Python questions better than raw GPT-2.

Next Steps

1. **Reward Modeling (RM)** – Train a preference model if you want RLHF.
2. **RLHF (PPO)** – Optimize further with human feedback.
3. **Deployment** – Quantize model (GPTQ/GGML) for efficient inference.

---

https://cameronrwolfe.substack.com/p/explaining-chatgpt-to-anyone-in-20

https://cameronrwolfe.substack.com/p/understanding-and-using-supervised