CS11-711 Advanced NLP

Debugging and Understanding NLP Models

Graham Neubig

Carnegie Mellon University
Language Technologies Institute

Site
https://phontron.com/class/anlp2022/

w/ Some Slides by Danish Pruthi

A Typical Situation

- You’ve implemented an NLP system based on neural networks
- You’ve looked at the code, and it looks OK
- It has low accuracy, or makes incomprehensible errors
- **What do I do?**

Three Model Understanding Dimensions

- **Debugging Implementation:** Identifying problems in your implementation (or assumptions)
- **Actionable Evaluation:** Identifying typical error cases and understanding how to fix them
- **Interpreting Predictions:** Examining individual predictions to dig deeper

Debugging

In Neural Net Models, Debugging is Paramount!

- Models are often **complicated and opaque**
- **Everything is a hyperparameter** (network size, model variations, batch size/strategy, optimizer/learning rate)
- Non-convex, stochastic optimization has **no guarantee of decreasing/converging loss**

Possible Causes

- **Training time problems**
  - Lack of model capacity
  - Poor training algorithm
  - Training time bug
- **Test time problems**
  - Disconnect between training and test
  - Failure of search algorithm
- **Overfitting**
- **Mismatch between optimized function and eval**

Don't debug all at once! Start top and work down.

Debugging at Training Time

Identifying Training Time Problems

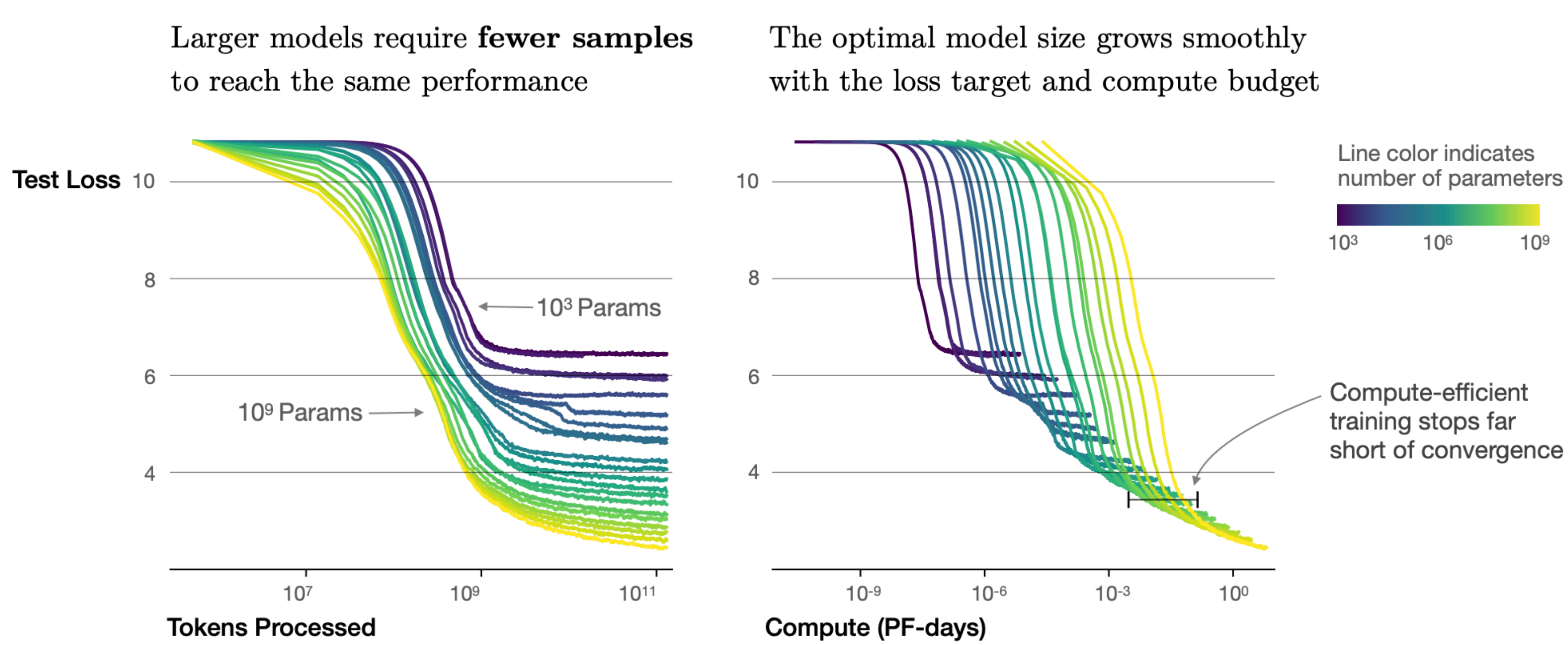
- Look at the **loss function** calculated on the **training set**
  - Is the loss function going down?
  - Is it going down basically to zero if you run training long enough (e.g. 20-30 epochs)?
  - If not, does it go down to zero if you use very small datasets?

