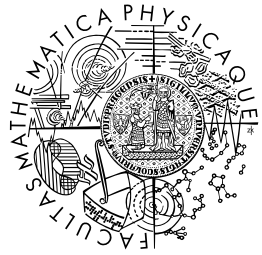


FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Bc. Matěj Kripner

Self-Supervised Summarization via Reinforcement Learning

Institute of Formal and Applied Linguistics

Supervisor of the master thesis: doc. RNDr. Ondřej Bojar, Ph.D.
Consultant of the master thesis: Mgr. Aleš Tamchyna, Ph.D.
Study programme: Artificial Intelligence
Study branch: Intelligent agents

Prague 2024

I declare that I carried out this master thesis on my own, and only with the cited sources, literature and other professional sources. I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ....

Author's signature

I'd like to express my gratitude to my supervisor Ondřej Bojar and my consultant Aleš Tamchyna for the many hours of consultations that provided me with invaluable feedback and often nudged me in the right direction. I'd also like to thank Milan Straka for his suggestions early on in the work.

Acknowledgement

This work was supported by the Ministry of Education, Youth and Sports of the Czech Republic through the e-INFRA CZ (ID:90254).

Title: Self-Supervised Summarization via Reinforcement Learning

Author: Bc. Matěj Kripner

Institute: Institute of Formal and Applied Linguistics

Supervisor: doc. RNDr. Ondřej Bojar, Ph.D., Institute of Formal and Applied Linguistics

Consultant: Mgr. Aleš Tamchyna, Ph.D.

Abstract: In deep learning, summarization models are traditionally trained using a maximum likelihood objective with reference summaries. Another line of work explores self-supervised approaches that do not require and are not limited by references. In this thesis, we opt for the latter approach. Our main contributions include the design of a novel dense reward function for summarization and its application for fine-tuning a sequence-to-sequence model via reinforcement learning. We build the whole training pipeline in a modular fashion, separately evaluating and tuning a supervised pre-training module, the reinforcement learning algorithm, and the reward function. After connecting all these components together, we also tune our self-learning approach as a whole. We evaluate the final checkpoints using 12 automatic and 3 manual metrics, revealing an improvement in reference-free metrics in nearly all cases.

Keywords: summarization, reinforcement learning, language model, self-supervision

Název práce: Automatická sumarizace z neanotovaných dat pomocí zpětnovazebního učení

Autor: Bc. Matěj Kripner

Ústav: Ústav formální a aplikované lingvistiky

Vedoucí diplomové práce: doc. RNDr. Ondřej Bojar, Ph.D., Ústav formální a aplikované lingvistiky

Konzultant: Mgr. Aleš Tamchyna, Ph.D.

Abstrakt: Sumarizační modely v kontextu hlubokého učení jsou tradičně trénovány metodou maximální věrohodnosti s použitím referenčních souhrnů. Aktivní je také výzkum v oblasti učení s vlastním dohledem (self-supervised), kde reference nejsou vyžadovány a výsledné modely jimi nejsou limitovány. Na tento výzkum navazujeme v této práci návrhem nové funkce odměn (reward function), která hodnotí kvalitu jednotlivých tokenů souhrnu. Tuto funkci pak aplikujeme ve zpětnovazebním učení. Celou trénovací logiku implementujeme modulárně, kdy nezávisle na sobě vyhodnocujeme a ladíme modul učení s učitelem, algoritmus zpětnovazebního učení a funkci odměn. Stejně tak ladíme i výsledný program po propojení těchto komponent. Výsledné modely vyhodnocujeme na 12 automatických a 3 manuálních metrikách. V téměř všech případech náš přístup zlepšil skóre na metrikách nevyžadujících referenční souhrn (reference-free).

Klíčová slova: sumarizace, zpětnovazební učení, jazykový model, učení s vlastním dohledem

Contents

Introduction

Automatic text summarization is a well-established task in NLP with the first extractive approaches dating as far back as the 1950s [Luhn, 1958]. With the advent of deep learning and the Transformer architecture [Vaswani et al., 2017], summarization has seen dramatic improvements using abstractive methods that capture the crux of a text without being limited to the original phrasing. Recently, large language models trained using unsupervised objectives on internet-scale datasets exhibited summarization as an emergent capability, surpassing previous results with just one-shot prompting [Radford et al., 2019, Brown et al., 2020, Goyal et al., 2022]. These capabilities were then improved by aligning the models to human preferences using reinforcement learning from human feedback (RLHF) [Stiennon et al., 2020].

