Final HW review HW4

- 音素识别 Phoneme Recognition(分类)

Load the phoneme data and standardize. Design MLP as probe. The inputs are the feature tensor in different Hubert layers. The middle layers' features have better performance. 使用交叉熵

- MFCC 回归 (回归 13 维 MFCC 标签)

MLP now about dimension is 13. Use R^2 as metric. 越接近 1 说明该层特征对 MFCC 解释力越强。通常指标随层数上升而下降因为模型逐渐抛弃声学特征

- Speaker Embedding Analysis (Clustering)

每一层输出的所有时间步特征向量相加除以总帧数，然后各自运行 t-SNE 算法，压缩到二维可视化 cluster.

- Automatic Speech Recognition (ASR)

数据载入并 Z-score 归一化，训练标签 charactertokenized，用预训练 wav2cev 提取(hubert)高阶音频特征，然后一维卷积下采样(降低计算量)，然后 LSTM，最后线性层投影至词表空间。使用 CTCLoss。推理阶段 CTC Decoder 用 beam search 合并连续重复字符并剔除 blank 标记。

- Text to Speech (TTS)

需要文本和音频的分词器，流匹配模型和声码器(vocoder)。输入序列组织为：

[SOS,text_tokens,TASK_ID,speech_tokens]

为文本和语音 token 分别设置嵌入层，并使用多层 transformer 解码器，并使用因果掩码，一个地方的语音生成只能看到文本和之前的语音 token 信息。计算预测的语音 token 和真实的语音 token 的交叉熵损失(忽略文本)。推理时[SOS, text_tokens, TASK_ID]然后自回归语音 token，直到生成 EOS。最后语音 token-流匹配-Mel 频谱图-vocoder-wav. 评估：wav-ASR-text，和 GT text 比较 WER

HW5 finetune for SQL generation

Zero shot：原始模型做 text2SQL; Finetune：训练时，问题和 SQL 拼接成：Question:[Q] \n\n SQL: [A] <|endoftext|>并且需要用-100 掩盖 question 部分 loss

关于 Prompt：可以是 question，同时可以加上几个例子给模型看进行引导。例子可以是随机的，但是也可以是刻意的：

所有训练集中的问题转化为向量，推理时问题也转化为向量，选择点积前 N 个问题加入 prompt。最后回答中截出 SQL。评估看 str 匹配和执行结果准确性

HW6 Image & Text Retrieval via CLIP

- CLIP image and text encoder 分别处理图片和文本，然后 embedding 归一化，计算余弦相似度分数矩阵。RSA 推理：Speaker: CLIP 分数矩阵每一张图所有文本描述的分数进行 softmax(可缩放)。Listener: 然后对这个新矩阵每一个文本的所有图像的分数进行 softmax。最后 pragmatic listener 拿到一个文本所认为的图片所对应的概率分布就是矩阵中该文本对应的所有图片的概率分布。

- Image Captioning

图像经过预训练视觉模型(ViT)，取最后一层所有 patch 特征经过 MLP 到语言 embedding 维：[Image_Embedding_1, ..., Image_Embedding_n, <Start_Token>]。然后需要解码。训练的时候，输入的是 image embedding+BOS token+Grount Truth Caption Token, teacher forcing 且因果掩码 + loss 只看文字部分。

Knowledge syntax coreference lexical semantics sentiment reasoning: what we learn from LM

COT 局限性：在开放问题不好，很会解释错误答案，longerCOT 通常模型性能更好

Reward hacking — exploiting errors in the reward model to achieve high estimated rewards

- E.g., longer outputs get higher reward, regardless of quality otherwise

Hallucination and miscalibration

- No examples of abstention to questions, so model will never know what it "doesn't know"

Off-policy reward model

- Our reward model isn't trained on outputs the model is likely to produce *now*, after finetuning a bit through RLHF

Generalization of preferences

- Can we learn when preferences are relevant or not? E.g., refusals to "kill a linux process"

Literal listener uses denotational semantics to map each utterance to the probability of all referents

$$p^{\text{Listener}}_{\text{Literal}}(r \mid x) = \frac{[\![x]\!]_r}{\sum_{r' \in R} [\![x]\!]_{r'}}$$