Is My Model Too Weak?

- Larger models tend to perform better, esp. when pre-trained (e.g. Raffel et al. 2020)

Model	GLUE Average	CoLA Matthew's	SST-2 Accuracy	MRPC F1	MRPC Accuracy	STS-B Pearson	STS-B Spearman
Previous best	89.4[a]	69.2[b]	97.1[a]	**93.6**[b]	**91.5**[b]	92.7[b]	92.3[b]
T5-Small	77.4	41.0	91.8	89.7	86.6	85.6	85.0
T5-Base	82.7	51.1	95.2	90.7	87.5	89.4	88.6
T5-Large	86.4	61.2	96.3	92.4	89.9	89.9	89.2
T5-3B	88.5	67.1	97.4	92.5	90.0	90.6	89.8
T5-11B	**90.3**	**71.6**	**97.5**	92.8	90.4	**93.1**	**92.8**

- Larger models can learn with fewer steps (Kaplan et al. 2020, Li et al. 2020)



Trouble w/ Optimization

- If increasing model size doesn't help, you may have an optimization problem
- Check your
  - **optimizer** (an Adam variant is standard)
  - **learning rate** (is the rate you're using standard, are you using decay?)
  - **initialization** (if from scratch, are you using a reasonable initialization range)
  - **minibatching** (are you using sufficiently large batches?)
- Pay attention to these details when replicating previous work

Debugging at Test Time

Training/Test Disconnects

- Usually your loss calculation and prediction will be implemented in different functions
- Especially true for structured prediction models (e.g. encoder-decoders)
- Like all software engineering: **duplicated code is a source of bugs**!
- Also, usually loss calculation is minibatched, generation not.

