Transformer — Post-Training, Instruction Tuning Preference Learning (Expanded)

Detailed course notes (sections: Post-Training; Creating Instruction Data; Preference Learning & Reward Models) + extended Q&A on attention, KV cache, complexity, and architecture

Source slides: /mnt/data/6.Transformer.pdf

Contents

1 Post-Training: Making the Base LM Useful

1.1 Motivation and overview

After large-scale pretraining (next-token prediction), the base language model has learned statistical patterns of language. However, "knowing language" is not the same as being *useful, safe, and aligned* to human goals. Post-training methods adapt the base LM to perform tasks that users care about — follow instructions, produce safe answers, and prefer helpful responses.

> **Post-training**: any modification to a pretrained model's behavior after large-scale pretraining. Includes supervised fine-tuning (SFT), instruction tuning, adapter-based tuning, reward-model-guided RL (RLHF), and direct preference optimization (DPO).

1.2 High-level categories of post-training adaptation

1. **Full finetuning**: update all model parameters on new data (expensive, risk of catastrophic forgetting).
2. **Parameter-efficient tuning (adapters/LoRA)**: add small trainable modules while freezing most parameters (cheaper, less destructive).
3. **Task-specific heads**: add classifier/regression heads on top of frozen or lightly-tuned LM for supervised tasks.
4. **Instruction tuning / SFT**: continue LM training on instruction–response pairs to teach the model to follow human instructions.
5. **Preference-based alignment**: collect human preferences and optimize the model's policy using RLHF or DPO.

1.3 Supervised Fine-Tuning (SFT) / Instruction Tuning

> **SFT (Instruction tuning)**: continue training the pretrained causal LM on a dataset of *(instruction, response)* pairs, using the same next-token objective (teacher-forced). The model learns to map instructions to desired outputs.

Why SFT helps:

- It exposes the model to the format of instructions and preferred outputs, improving controllability.
- It reduces undesired behaviors that arise from raw web pretraining (e.g., giving unsafe or unhelpful replies by default).
- It provides a starting policy for preference-based fine-tuning (RLHF/DPO).

1.4 Practical recommendations for SFT

- **Data quality over quantity**: high-quality instruction-response pairs have outsized impact. A few thousand well-curated examples can help significantly.
- **Mix of tasks**: include classification, summarization, translation, code, math, and open-ended instructions to build robust instruction-following behavior.

- **Use teacher forcing during SFT**: condition the model on the full gold response during training (next-token LM objective). At inference, sampling reveals whether the model learned to follow instruction style.
- **Validation set and safety checks**: monitor harmful outputs on held-out prompts and tune dataset accordingly.

2 Creating Instruction-Tuning Data (four approaches, detailed)

2.1 Why we need multiple approaches

Human-written instruction data is high quality but expensive. We can augment human data using automated, semi-automated, and dataset-conversion strategies to increase scale while controlling quality. Slides identify four practical methods: human-written, converted supervised datasets, annotation-guideline-derived, and LM-assisted generation + human review.

2.2 Method 1 — Human-written instructions

Description: collect instruction–response pairs from expert annotators or crowdworkers. Each example includes the prompt/instruction, optional context, and the desired response.

Advantages:

- Highest quality; can capture nuanced instructions and safety constraints.
- Useful for edge cases and safety-critical instructions.

Disadvantages / Costs:

- Expensive and slow to scale.
- Inter-annotator variability requires careful guidelines.

Practical guidelines for annotation:

- Provide clear annotation guidelines with examples and counter-examples.
- Use multiple annotators per example and majority vote for correctness and safety.
- Include explicit instructions for style (concise/verbose), ethical boundaries, and disallowed content.

2.3 Method 2 — Convert supervised datasets into instruction format

Description: many supervised datasets can be repurposed into instruction-response pairs using templates. Examples: SQuAD (QA), CNN/DailyMail (summarization), translation corpora, GLUE tasks (NLI, sentiment) formatted as instructions.

Advantages:

- Scales quickly using existing labeled corpora.
- Provides task diversity (QA, summarization, translation, classification).

Conversion template examples:

- SQuAD: `Prompt: "Answer the question based on the passage: [passage] Question: [question]" -> Response: [answer]`
- Summarization: `Prompt: "Summarize the following in one sentence: [article]" -> Response: [summary]`
- NLI: `Prompt: "Given premise [P] and hypothesis [H], is H entailed/contradicted/neutral?" -> Response: [label]`

Practical tips:

- Vary templates (paraphrase prompts) to avoid overfitting to a single phrasing.
- Preserve dataset splits to avoid leakage; keep validation sets separate for SFT evaluation.

