NLP LAB 2021 - VDA

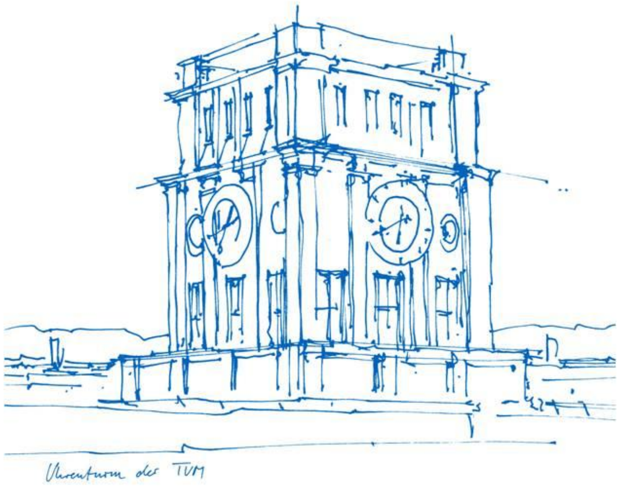
Ananta Bhattarai, Sophie Schoen, Viviane Rehor

Technische Universität München

Department of Informatics

Research Group Social Computing

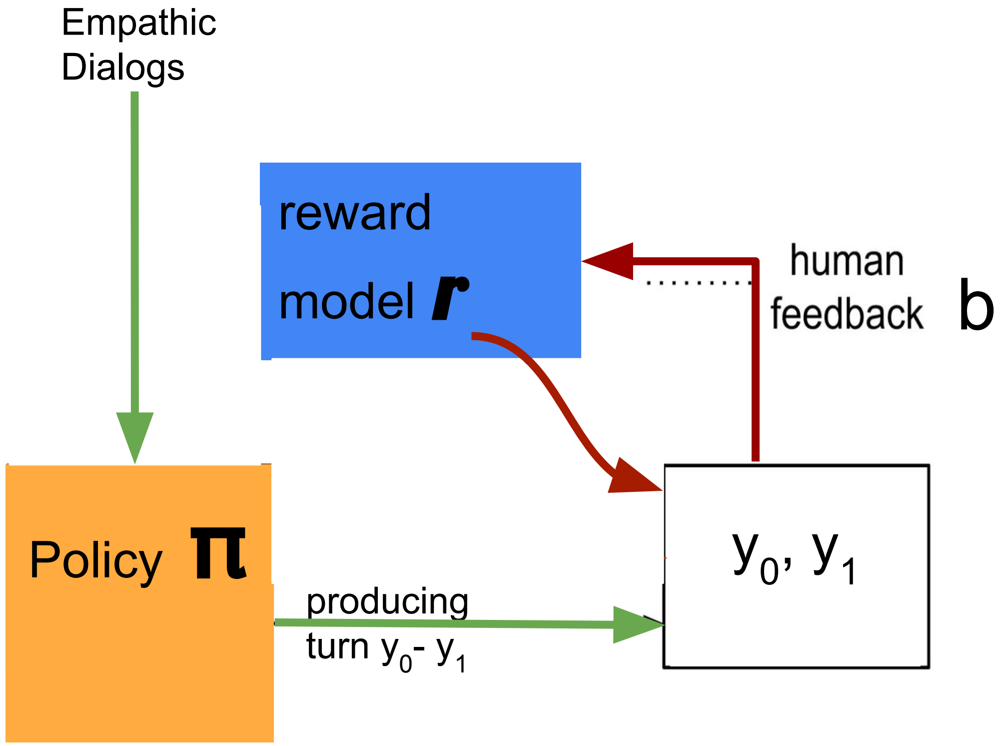
Munich, July 07 2021



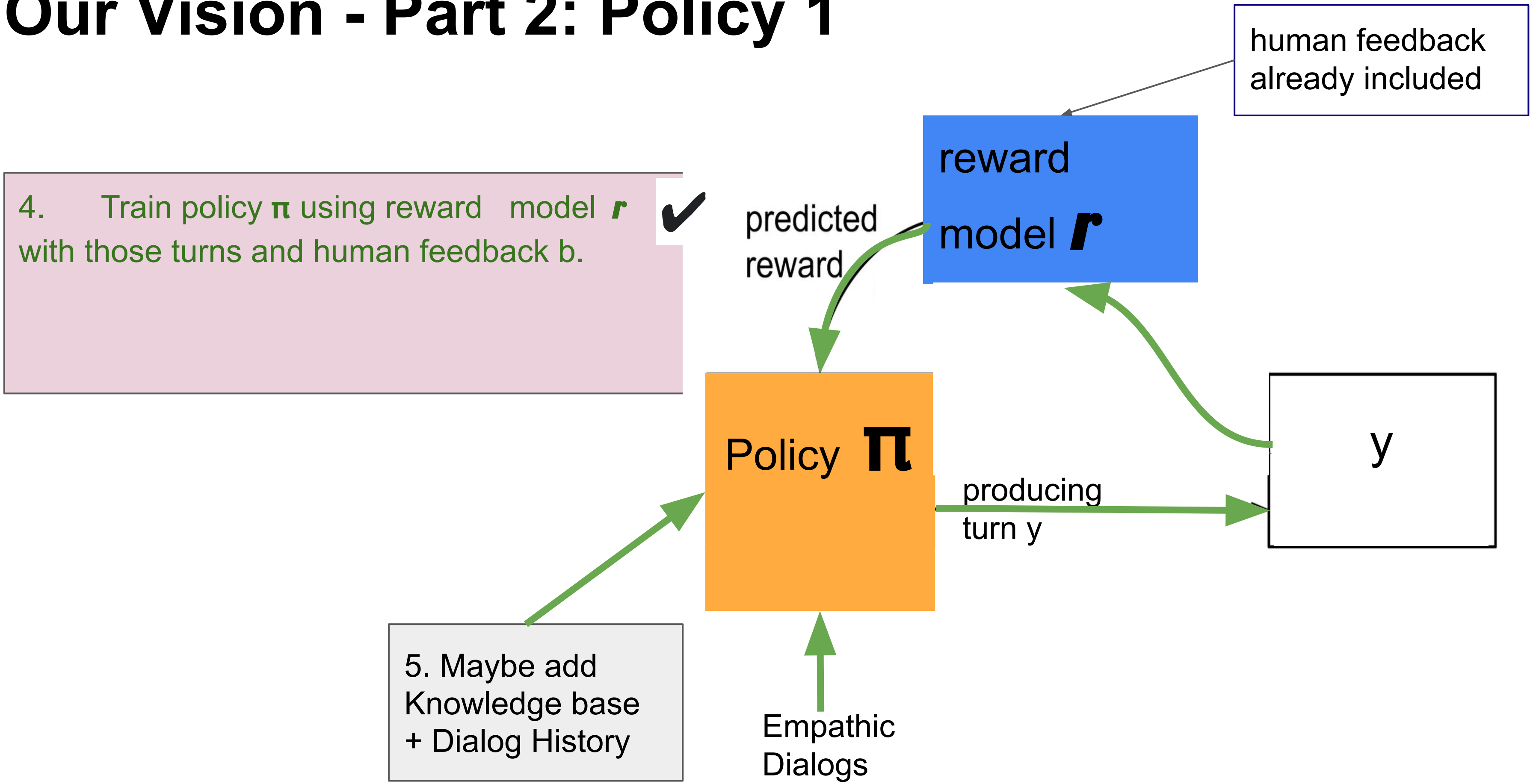
Our Vision - Part 1

1. Creating a Baseline: Fine-Tuning GPT-2 with Empathic Dialogues datasets
   a. Data Preprocessing (Encoding, Decoding & Importing Data) ✔
   b. Fine-Tuning to get π ✔
2. Producing turns y_0- y_1 to give human feedback on
   a. *With test/validation set of* ✔

      Empathic Dialogs dataset
3. Train reward model r with those turns and human feedback b ✔