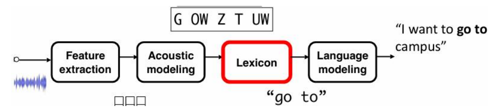
Pragmatic speaker re-normalizes probabilities over utterances given the literal listener's interpretations

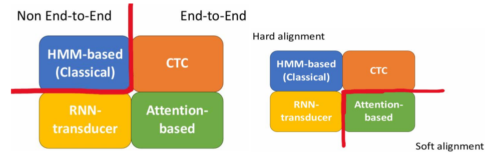
$$p^{\text{Speaker}}_{\text{Pragmatic}}(x \mid r) = \frac{p^{\text{Listener}}_{\text{Literal}}(r \mid x)}{\sum_{x' \in X} p^{\text{Listener}}_{\text{Literal}}(r \mid x')}$$

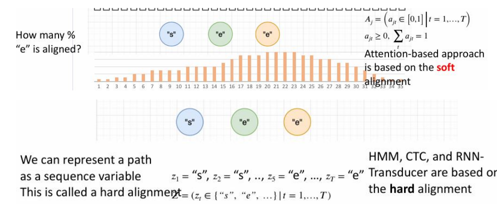
Pragmatic listener takes into account alternative utterances that the speaker *could* have used to refer to a referent, but didn't

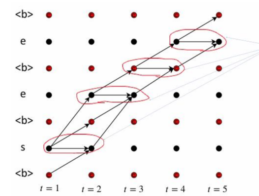
$$p^{\text{Listener}}_{\text{Pragmatic}}(r \mid x) = \frac{p^{\text{Speaker}}_{\text{Pragmatic}}(x \mid r)}{\sum_{r' \in R} p^{\text{Speaker}}_{\text{Pragmatic}}(x \mid r')}$$

Matching problem: Dynamic Warping



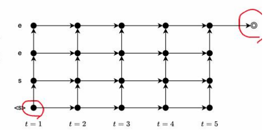






1. Must start at the first <b> or s
2. Must end at the last <b> or e
3. All characters can be repeated
4. <b> can be skipped except when it is inserted between repeated character
   - "s", "<b>", "e": we can skip <b>
   - "e", "<b>", "e": we **cannot** skip <b>

- Alternative alignment paths
  - $W = (w_1, \ldots, w_J)$: J-length label sequence
  - $O = (o_1, \ldots, o_T)$: T-length input sequence
- Similar to CTC, consider all possible paths in the right-side trellis
  - Start: left bottom corner, i.e., <s> (start of sentence) at $t = 1$
  - End: top right corner
  - ↑: output token (it does not consume the input frame, unlike CTC)!
  - →: no output (∅)
  - f^{-1}: W=('s', 'e', 'e') → Z=(∅, 's', ∅, ∅, 'e', 'e', ∅, ∅), ('s', ∅, 'e', ∅, ∅, ∅, 'e', ∅) etc. if $T = 5$ and $J = 3$
  - $f: Z \to W$ is a many to one mapping (we can just remove ∅
- Consider all possible paths including the case that it allows the output of the token W



CTC, RNN-transducer, and HMM are represented by a similar function form, i.e., $\sum_{Z \text{ or } S} \prod_t f(t)$

All possible alignments are constrained by each method

The most difficult issue in speech recognition: the input and output lengths are different

- We need some alignments

The soft alignment case

- We use an attention-based neural network (this will be introduced later).

The hard alignment cases

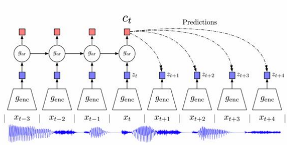
- We explicitly introduce an alignment path Z.
- We can use it for CTC, RNN-transducer, and HMM-based approach

Self-su: Contrastive,Predictive,Generative

$$f_k(x_{t+k}, c_t) = \exp\left(z_{t+k}^T W_k c_t\right)$$

$$\mathcal{L}_N = -\mathop{\mathbb{E}}_{X}\left[\log \frac{f_k(x_{t+k}, c_t)}{\sum_{x_j \in X} f_k(x_j, c_t)}\right]$$

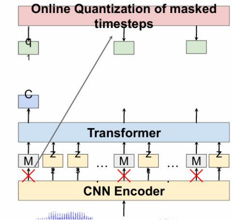
InfoNCE maximizes the mutual information between the input signal and the learned latent variables C.
Strategies for sampling negative and positive examples determine the nature of representations, e.g., whether they are good for ASR or Speaker ID.