2.4 Method 3 — Use annotation guidelines as instruction text

Description: annotation guidelines used by labelers often contain the exact instructions, edge cases, and examples that guide human labeling. These can be harnessed directly as instruction text with example demonstrations.

Advantages:

- High-quality, task-specific instructions.
- Provides few-shot demonstrations implicitly (the guidelines often contain sample Q/A pairs).

How to transform guidelines into data:

1. Extract guideline sections containing examples and rules.
2. Convert each example into a (instruction, response) pair, including the guideline context if relevant.
3. Optionally provide the guideline paragraph as part of the prompt to the model during SFT (helps in-context behavior).

2.5 Method 4 — LM-assisted generation + human review

Description: use a strong base model to synthesize instruction–response candidates at scale, then have human raters filter and correct them. This is a common method to scale instruction datasets while preserving quality.

Workflow:

1. Seed with a small curated set of instructions.
2. Use the LM to generate paraphrases and candidate responses (vary temperature / top-p to increase diversity).
3. Have human raters review and label candidates (accept/reject/edit).
4. Add accepted pairs to the training pool; iterate.

Pros/cons:

- **Pros**: cheap, fast, increases diversity.

- **Cons**: requires human review to filter hallucinations and unsafe outputs.

2.6 Combining methods

Best practice: combine multiple sources. Use a small, high-quality human core and grow breadth using converted datasets and LM-assisted generation with careful review. Weight examples or use curriculum to present higher-quality examples more often during SFT.

3 Preference Learning & Reward Models (detailed)

3.1 Motivation

Humans often have preferences that are hard to encode as a supervised target string. For open-ended outputs, it is easier to collect preference judgments (which of two outputs is better) than to write the "correct" output. Preference learning creates a scalar reward model that predicts human preference and can be used to optimize the LM.

Preference data: triplets (x, o_i, o_j, y) where x is the prompt, o_i, o_j are two model outputs, and y is a human label indicating which output is preferred (e.g., $y = 1$ if o_i preferred over o_j).

3.2 Bradley–Terry model (pairwise preference modeling)

Assume each output o (for a given prompt) has a latent score z_o given by a reward model $r_\phi(x, o)$. Bradley–Terry says the probability o_i is preferred to o_j is:

$$P(o_i \succ o_j \mid x) = \sigma(z_i - z_j) = \frac{1}{1 + e^{-(z_i - z_j)}}.$$

The training objective is to minimize cross-entropy between predicted pairwise probabilities and human labels.

3.3 Numeric example (Bradley–Terry logistic probability & loss)

Toy example: suppose the reward model predicts $z_A = 1.2$ for output A and $z_B = 0.4$ for output B on the same prompt. Then

$$\Delta = z_A - z_B = 0.8.$$

The probability A preferred over B is

$$P(A \succ B) = \sigma(0.8) = \frac{1}{1 + e^{-0.8}} \approx 0.68997.$$

If human label says A was preferred ($y = 1$), the pairwise cross-entropy loss is

$$\mathcal{L} = -\log P(A \succ B) \approx -\log(0.68997) \approx 0.371.$$

If instead human chose B ($y = 0$), loss would be

$$\mathcal{L} = -\log(1 - 0.68997) \approx -\log(0.31003) \approx 1.171.$$

This numeric example shows how score differences translate into probabilities and losses.

3.4 Training a reward model

- **Model form**: reward model $r_\phi(x, o)$ can be a small transformer or the LM head applied to the concatenation of prompt and output.
- **Loss**: pairwise cross-entropy using Bradley–Terry as above.
- **Data**: collect many human comparisons across diverse prompts and candidate outputs.
- **Regularization**: guard against reward model overfitting; use held-out preference sets and check calibration.

3.5 From reward model to policy optimization

Two main families to optimize policies (LMs) using a reward model:

1. **RLHF (Reinforcement Learning from Human Feedback)**: use policy-gradient (e.g. PPO) to optimize the LM to maximize reward, often with a KL penalty to keep policy close to the SFT policy.
2. **DPO (Direct Preference Optimization)**: directly optimize the model parameters to increase preference probability under a Bradley–Terry-like objective without a full RL loop (simpler, with theoretical motivations).