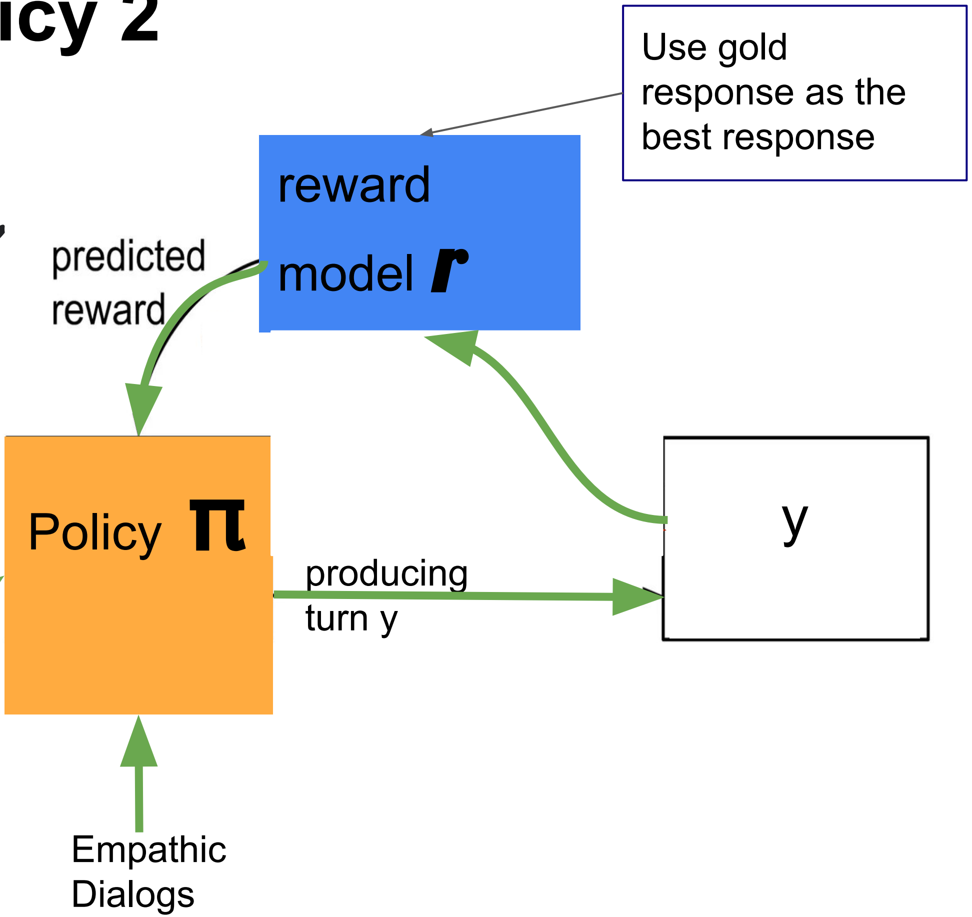
Our Vision - Part 2: Policy 1



Our Vision - Part 2: Policy 2

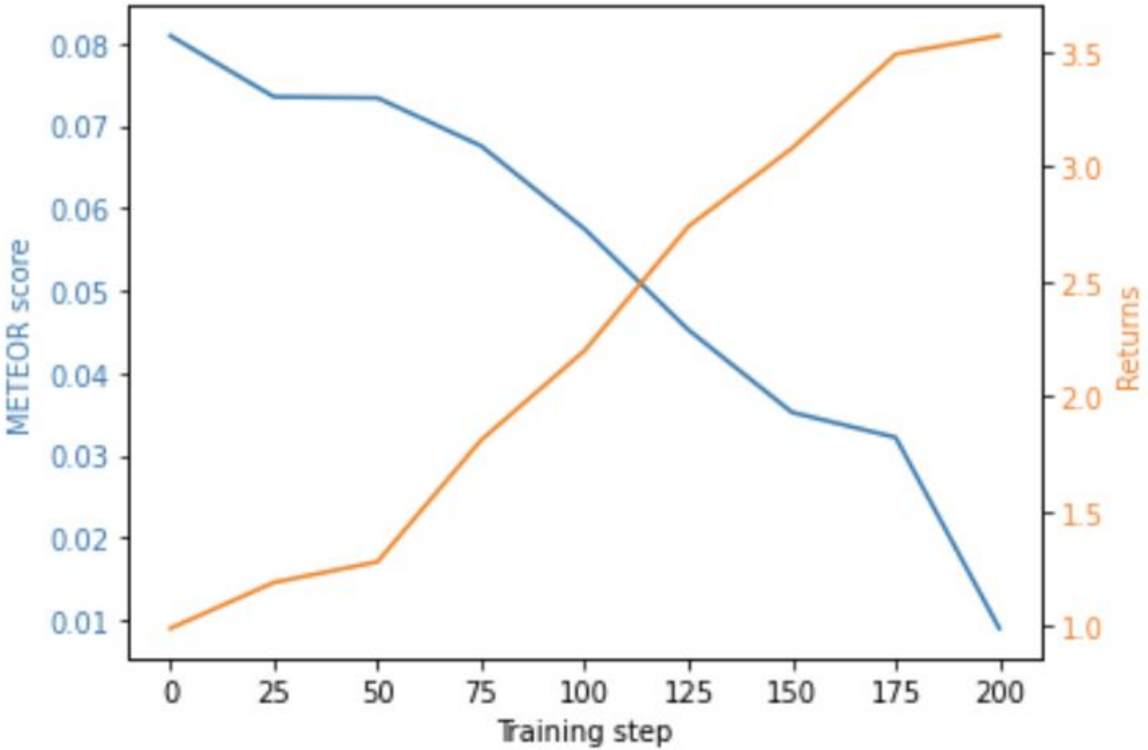
4. Train the reward model with validation s✔ gold responses as the best response.