Furthermore, reinforcement learning has been used to directly optimize reference-free summarization metrics like SUPERT [Gao et al., 2020]. Instead of optimizing a scalar per-sequence reward, previous work has also explored redistributing the reward to individual summary tokens based on the attention map of Transformer architecture, densifying the training signal [Chan et al., 2024].

We build on this in optimizing summarization models for a novel per-token reward function that rewards relevant pieces of information and penalizes wrong, misleading, or irrelevant phrases. To this end we employ a sequence-to-sequence model (*predictor*) based on BART [Lewis et al., 2019], fine-tuned to predict a text based on its summary. The basic idea is that when some of the summary tokens are masked, the increase in predictor's perplexity gives us a signal about the importance of these tokens.

Our main contributions are the following:

- We introduce a new approach for evaluating the quality of individual summary tokens, leading to a dense reward function for summarization.
- We provide a family of such reward functions by implementing various architectural and parameter options.
- We design proxy metrics for evaluating the quality of such reward functions and conduct extensive experiments including a grid search.
- We provide a simple implementation of the REINFORCE algorithm [Williams, 1987, 1992] and test it on an artificial task of maximizing the frequency of output words starting with the letter `T`.
- We apply REINFORCE using our reward function to train BART-base and BART-large models on XSum and CNN/Daily Mail datasets (with labels removed), experimenting with different hyperparameter configurations. The models are first pre-trained in a supervised fashion on a different part of the datasets.
- We use the SummEval [Fabbri et al., 2020] package to evaluate the resulting models on 12 summarization metrics. Our approach improves performance on all reference-free metrics.
- We conduct a small human evaluation using 3 manual metrics. Our approach improves performance on all these metrics, but the sample size is too small to make a definitive conclusion.

- We create pull requests to these open-source repositories: SummEval (fixes in the installation process), RL4MLs (migration to new versions of dependencies), Transformers (a simple bug fix).

The whole codebase is implemented using PyTorch 2 [Ansel et al., 2024], mostly in combination with the Transformers library [Wolf et al., 2020].

The thesis is structured as follows. In the first chapter, we give an overview of related work in summarization and reinforcement learning. In the second chapter, we describe the datasets and family of models used in all experiments. The third chapter deals with supervised fine-tuning of these models to provide a baseline and a starting point for the reinforcement learning experiments. In the fourth chapter, we discuss our implementation of the REINFORCE algorithm and test it on an artificial task. In the fifth chapter, we introduce a new reward function for summarization and experiment with various design choices. The final chapter connects all the previous work by improving the previously fine-tuned models using reinforcement learning with our reward function and evaluating the end result using the SummEval package and human judgement.

Thanks to this modular design, the individual parts (supervised fine-tuning, reinforcement learning, reward function) can be exchanged for a different implementation or used independently.

1. Background

In this chapter we give a brief overview of summarization, reinforcement learning, and relevant previous results.

1.1 Summarization

Automatic summarization in the context of natural language processing is the process of shortening an input text so that the result represents some of the relevant information within the input [Lloret and Palomar, 2012]. What is meant by *relevant* is influenced by the type of user reading the summaries and can be conditioned on a query (*query-based* as opposed to *generic*). The goal of the summary can be to contain as much information as possible or to only inform about the scope of the summarized document without giving any details (*informative* versus *indicative*). The summarization system can work by extracting parts of the input text (e.g. sentences) or by writing the summary from scratch (*extractive* versus *abstractive*). Based on the type of input, summarization is either *single-document* or *multi-document*.

We focus on generic abstractive single-document summarization.

1.1.1 ROUGE

Summarization systems are traditionally evaluated using ROUGE [Lin, 2004], Recall-Oriented Understudy for Gisting Evaluation. ROUGE-n is defined as the n-gram recall between a generated summary and a reference summary, i.e. the ratio of reference n-grams that are also generated. ROUGE-L is instead based on the length of the longest common subsequence between generated and reference summaries.

For an overview of the limitations of ROUGE, refer to Schluter [2017], Ganesan [2018].

1.2 Reinforcement Learning

Reinforcement learning is a machine learning paradigm concerned with maximizing the cumulative reward of an agent in a dynamic environment through trial-and-error [Kaelbling et al., 1996, Sutton and Barto, 2018]. The agent executes actions using its policy based on the observed state of the environment. The key difference from supervised learning is that the agent does not receive feedback about which action it should have taken, but rather just about the quality of the taken actions. Moreover, the feedback might be incomplete or delayed.

In the context of language models, actions can be individual output tokens or entire output sequences. Rewards are then calculated using automatic metrics [Gao et al., 2020], simulated environments [Carta et al., 2023], or human feedback [Ziegler et al., 2019]. Importantly, these rewards do not have to be differentiable.