Debugging Minibatching

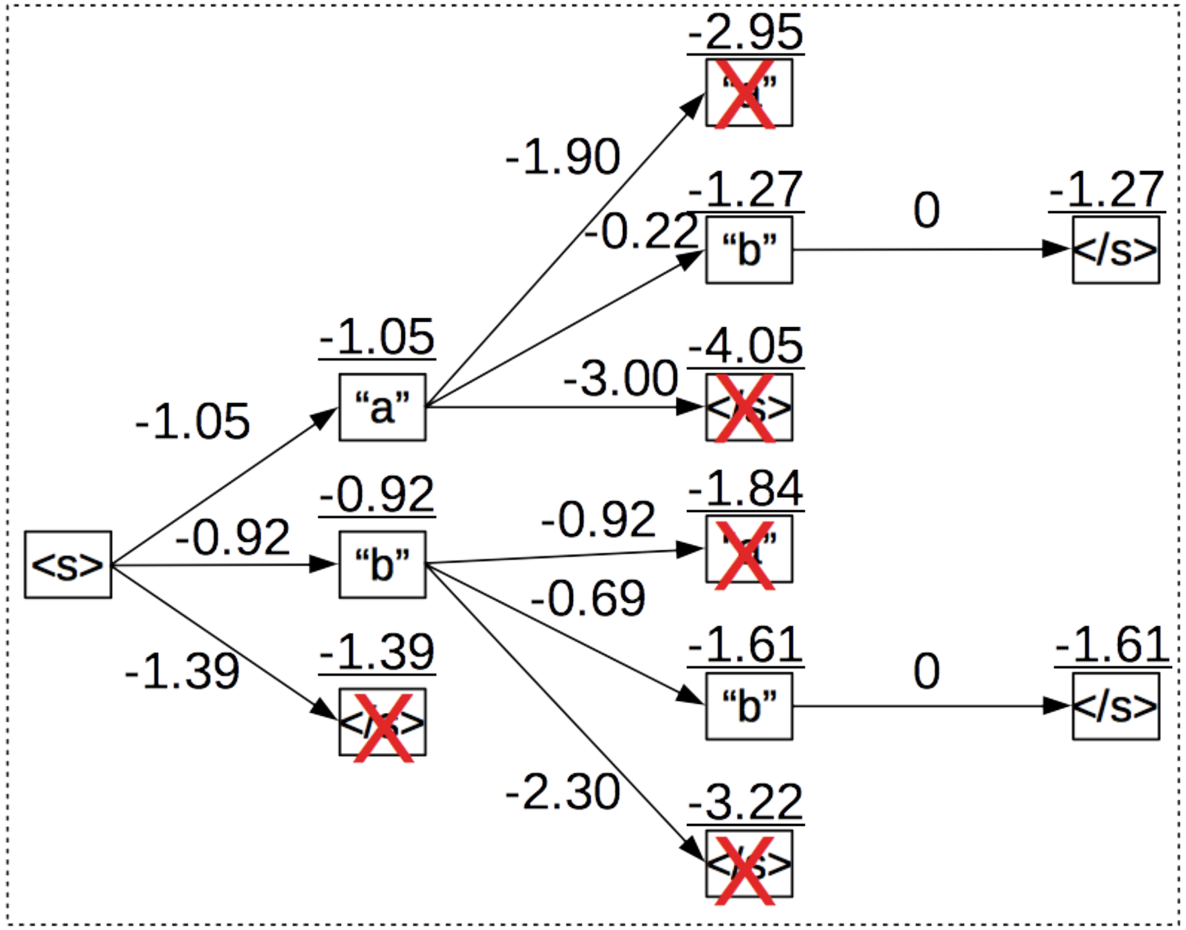
- Debugging mini-batched loss calculation
  - Calculate loss with **large batch size** (e.g. 32)
  - Calculate loss for **each sentence individually and sum**
  - The values should be the same (modulo numerical precision)
- Create a unit test that tests this!

Debugging Structured Generation

- Your decoding code should get the same score as loss calculation
- Test this:
    - Call **decoding function**, to generate an output, and keep track of its score
    - Call **loss function** on the generated output
    - The score of the two functions should be the same
- Create a unit test doing this!

Beam Search

- Instead of picking one high-probability word, maintain several paths



Debugging Search

- As you make search better, the **model score** should get better (almost all the time)
- Search w/ varying beam sizes and make sure you get a better overall model score with larger sizes
- Create a unit test testing this!