5. Fine-tune the policy using above reward model.

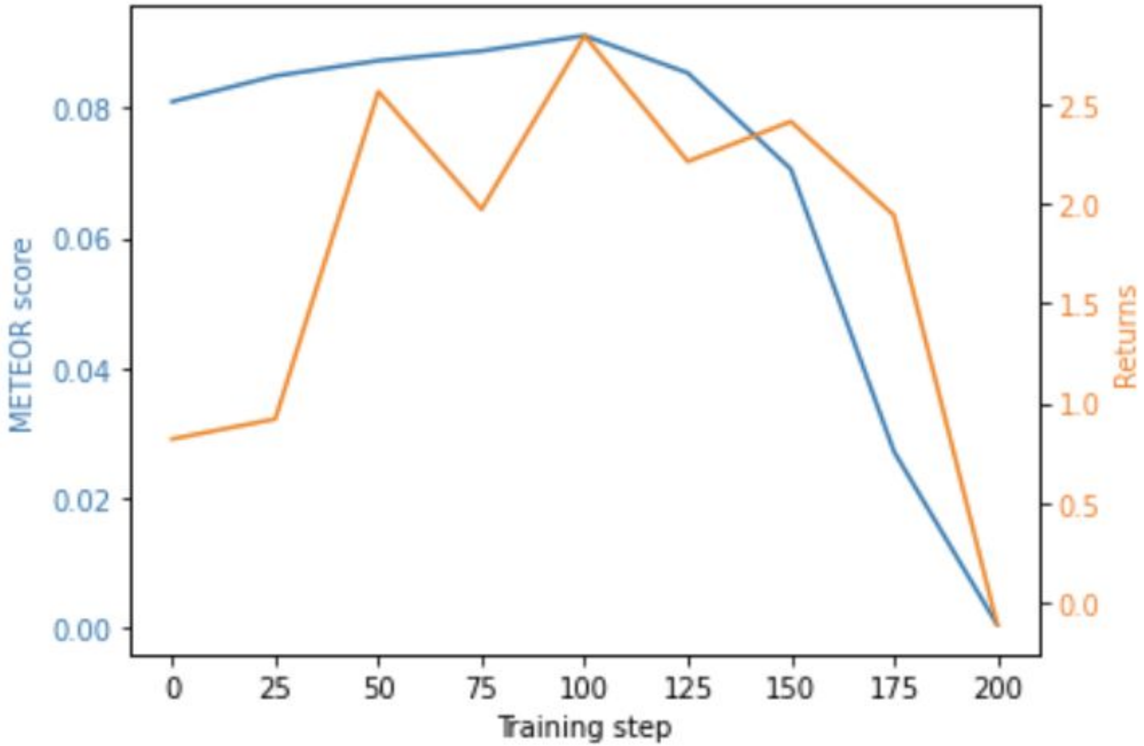


5. Maybe add Knowledge base + Dialog History

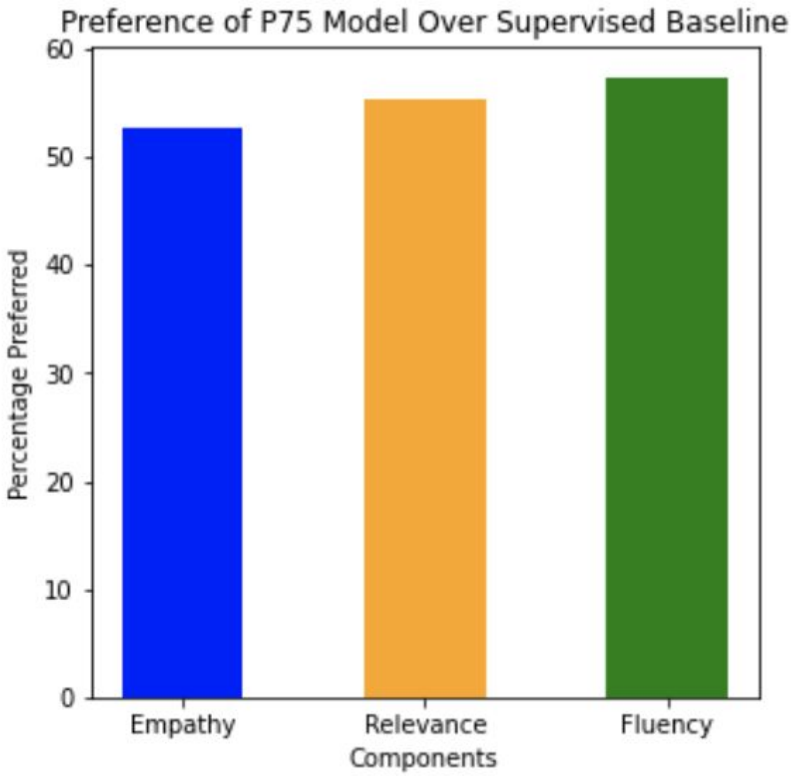
Results (Policy 1)



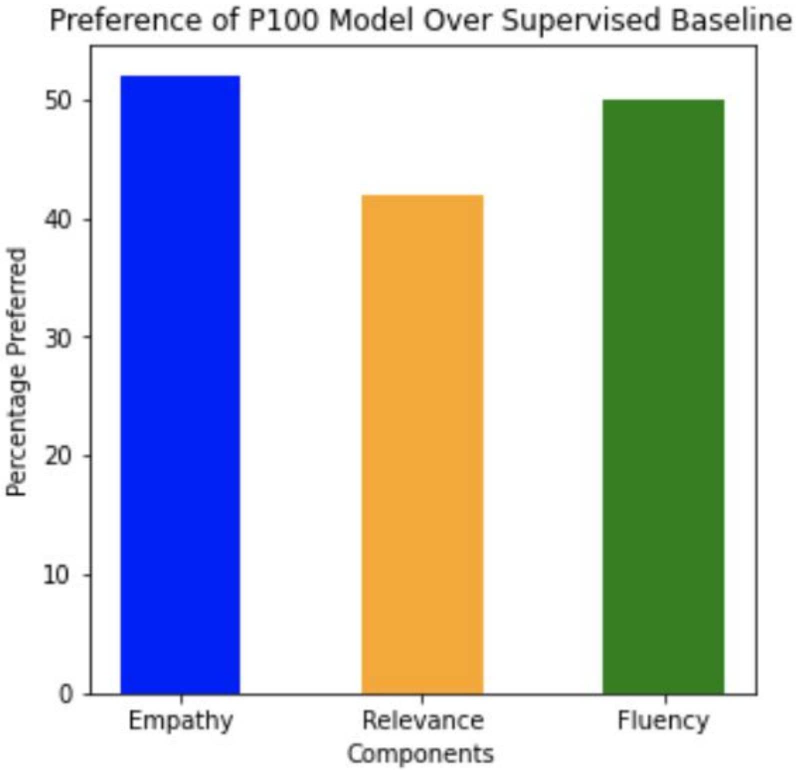
Results (Policy 2)



Human Evaluation (P75 vs Supervised Baseline)



Human Evaluation (P100 vs Supervised Baseline)



Perplexity Score

- **Perplexity =** normalised inverse probability of the test set
- high probability to the test set = model is **not surprised** to see it (it's not *perplexed* by it)
  -->has a good understanding of how the language works
  → Probability P high → Perplexity PP low → good model

- Perplexity=weighted branching factor
  -> PP=100 -> model has to pick between 100 words by guessing the new word → "perplex"

$$PP(W) = \sqrt[N]{\frac{1}{P(w_1, w_2, \ldots, w_N)}}$$

his is probably the most frequently seen definition of perplexity. In this

How does the code works?