CPC(up) wav2vec(down)

- The goal is to maximize the similarity between the learned contextual representation and the quantized input features

$$\mathcal{L}_m = -\log \frac{\exp(sim(\mathbf{c}_t, \mathbf{q}_t)/\kappa)}{\sum_{\tilde{\mathbf{q}} \sim \mathbf{Q}_t} \exp(sim(\mathbf{c}_t, \tilde{\mathbf{q}})/\kappa)}$$

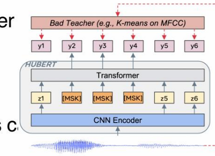


Contrastive(above two) Predictive(Hu)

- Small codebook sizes, e.g. 100, 500.
- The loss is only applied over masked regions.

$$L_m(\theta; X, M, Y) = \sum_{t \in M} \log p(y_t \mid \tilde{X}, t)$$

- The learned latent features can be quantized for another learning iteration.

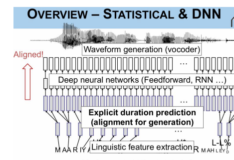


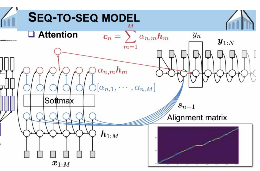
How to model a multimodal conditional distribution p(x|y)?

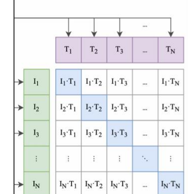
- Autoregressive, GAN, VAE, Flow, Diffusion Model, etc
- Since L1/L2 can be applied to mel-spectrogram, while cannot be directly applied to waveform
- Advanced generative models are developed faster in vocoder than in acoustic model, but finally acoustic models catch up ☺

Generative models for acoustic model/vocoder

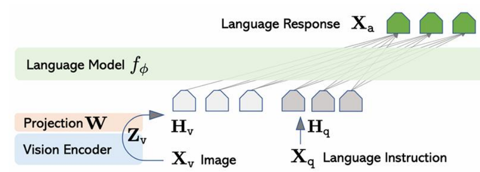
- Text to speech mapping p(x|y) is multimodal, since one text can correspond to multiple speech variations
  - Acoustic model, phoneme-spectrogram mapping: duration/pitch/energy/formant
  - Vocoder, spectrogram-waveform mapping: phase



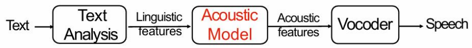


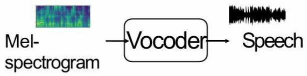


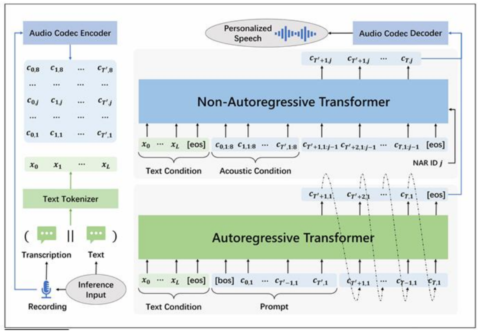
```
# image_encoder - ResNet or Vision Transformer
# text_encoder  - CBOW or Text Transformer
# I[n, h, w, c] - minibatch of aligned images
# T[n, l]       - minibatch of aligned texts
# W_i[d_i, d_e] - learned proj of image to embed
# W_t[d_t, d_e] - learned proj of text to embed
# t             - learned temperature parameter

# extract feature representations of each modality
I_f = image_encoder(I) #[n, d_i]
T_f = text_encoder(T)  #[n, d_t]

# joint multimodal embedding [n, d_e]
I_e = l2_normalize(np.dot(I_f, W_i), axis=1)
T_e = l2_normalize(np.dot(T_f, W_t), axis=1)

# scaled pairwise cosine similarities [n, n]
logits = np.dot(I_e, T_e.T) * np.exp(t)

# symmetric loss function
labels = np.arange(n)
loss_i = cross_entropy_loss(logits, labels, axis=0)
loss_t = cross_entropy_loss(logits, labels, axis=1)
loss   = (loss_i + loss_t)/2
```