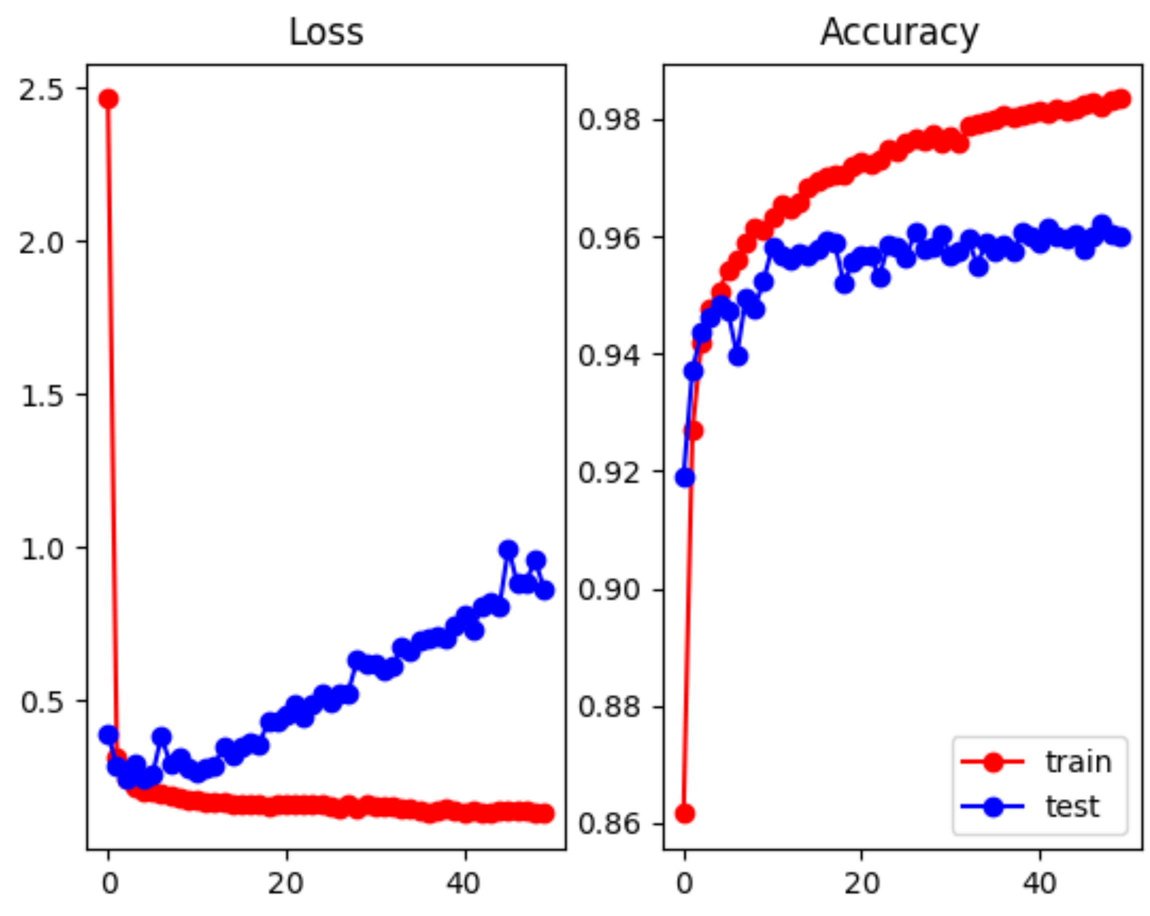
Mismatch b/t Optimized Function and Evaluation Metric

Loss Function, Evaluation Metric

- It is very common to optimize for maximum likelihood for training
- But even though likelihood is getting better, accuracy can get worse

Example w/ Classification

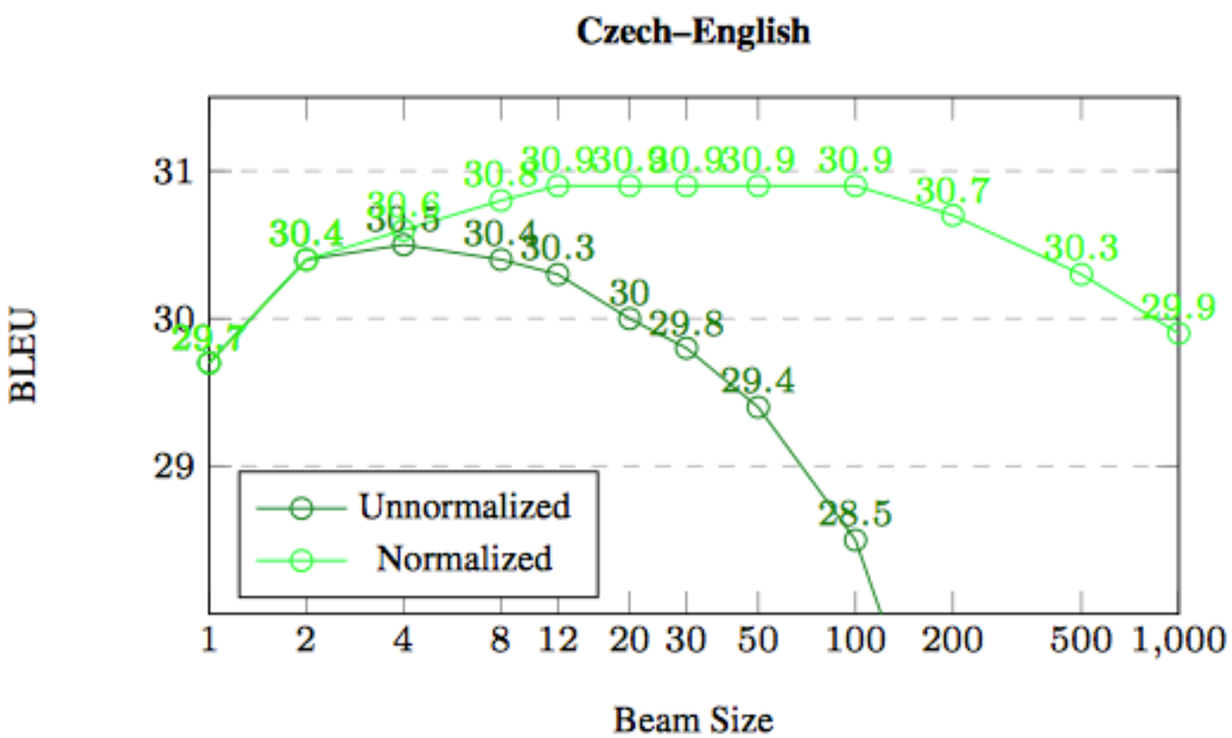
- Loss and accuracy are de-correlated (see dev)