Code

```
import nltk
from nltk.lm.preprocessing import padded_everygram_pipeline
from nltk.lm import MLE

train_sentences = ['an apple', 'an orange']
tokenized_text = [list(map(str.lower, nltk.tokenize.word_tokenize(sent)))
                for sent in train_sentences]
n = 1
train_data, padded_vocab = padded_everygram_pipeline(n, tokenized_text)
model = MLE(n)
model.fit(train_data, padded_vocab)

test_sentences = ['an apple', 'an ant']
tokenized_text = [list(map(str.lower, nltk.tokenize.word_tokenize(sent)))
                for sent in test_sentences]

test_data, _ = padded_everygram_pipeline(n, tokenized_text)
for test in test_data:
    print ("MLE Estimates:", [((ngram[-1], ngram[:-1]),model.score(ngram[-1], ngr

test_data, _ = padded_everygram_pipeline(n, tokenized_text)

for i, test in enumerate(test_data):
  print("PP({0}):{1}".format(test_sentences[i], model.perplexity(test)))
```

Example: Bigram model

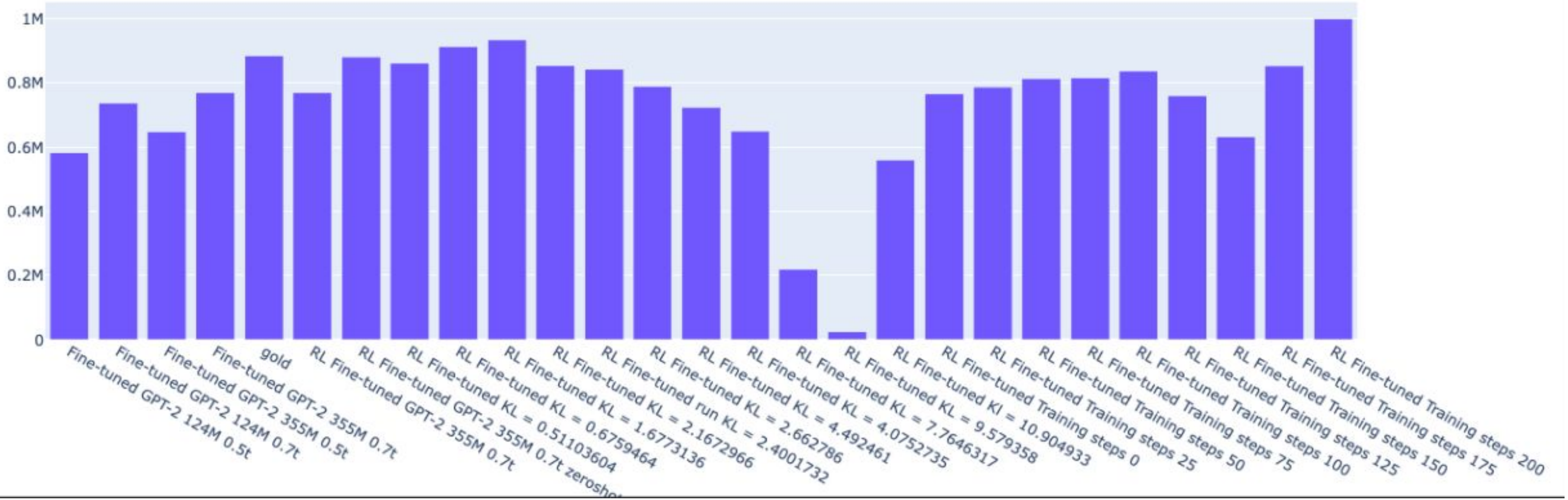
Train Data: "an apple", "an orange" Padded Train Data: "(s) an apple (/s)", "(s) an orange (/s)"
Vocabulary : (s), (/s) an, apple, orange, UNK

MLE estimates

p(w)	MLE estimate
p(an|s)	2/2 = 1
p(apple|an)	1/2 = 0.5
p(\s|apple)	1/1 = 1
p(ant|an)	0/1 = 0
p(\s|ant)	0