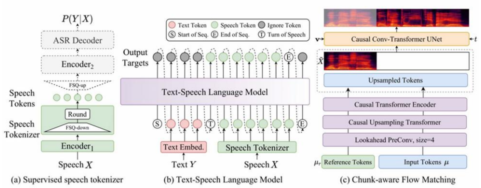


- Predict acoustic features from linguistic features









Cozyvoice2 (Du et al., 2024)

Preliminary: generative vs. discriminative models

- Generative model: parameterizes a joint distribution over random variables X and Y
- Discriminative model: parameterizes a conditional distribution of target Y given observation X

Classification: X = instance features, Y = instance class

Training objective: maximize **joint** likelihood of training data (maximum likelihood estimation)

First-token POS tag
Emission probabilities
Transition probabilities
Probability of stopping

$$\sum_i \log P\left(\overline{y}^{(i)}, \overline{x}^{(i)}\right)$$
$$= \sum_{i=1}^{N} \left(\log P\left(y_1^{(i)}\right) + \sum_{j=1}^{|\overline{x}^{(i)}|} \log P\left(x_j^{(i)} \mid y_j^{(i)}\right) + \sum_{j=1}^{|\overline{x}^{(i)}|} \log P\left(y_j^{(i)} \mid y_{j-1}^{(i)}\right) + \log P\left(\texttt{STOP} \mid y_{|\overline{x}^{(i)}|}^{(i)}\right) \right)$$

training data

they can
N V

they fish
N V

start probabilities

	Count	$P(y_1)$
N	3	0.75
V	1	0.25

transition probabilities

	N	V	STOP
N	1	3	1
V	1	1	3

$p(y_j \mid y_{j-1})$

	N	V	STOP
N	0.2	0.6	0.2
V	0.2	0.2	0.6

$$v_1(\tilde{y}) = \log P(x_1 \mid \tilde{y}) + \log P(\tilde{y})$$
$$v_i(\tilde{y}) = \log P(x_i \mid \tilde{y}) + \max_{\tilde{y}_{\text{prev}}} \left(\log P(\tilde{y} \mid \tilde{y}_{\text{prev}}) + v_{i-1}(\tilde{y}_{\text{prev}})\right)$$

- The best tag sequence has the score:

$$\max_{\tilde{y}_n,\ldots,\tilde{y}_1} \left(\log P(\texttt{STOP} \mid \tilde{y}_n) + \log P(x_n \mid \tilde{y}_n) + \cdots + \log P(x_1 \mid \tilde{y}_1) + \log P(\tilde{y}_1)\right)$$

To compute $\max_{\tilde{y}} \log P(\overline{x}, \overline{y})$:

```
v = 0^{|x̄|×|P|}
for i = 1 ... |x̄|
  for ỹ in P
    if i == 1:
      v_1(ỹ) = log P(x_1 | ỹ) + log P(ỹ)
    else:
      v_i(ỹ) = log P(x_i | ỹ) + max_{ỹ_prev} (log P(ỹ | ỹ_prev) + v_{i-1}(ỹ_prev))
return v_{|x̄|+1}(STOP)
```

keep track of this over time to reconstruct the maximum-probability sequence $\arg\max_{\tilde{y}} \log P(\overline{x}, \overline{y})$

Likelihood: given a sequence of observations, how likely is that sequence according to the HMM?
Easy — just compute using our decomposition of P(x, y)

Decoding: given a sequence of observations, what's the most likely sequence of hidden states?
Easy — use Viterbi

Learning: given a sequence of observations and the set of possible states, what are the HMM parameters that maximize the probability of the sequence?
Easy — compute using counts in data

$$\alpha \in \mathbb{R}_{0:1}^{|\overline{x}| \times |\mathcal{S}|}$$
$$\alpha_1(s) = p_i(s) p(x_1 \mid s)$$
$$\alpha_i(s) = \sum_{s' \in \mathcal{S}} \alpha_{i-1}(s') p_t(s \mid s') p_e(x_i \mid s)$$

To compute $p(\overline{x})$:

```
alpha = 0^{|x̄|×|S|}
for i = 1 ... |x̄|
  for s in states:
    if i == 1:
      alpha[i, s] = p_i(s) p(x_1 | s)
    else:
      for s' in states:
        alpha[i, s] += alpha[i-1, s'] p_t(s | s') p_e(x_i | s)

return sum([alpha[|x̄|, s] for s in states])
```