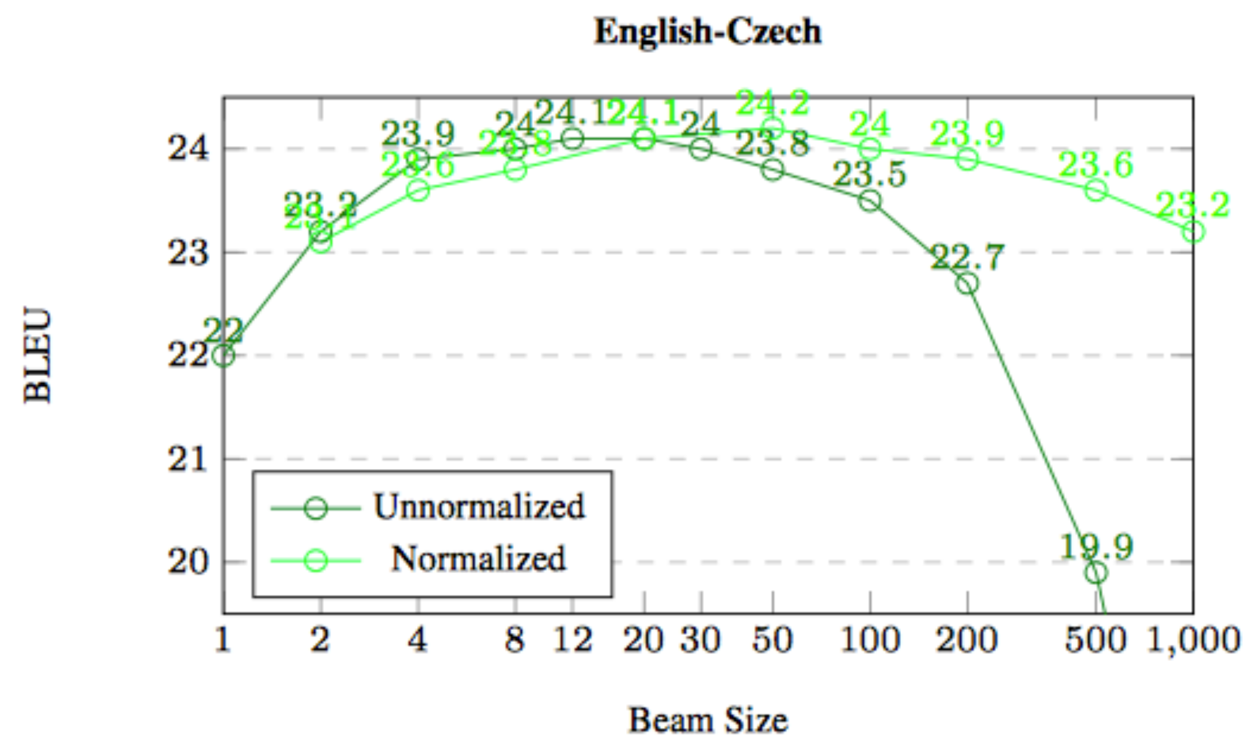


- Why? Model gets more confident about its mistakes.

A Starker Example
(Koehn and Knowles 2017)

- Better search (=better model score) can result in worse BLEU score!