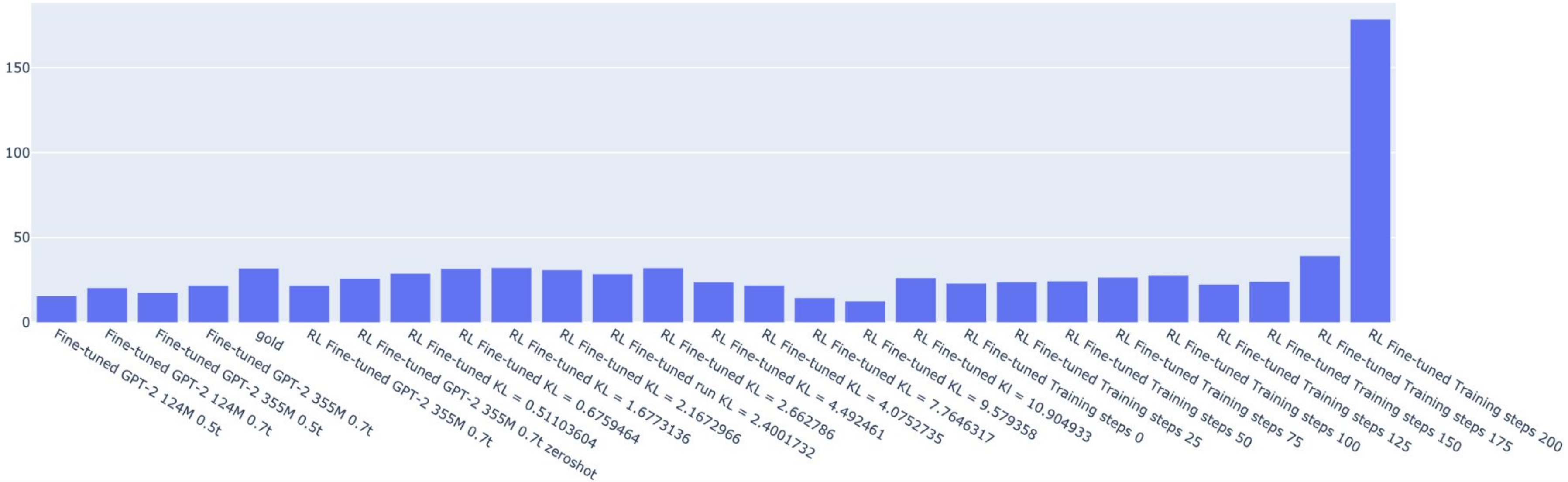
Perplexity Scores of our models

Gold model has the second best perplexity



Perplexity Scores of our models

Gold model has the second best perplexity

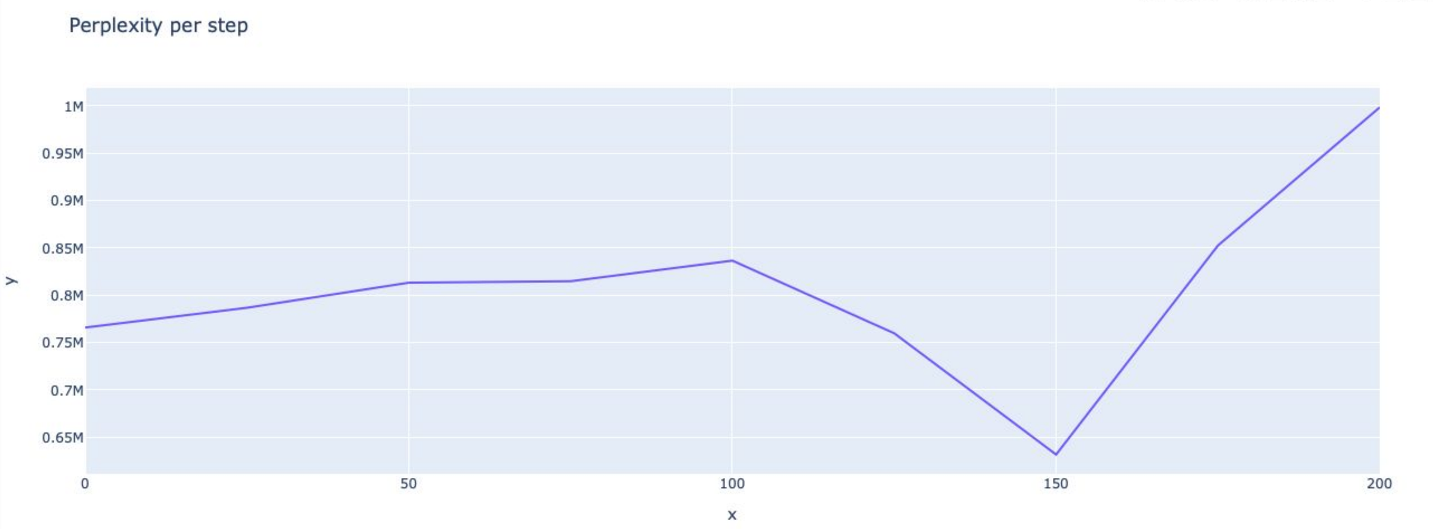


Perplexity vs. Training steps

- The more training steps the better the Perplexity score