Base case: $\beta_{|\overline{x}|}(s) = 1$

Recurrence relation: $\beta_i(s) = \sum_{s' \in \mathcal{S}} p_t(s \mid s') p_e(x_i \mid s) \beta_{i+1}(s')$

Termination: $p(\overline{x}) = \sum_{s \in \mathcal{S}} p_i(s) p_e(x_1 \mid s) \beta_1(s)$

```
initialize p̂_t(s | s') and p̂_e(x | s)
until convergence, iterate over examples x̄:
  E-step: assuming probabilities are correct,
          compute pseudocounts
     compute α and β for x̄ using current probs
     compute temporary counts
```

$$\gamma_i(s) = \frac{\alpha_i(s)\beta_i(s)}{\sum_{s' \in \mathcal{S}} \alpha_i(s)\beta_i(s')}$$

$$\xi_i(s, s') = \frac{\hat{p}_t(s, s')\hat{p}_e(x_i \mid s)\alpha_i(s)\beta_{i+1}(s')}{\sum_{\tilde{s} \in \mathcal{S}} \sum_{\tilde{s}' \in \mathcal{S}} \hat{p}_t(\tilde{s}, \tilde{s}')\hat{p}_e(x_i \mid \tilde{s})\alpha_i(\tilde{s})\beta_{i+1}(\tilde{s}')}$$

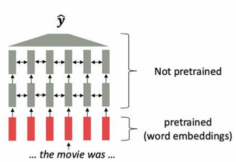
```
  M-step: assuming counts are correct,
          recompute probabilities
```

$$\hat{p}_t(s \mid s') = \frac{\sum_{i=1}^{|\overline{x}|} \xi_i(s, s')}{\sum_{s \in \mathcal{S}} \sum_{i=1}^{|\overline{x}|} \xi_i(s, s')} \qquad \hat{p}_e(x \mid s) = \frac{\sum_{i=1, x_i = x}^{|\overline{x}|} \gamma_i(s)}{\sum_{i=1}^{|\overline{x}|} \gamma_i(s)}$$

One popular pre-LLM approach

E.g., Skip-Gram, GloVe — just download and initialize your favorite model with these embeddings

Then train the model for your target task



In self-refinement, we iteratively:

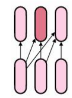
- Generate: map from task description to answer/solution, possibly given existing critique of past answers/solutions
- Verify: map from task description and a proposed answer/solution to some type of feedback

Low-resource languages or dialects (e.g., African American English)

Non-written languages (e.g., American Sign Language)

Language from people who aren't on the web (e.g., older adults)

Data that is increasingly getting excluded from scraping, for better or worse



Decoder-only
Probability of each word depends only on the previous tokens generated so far

$$p(y_i) \propto f(\langle y_1, \ldots, y_{i-1} \rangle)$$

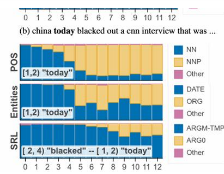
Better at language modeling when we train for longer, increase our dataset size, and increase the model size

Scaling laws tell us what test loss to expect given the amount of compute, data, and parameters.

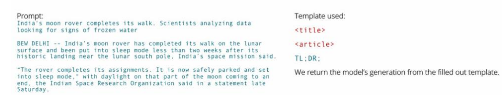
One conclusion: we can reliably improve performance if we keep scaling up (data, model size, time for training)

Another conclusion: we can experiment with smaller models, and trends will probably generalize to larger models

Intermediate representations of BERT at different layers contain sufficient information to perform well on NLP tasks, without any task-specific training!

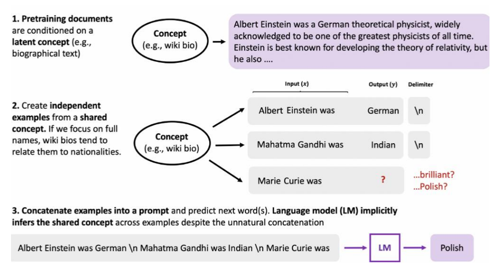


Main idea: format input to the model as if it were generic web text data.