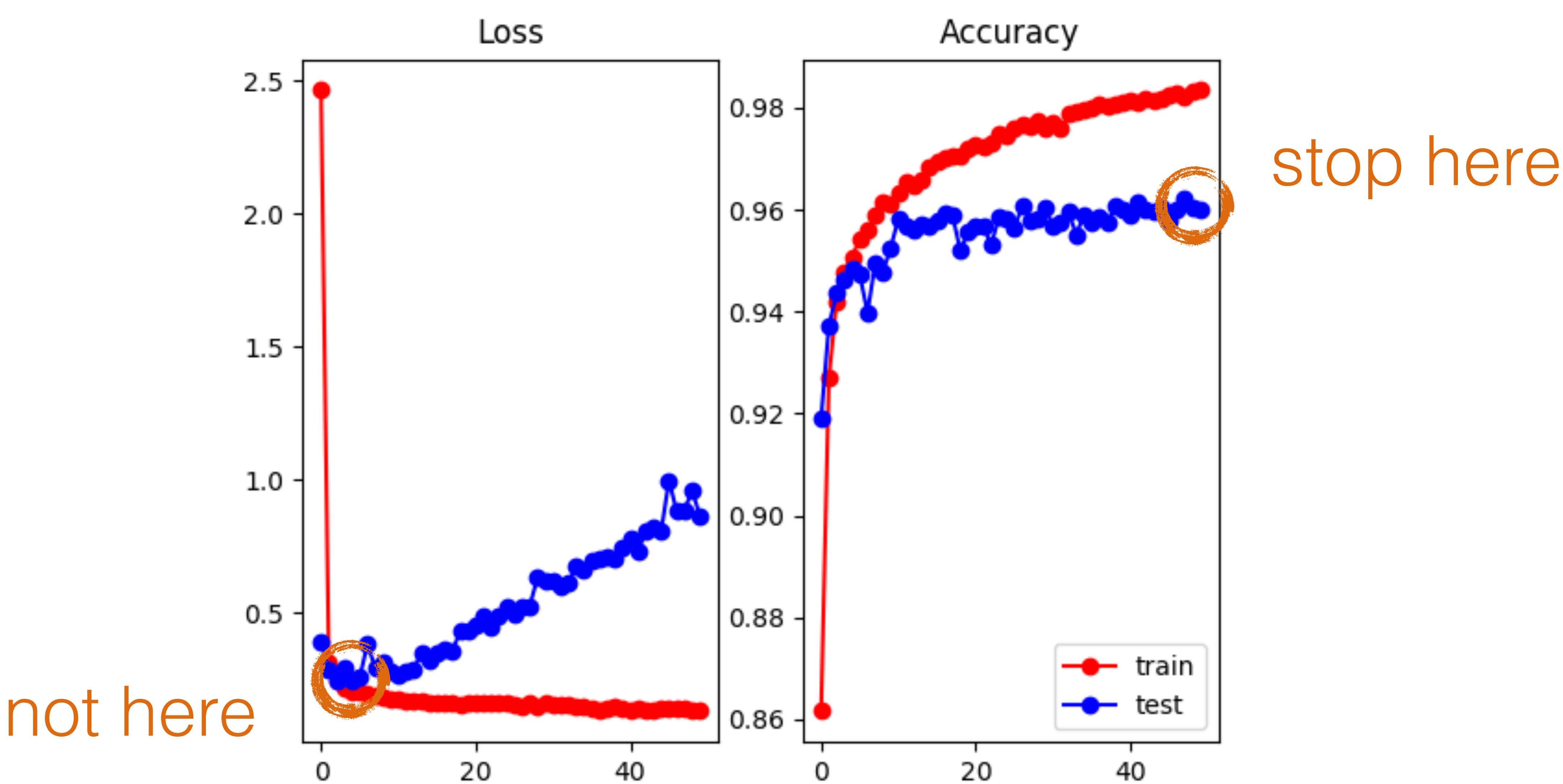




- Why? Shorter sentences have higher likelihood, better search finds them, but BLEU likes correct-length sentences.

