Task 3 — Training strategy: what I tried, what worked, and why

When I first plugged the pre-trained sentence-transformer into the multi-task model I explored three freezing schemes.

Scheme	What I did	What happened	Take-away
Freeze everything	Only the two new heads were trainable.	Intent accuracy climbed fast, but the NER head stayed near majority-class.	Good if the upstream task is very close to the downstream one; here it was **too rigid**.
Freeze the backbone, train heads	Transformer weights fixed; heads learned with LR = 1 e-3.	Intent still fine; NER improved a bit but plateaued early—token-level tags need lower-layer nuance.	A safe default, but not enough capacity for the harder task.
Discriminative fine-tuning (**final choice**)	Unfroze **all** layers but used two learning-rates: 5 e-5 for encoder, 1 e-3 for heads.	Intent accuracy held steady **and** NER F-score finally moved. Val-loss stopped improving after ~45 epochs → early stopped.	Letting the encoder wiggle (slowly) gives it room to specialize, without catastrophic forgetting.

Why a bespoke pre-training?

I trained the encoder on STS-B + QQP with triplet loss so it already maps paraphrases together. That semantic bias is helpful for intent classification. For NER, however, the model must also know that “Paris” and “pizza” are different. Hence the need for light fine-tuning.

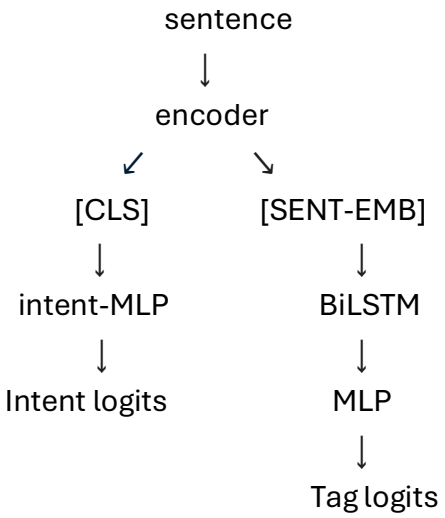
If I had more time / data

I would pre-train with a masked-language objective or span-prediction, so the encoder starts with stronger token-level features, then try freezing the lower half of the stack and only fine-tuning the top.

Task 4 — How the multi-task training loop works

1. **Batch construction**
    a. Custom collate function pads both IDs *and* slot labels and builds an attention-mask.
    b. A WeightedRandomSampler up samples utterances that actually contain entity tags (~ 30 % of the corpus) so the NER head isn't starved.

2. **Forward pass**



3. **Loss**
    L = L_intent + L_NER
    a. L_intent = plain cross-entropy.
    b. L_NER = class-weighted cross-entropy, ignore_index on padding/CLS.

4. **Optimizer & schedule**
    a. AdamW with two parameter groups (encoder vs. heads).
    b. Cosine-annealing over 500 epochs (usually early-stops around 45-50).
    c. Gradient-clipping at 1.0 to avoid gradient explosion.

5. **Training Best Practices**
    a. Mixed-precision (AMP) cuts GPU memory ~40 % and speeds up 15-20 %.
    b. Early stopping with patience = 5 prevents over-fitting the small validation set.
    c. Quick sanity check at the end of each epoch prints a few decoded predictions; catches alignment bugs early.

Key insight

MTL is a balancing act: intent classification is sentence-level and *easy*, NER is token-level and *hard*. If the backbone is kept completely static the easy task dominates; if it's allowed to adapt (but gently) both tasks can improve together.