3.6 Concise RLHF pseudocode

```
# Precondition: pretrained LM and SFT-initialized policy pi_theta
# 1. Collect preference data by sampling outputs from pi_theta for prompts
# 2. Train reward model r_phi on human pairwise labels
# 3. Use PPO to update pi_theta to maximize E[r_phi] - beta * KL(pi_theta || pi_ref)
#    where pi_ref is the reference (often SFT) policy and beta controls closeness
```

3.7 Concise DPO idea (intuitive)

DPO frames preference optimization as a classification of which output is preferred using the ratio of probabilities under the current policy vs a reference policy. It yields a loss that can be optimized by gradient descent directly on model parameters and avoids heavy RL machinery.

3.8 Practical numeric toy for reward training + one-step update (very small)

Setup: small LM policy outputs two candidate completions A and B for prompt x. SFT reference policy has probabilities $p_{ref}(A) = 0.6$, $p_{ref}(B) = 0.4$. Current policy has $p_\theta(A) = 0.5$, $p_\theta(B) = 0.5$. Reward model predicts $z_A = 0.9, z_B = 0.2$. Human prefers A.

RLHF intuition (one gradient step): Using a simplified surrogate objective (maximize reward-weighted log-probabilities), the policy update increases probability of A in proportion to reward difference. Numerical demonstration: current log-probabilities are $\log 0.5 = -0.6931$. For small learning rate, the policy's probability of A will increase slightly — use actual RL framework for proper calculation. This toy illustrates directionality: outputs with higher reward get boosted.

3.9 Evaluation & safety checks

- Validate reward model on held-out human preference data.
- Run adversarial prompts to test for reward hacking / exploit behaviors.

- Monitor whether policy optimization increases unintended behavior (e.g., verbosity, gaming reward). Use KL penalties and reward model regularization.

Appendices and practical material

A: Checklist for building instruction-tuned models

- Collect a high-quality human core dataset (1k–10k examples).
- Convert diverse supervised datasets to instructions using varied templates.
- Use LM-assisted generation to expand dataset and have humans review.
- Run SFT with mixed batches (weight high-quality examples more).
- Collect pairwise preference data for policy areas where humans care about trade-offs.
- Train a reward model; evaluate on held-out preference data.
- Use PPO/DPO with a KL penalty to refine the policy.

B: Example LaTeX-friendly diagram (preference pipeline)

C: References and slide source

These notes were written based on the slide deck provided: /mnt/data/6.Transformer.pdf and general best-practices in the literature on RLHF, instruction tuning and reward modeling.

4 Appendix D — Complete Q&A: Attention, Masking, KV Cache, Q/K/V, N2̂, Layers & Heads

This appendix answers, in order, the detailed conceptual and technical questions you asked during reading. It is written to be copy-paste ready into your notebook and to resolve all common confusions.

4.1 D.1 Training vs Inference (recap)

- **Training:** weights are updated; we run full-sequence forward and backward passes; we compute Q,K,V for all tokens in parallel.
- **Inference:** weights are frozen; we generate tokens one-by-one; only the new token's query (and its K,V) are computed in that step — the past K,V are reused from cache.

4.2 D.2 What is causal masking and why does it still lead to $O(N^2)$?

Causal masking ensures that token i cannot attend to tokens $j > i$ by setting those logits to $-\infty$. Even though each row i only uses columns $1..i$, the total number of pairwise comparisons across the sequence is

$$1 + 2 + \cdots + N = \frac{N(N+1)}{2} = O(N^2),$$

so complexity remains quadratic.

4.3 D.3 Masked self-attention (formulas + labels)

Given input $X^{(\ell)} \in \mathbb{R}^{N \times d}$ at layer ℓ and per-head projection matrices,

$$Q^{(\ell)} = X^{(\ell)} W_Q^{(\ell)}, \quad K^{(\ell)} = X^{(\ell)} W_K^{(\ell)}, \quad V^{(\ell)} = X^{(\ell)} W_V^{(\ell)}.$$

Scaled scores and masked attention:

$$S = \frac{QK^\top}{\sqrt{d_{\text{head}}}} + M, \quad A = \text{softmax}(S), \quad \text{attn} = AV.$$

For causal attention $M_{ij} = -\infty$ when $j > i$.

4.4 D.4 Intuition: Q asks, K offers, V contains

- Q_i is "what I (position i) need".
- K_j is "what position j offers".
- The dot product $Q_i \cdot K_j$ measures usefulness; softmax converts to weights $\alpha_{i,j}$.
- V_j is the content mixed by those weights to produce the attention vector for position i.