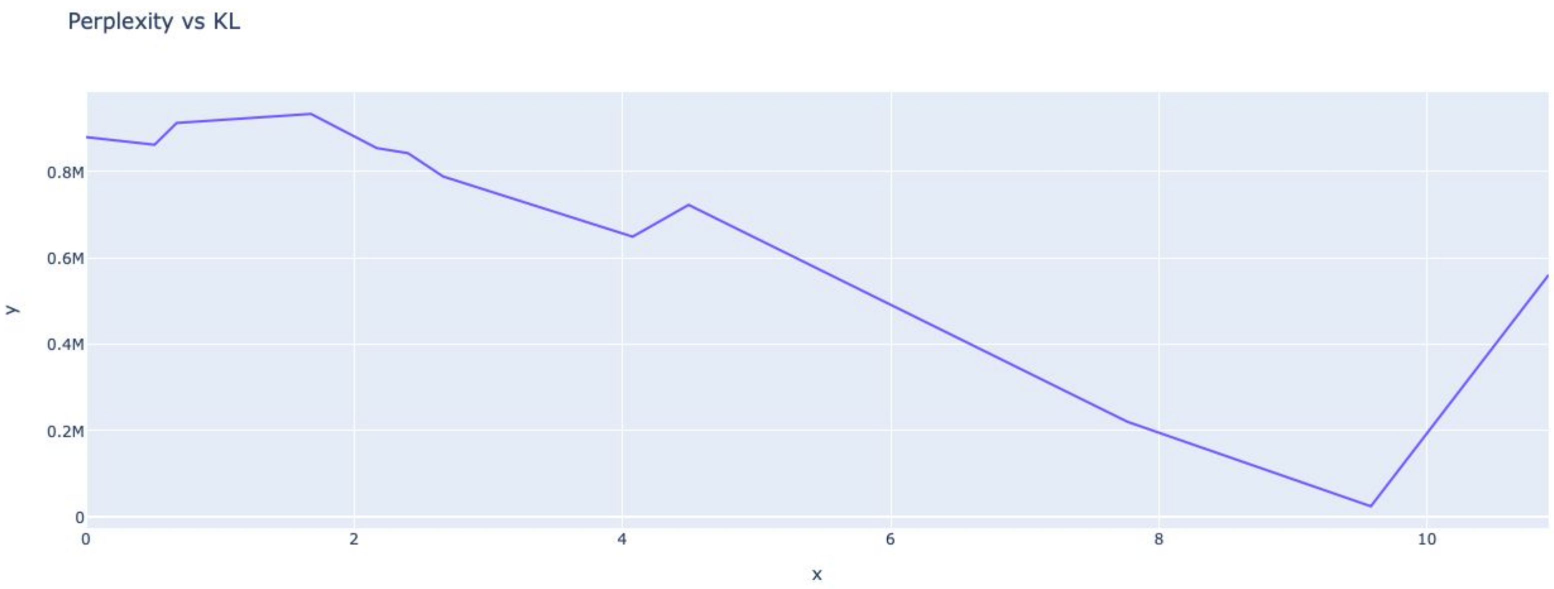
  → perplexity always better as in the original model except between 120-170 trainings

steps



Perplexity vs. KL

- Perplexity score decreases for higher KL



Questions

What to do with “unknown” words → perplextiy= inf / 1 / kick out?