Prompt:
The dog chased a squirrel at the park. = 那只狗在公园里追一只松鼠。
I was late for class. = 我上课迟到了。
The hippopotamus ate my homework. =

Template Used:
<example1_en> = <example1_zh>
<example2_en> = <example2_zh>
<query_en> =



Main idea: "prime" model to generate step-by-step solution to input problem

COT; Structured Prompting

Main idea: prompt LMs to "call" tools, e.g., by interleaving language output with calls to a calculator:

Goal during training: find a prompt that maximizes some reward (e.g., accuracy) over the training dataset

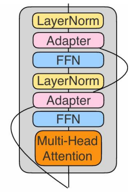
$$\arg\max_{p \in \mathcal{V}^\dagger} \mathop{\mathbb{E}}_{x \in \mathcal{D}} \mathcal{R}(y \sim \text{LLM}(px))$$

Optimize:

$$\arg\max_{p \in \mathbb{R}^d} \mathop{\mathbb{E}}_{(x,y) \in \mathcal{D}} \text{LLM}(y \mid [p; \phi(x)])$$

At inference time, always prepend embedding *p* to inputs

- Modify the network directly by injecting additional parameters into transformer cells
- Initialize the adapter as an identity function
- Finetune **only** the adapter parameters, keeping everything else frozen
- Pretty fast to train (especially compared to full fine-tuning)
- But adding layers makes the model larger, and inference slower



We can express the new value as $W' = W + \Delta W$

- In DiffPruning: we'd just learn ΔW directly
- Can we learn even fewer parameters?

$$\Delta W = BA$$
$$B \in \mathbb{R}^{d \times r}$$
$$A \in \mathbb{R}^{r \times k}$$

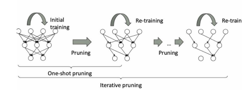
Low-rank: $r \ll \min(d, k)$

At the beginning of finetuning, initialize:
$B = \mathbf{0}$
$A \sim \mathcal{N}(0, \sigma^2)$
(so that ΔW behaves as identity function)

Main principle: use lower-precision representations of network parameters during inference

Reduces the space required to store the model during inference

- Remove lowest-magnitude weights (set values to 0)
- Re-train network (freezing removed weights)
- Iterate between pruning and re-training



Activations at input to each layer will be increasingly out-of-distribution!

Instead: train model to expect quantized inputs at each layer

- Forward pass: quantize
- Backward pass: don't quantize

Main idea: just train a new network (possibly from scratch) on task-specific data sampled from a much larger model

No need for access to larger model's weights or output probabilities, just its outputs

Basic premise: adjust language model probabilities to be conditioned on inputs generated in a more human-friendly interface

- Independently sample candidate responses from the instruction-tuned model
- Ask an annotator to rank the set of candidates
- Train a model to predict the scalar quality of independent candidates, using the principle that if some response A is ranked higher than response B, it's higher quality
- Fine-tune the instruction-tuned model via reinforcement learning (RL), with rewards assigned by this auxiliary model

- First, convert preference data to training data for this model:

$$x, \langle \tilde{y}_1, \ldots, \tilde{y}_N \rangle \quad r(\tilde{y}_i) \ge r(\tilde{y}_{i+1}) \quad \longrightarrow \quad \mathcal{D}_{\text{pref}} = \{(x, \tilde{y}_w, \tilde{y}_l)\} \quad r(\tilde{y}_w) > r(\tilde{y}_l)$$

- Then optimize *r* to give higher scores to winning completions vs. losing completions:

$$\mathcal{L}(\theta) = -\frac{1}{\binom{N}{2}} \mathbb{E}_{(x, \tilde{y}_w, \tilde{y}_l \sim \mathcal{D}_{\text{pref}})} \log(\sigma(r(x, \tilde{y}_w; \theta) - r(x, \tilde{y}_l; \theta)))$$

$$x \sim \mathcal{D}_{\text{prompts}}$$
$$\tilde{y}_t \sim \pi(\cdot \mid x, \tilde{y}_{<t}; \theta)$$
$$s = r(x, \tilde{y}; \theta_{\text{reward}})$$

Objective to maximize:

$$\mathbb{E}_{(x,\tilde{y})} \left(s - \beta \log \left(\frac{\pi(\tilde{y} \mid x; \theta)}{\pi(\tilde{y} \mid x; \theta_{\text{instruct}})} \right) \right) + \mathbb{E}_{x \in \mathcal{D}_{\text{pretrain}}} \log \pi(x; \theta)$$

Reward maximization — KL divergence with instruction-tuned model — Base LM objective