4.5 D.5 Numeric example (full walkthrough)

(Repeated here concisely so you can paste it into notes.)

```
Sequence X (N=3, d=2):
X = [[1,0], [0,1], [1,1]]
W_Q = [[1,0],[0,1]] (identity)
W_K = [[1,1],[1,0]]
W_V = [[1,0],[0,1]] (identity)

Q = X @ W_Q = X
K = X @ W_K = [[1,1],[1,0],[2,1]]
V = X @ W_V = X

For position 3: Q3 = [1,1]
Scores s = Q3 @ K.T = [2,1,3]
softmax(s)  [0.245, 0.090, 0.665]
attn3 = 0.245*[1,0] + 0.090*[0,1] + 0.665*[1,1] = [0.910, 0.755]
```

4.6 D.6 What does the attention vector represent?

- The attention vector (e.g. $[0.91, 0.76]$) is a compressed representation of position 3 after integrating relevant past tokens.
- Each coordinate is a learned feature; the vector as a whole is used (after FFN and stacking) to compute logits via $\nu = W_{\text{vocab}}^\top h$.
- The vector is what the model uses to predict the next token — it must therefore encode all signals relevant to prediction (syntax, semantics, local context, likely continuations).

4.7 D.7 Why attention costs scale quadratically (another perspective)

- The attention computation forms an $N \times N$ score matrix; even if half the entries are masked away, the number of nonzero comparisons sums to $\approx \frac{N^2}{2}$.

- Memory-wise one stores at least the score/weight matrix and intermediate activations, which also scale as $O(N^2)$.

4.8 D.8 KV cache: what it stores and why per-layer/per-head

- For each layer ℓ and head h we store cached matrices

$$K_{\text{cache}}^{(\ell,h)} \in \mathbb{R}^{T \times d_{\text{head}}}, \quad V_{\text{cache}}^{(\ell,h)} \in \mathbb{R}^{T \times d_{\text{head}}}.$$

- Keys/Values differ across layers and heads because each layer/head has its own projection matrices; therefore the cache is indexed by layer and head (or has a head dimension).

- During generation we only compute Q for the new token and matmul against cached Ks to obtain scores; then we multiply weights by cached Vs for the output.

4.9 D.9 Why K and V stay fixed during inference (intuitive + math)

Intuition: past tokens are *fixed* text; their representations at each layer are final for that step and do not change when new tokens are generated. Keys and values are computed from these representations once and reused.

Math: at layer ℓ, $K = X^{(\ell)} W_K^{(\ell)}$ and $V = X^{(\ell)} W_V^{(\ell)}$. If the rows of $X^{(\ell)}$ for tokens $1..t-1$ remain unchanged, then their projections K_j and V_j remain unchanged.

4.10 D.10 Data layout and efficient inference tips

- Typical cache layout: a tensor shaped (L, B, H, S_cap, d_head).

- Preallocate S_cap to the maximum context length to avoid reallocations.

- Use contiguous memory layouts and batched matmuls; avoid many small concatenations.

- Use large finite negative masks (e.g., -1e9) instead of true -Inf in float16.

4.11 D.11 Layers vs Heads — one final consolidated view

- **Heads:** parallel attention mechanisms inside a layer. Each head learns a different attention pattern ("where to look") and has separate projection matrices.

- **Layers:** stacked blocks. Each block runs multi-head attention + FFN and produces new token representations. Layers learn to combine head outputs into increasingly abstract features ("what it means").

- They are complementary: heads supply diverse signals; layers integrate and refine them across depth.

4.12 D.12 Quick cheat-sheet (copy this into your notebook)

Cheat-sheet
Q = XW_Q (query), K = XW_K (key), V = XW_V (value) — per layer and per head.
Attention: S = QK$^T/sqrt(d_head) + M; A = softmax(S); out = AV.$
$Causal mask M\_ij = 0 if j <= i else - 1e9.$
$KV cache stores K, V for past tokens per (layer, head) so inference only computes Q for new tokens.$
$Total attention work across sequence length N N(N+1)/2 = O(N^2).$

If you want more

I can:

- (A) merge this appendix into earlier Sections 1–4 in a single file (I can output the merged LaTeX),
- (B) add TikZ diagrams that visualize the masked score matrix, cache layout, and per-layer/head structure, or
- (C) produce a runnable PyTorch notebook that implements the toy numeric example and a minimal KV-cached autoregressive loop.

Tell me which option you prefer and I will produce it directly into this document.