Managing Loss Function/Eval Metric Differences

- Most principled way: use a method like reinforcement learning
- Easier way: Early stopping w/ evaluation metric



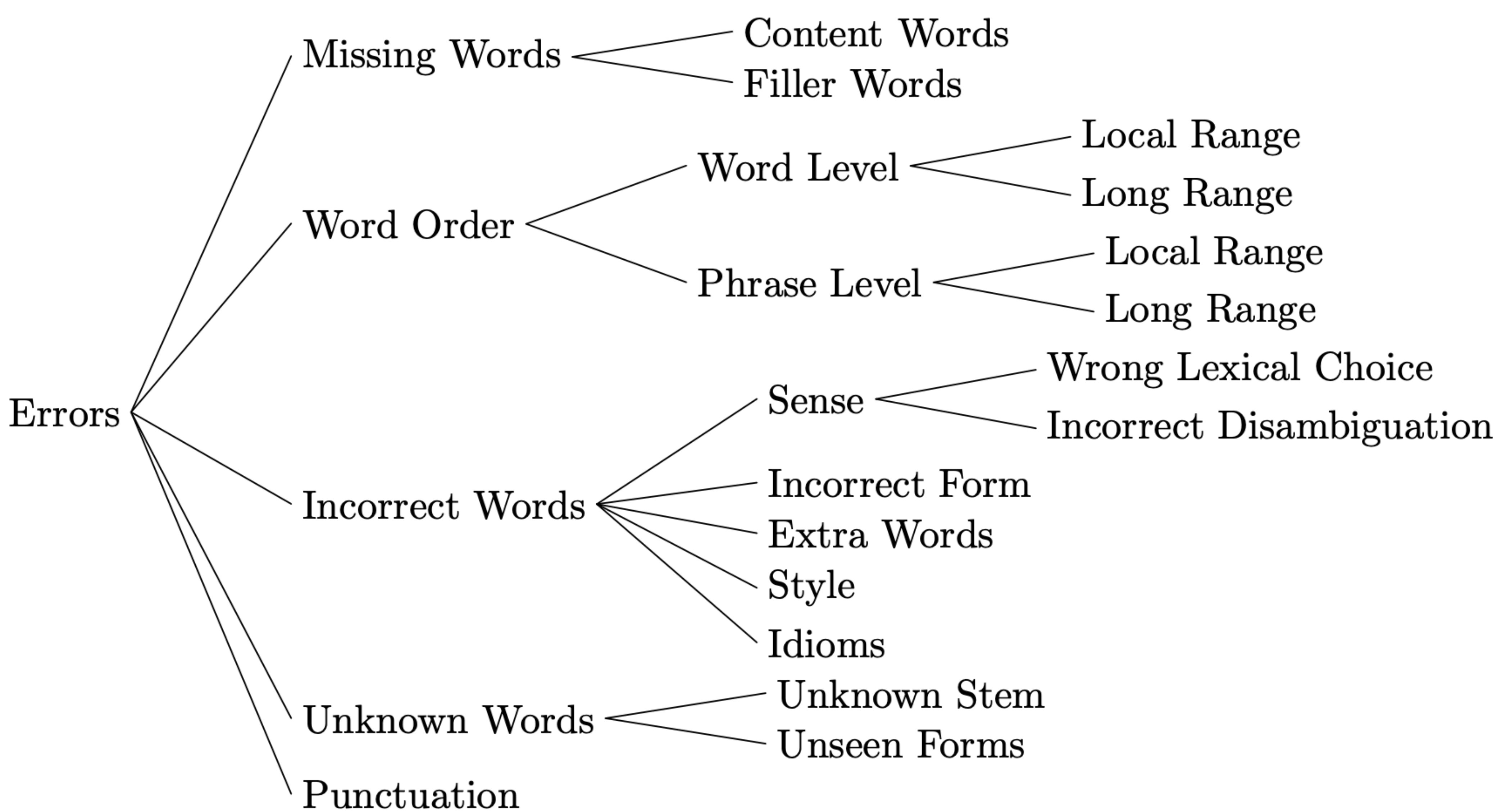
Actionable Evaluation

Look At Your Data!

- Both bugs and research directions can be found by **looking at your model outputs**
- The first word of the sentence is dropped every generation
  > went to the store yesterday
  > bought a dog
  → implementation error?
- The model is consistently failing on named entities
  → need a better model of named entities?