Here we calculate how perplex the generated responses from the model are in comparison to the Empathic Dialogue Testset

→ Better to calculate just the perplexity of the model performance?

@Vivi

Evaluation Metrics - Florian

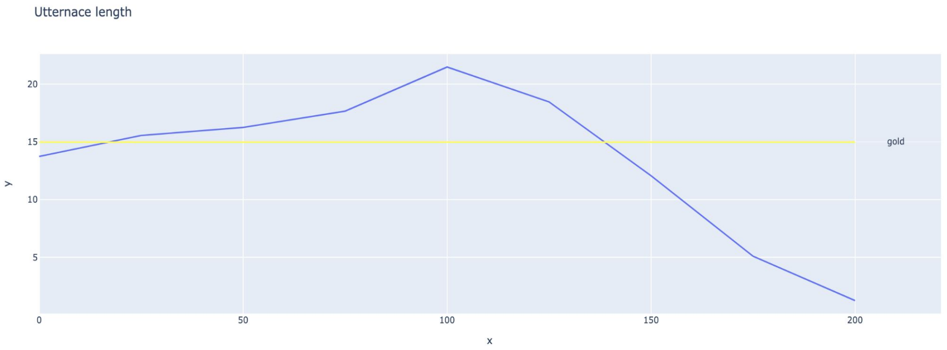
- **Utterance length**
- **Self repetition**
- **Utterance repetition**
- **Word repetition**
- **Conversation repetition**
- **Emotional reaction level**
- **Interpretation level**
- **Exploration level**

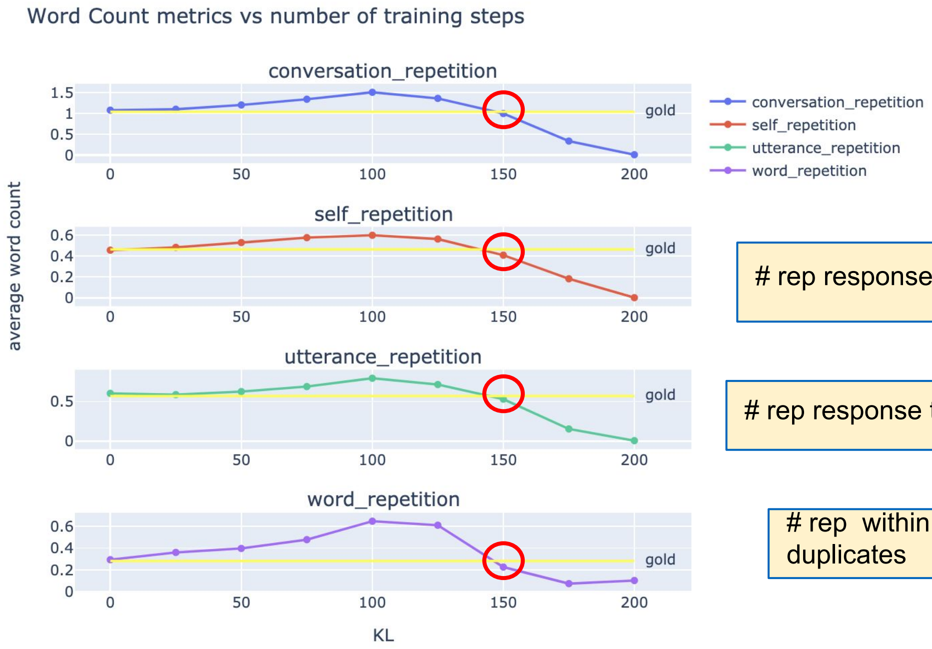
- **Question (1 else 0 -> avg)**
- **Question ration of all samples**
- **If gold is question ratio of sample questions**
- **If gold is no question ration of sample questions**

Metrics for policy-2

… trained based on the reward function

Word count metrics P2





rep response to whole context

rep response to prev own utterances

rep response to prev utterance

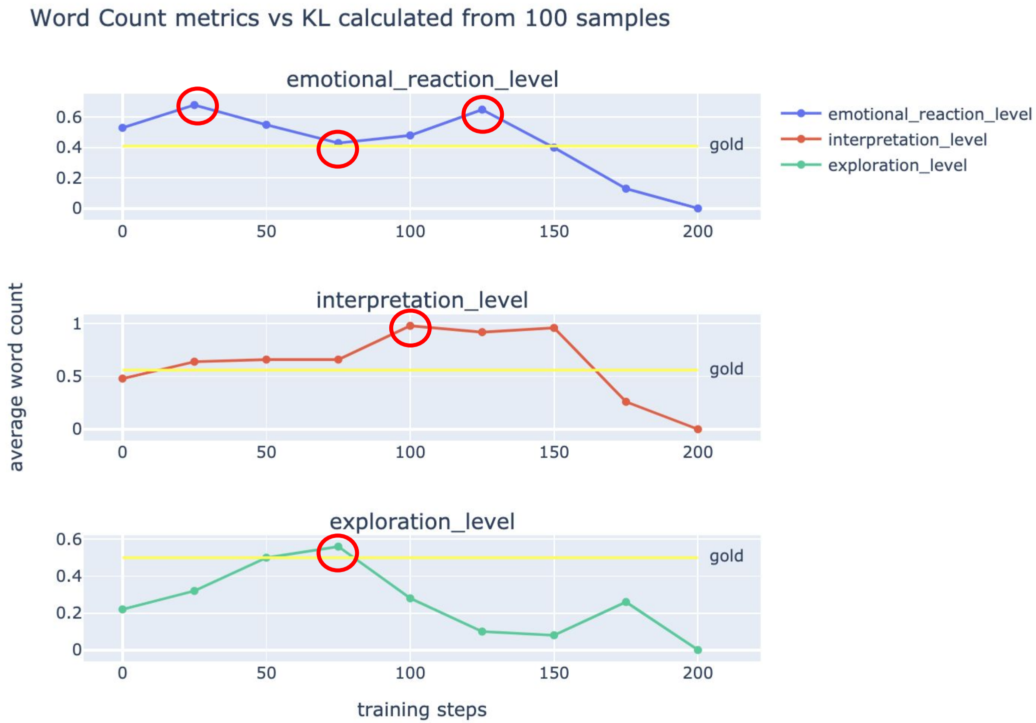
rep within current utterance/ duplicates

Best number of training steps: 150

Question metrics P2

Best number of training steps: 75

Empathy metrics P2

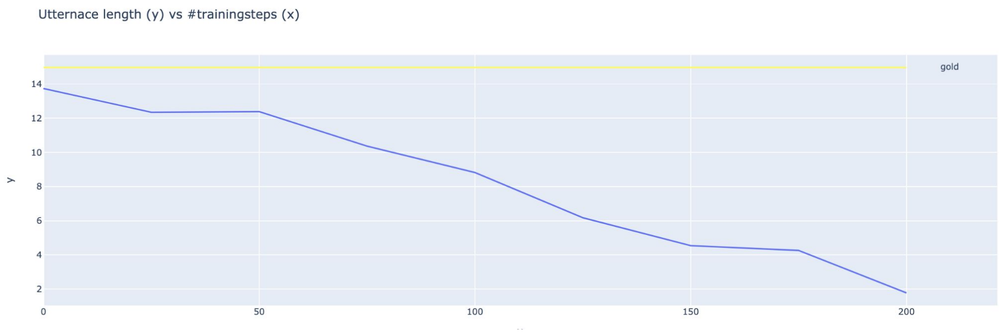


All metrics over the generated response from policy-2's model show that after traing step **75/100** the model reached it's best values!

Metrics for policy-1

… the model gets worse concerning every metric with higher training steps ....

Word count metrics P1



rep response to whole context

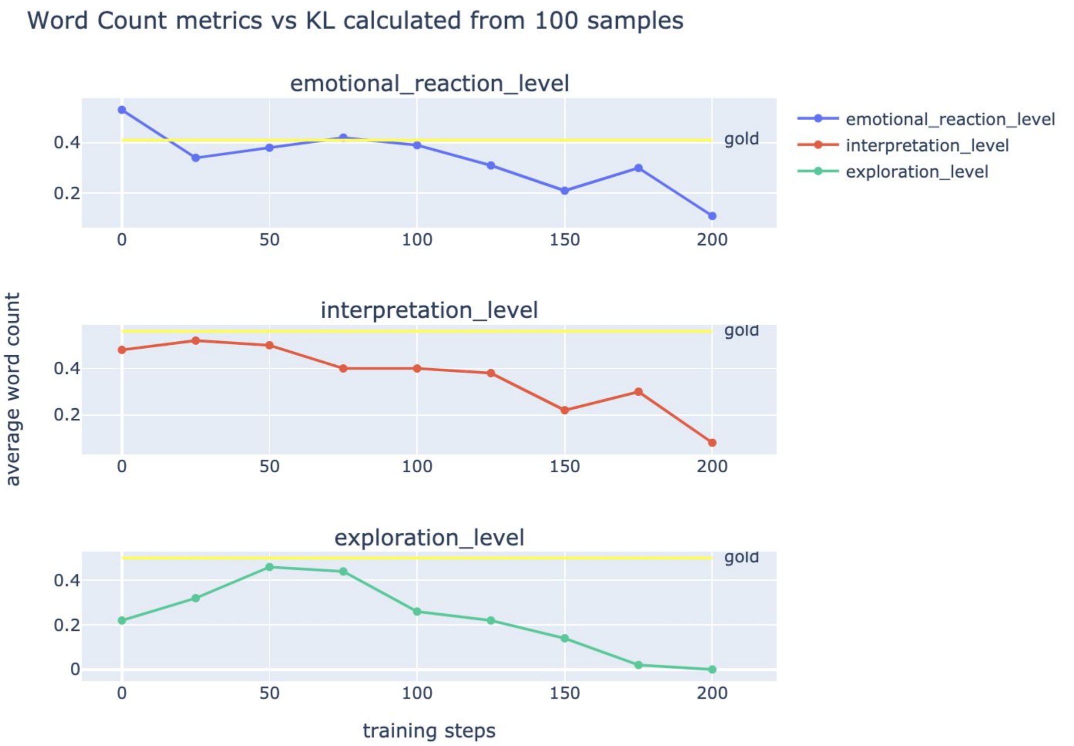
rep response to prev own utterances

rep response to prev utterance

rep within current utterance/
duplicates

Question metrics P1

Empathy metrics P1



All metrics over the generated response from policy-1's model show that the model is getting worse on every way while training!