Systematic Qualitative Analysis of Model Errors

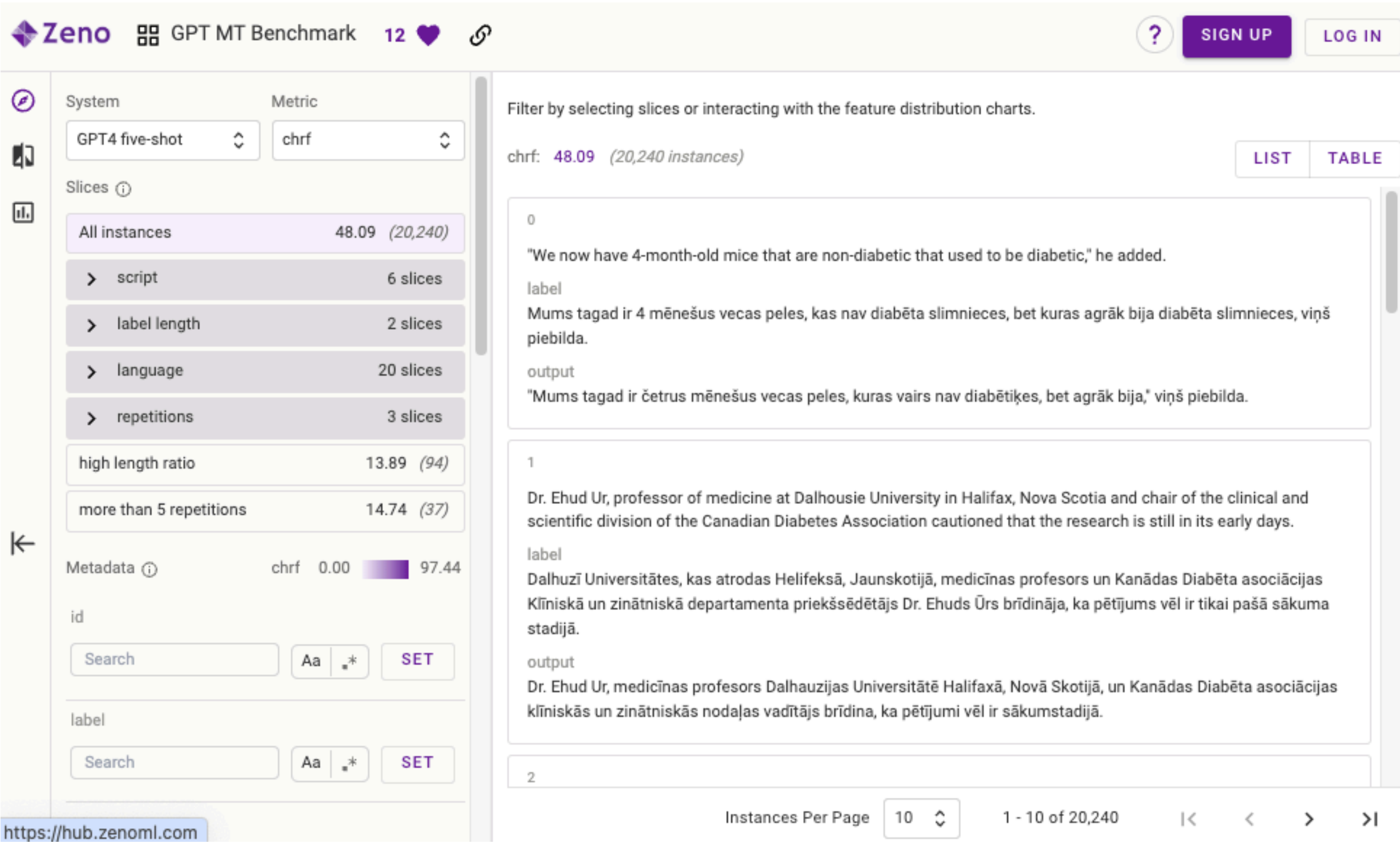
- **Look at 100-200 errors**
- Try to **group them** into a typology (pre-defined or on the fly)
- Example: Vilar et al. (2006)



Quantitative Analysis

- Measure gains quantitatively. What is the phenomenon you chose to focus on? Is that phenomenon getting better?
  - **You focused on low-frequency words:** is accuracy on low frequency words increasing?
  - **You focused on syntax:** is syntax or word ordering getting better, are you doing better on long-distance dependencies?
  - **You focused on search:** how many search errors are being reduced?

Example: Zeno



https://zenoml.com

Questions?