In this context we refer to reward functions that assign a scalar value to each token as *dense*. This is in contrast to *sparse* reward functions which assign a single scalar to the entire output sequence.

1.2.1 REINFORCE

REINFORCE [Williams, 1987, 1992] is an algorithm for reinforcement learning which belongs to the class of direct policy search. As such, it searches for the agent's policy directly, instead of, for example, estimating the state value function and representing the policy implicitly. More specifically, it is an instance of the policy gradient method which we now describe.

Policy gradient methods (also called gradient-based) parametrize the agent's policy π_θ with a parameter vector θ. Importantly, the probability of executing action a in state s, i.e. $\pi_\theta(a|s)$, has to be differentiable with respect to θ. In a supervised setting, the reference action a would be known, and we could directly use the gradient $\nabla_\theta \pi_\theta(a|s)$ in gradient descent.

In reinforcement learning, we seek to maximize some performance measure $J(\theta)$ which in finite episodic tasks is taken to be the value of the initial state, i.e. $v_{\pi_\theta}(s_0)$. We use the *policy gradient theorem* to calculate how the policy should be altered to increase $J(\theta)$. The theorem states that:

$$\nabla_\theta J(\theta) \propto \sum_s \mu(s) \sum_a q_{\pi_\theta}(s,a) \nabla_\theta \pi_\theta(a|s), \tag{1.1}$$

where $\mu(s)$ is the on-policy distribution under π_θ, representing how often a state s is encountered, i.e.:

$$\mu(s) = \sum_{k=0}^{\infty} \gamma^k P(s_0 \to s \text{ in } k \text{ steps} \mid \pi_\theta), \tag{1.2}$$

and $q_{\pi_\theta}(s,a)$ is the state-action value, i.e.:

$$q_{\pi_\theta}(s_t, a) = \mathbb{E}_{s_{t+1}, r_{t+1}} \left[r_{t+1} + \gamma v_{\pi_\theta}(s_{t+1}) \right]. \tag{1.3}$$

The right-hand-side of Equation (1.1) can then be used as the gradient in a gradient descent update, since the effect of any proportionality constant is negated by the arbitrary choice of learning rate. The REINFORCE algorithm provides a way to use states and actions sampled from the agent's interaction with environment to obtain an expression whose expected value is equal to this gradient. We start with the policy gradient theorem and replace the sum over all states with an expectation, utilizing the definition of on-policy distribution:

$$\nabla_\theta J(\theta) \propto \mathbb{E}_{s\sim\mu} \left[\sum_a q_{\pi_\theta}(s,a) \nabla_\theta \pi_\theta(a|s) \right]. \tag{1.4}$$

Then we multiply the expression by $1 = \frac{\pi_\theta(a|s)}{\pi_\theta(a|s)}$ to enable once again replacing the sum with expectation:

$$\begin{aligned} \nabla_\theta J(\theta) &\propto \mathbb{E}_{s\sim\mu} \left[\sum_a \pi_\theta(a|s) q_{\pi_\theta}(s,a) \frac{\nabla_\theta \pi_\theta(a|s)}{\pi_\theta(a|s)} \right] \\ &= \mathbb{E}_{s\sim\mu, a\sim\pi_\theta} \left[q_{\pi_\theta}(s,a) \frac{\nabla_\theta \pi_\theta(a|s)}{\pi_\theta(a|s)} \right]. \end{aligned} \tag{1.5}$$

Finally, we estimate $q_{\pi_\theta}(s,a)$ with the sample return $G_t = \sum_{t'=t}^{\infty} \gamma^{t'-t} r_{t'+1}$, which is an unbiased estimator. This yields the final REINFORCE update:

$$\theta_{t+1} = \theta_t + \alpha G \frac{\nabla_\theta \pi_\theta(a|s)}{\pi_\theta(a|s)}. \tag{1.6}$$

Intuitively, this update attempts to increase the probability of the taken action proportionally to its observed performance. It uses the $\pi_\theta(a|s)^{-1}$ factor to down-weight actions which are sampled more often and therefore will be updated more frequently.

We can use the identity $\nabla \ln(x) = \frac{\nabla x}{x}$ to rewrite the update to the following form:

$$\theta_{t+1} = \theta_t + \alpha G \nabla_\theta \ln \pi_\theta(a|s). \tag{1.7}$$

Note that this is just a weighted cross-entropy loss with actions used as reference labels and returns used as weights.

1.3 BART Architecture

BART [Lewis et al., 2019] is a family of encoder-decoder sequence-to-sequence models trained using the objective of de-noising corrupted text. It closely follows the Transformer architecture from Vaswani et al. [2017] with modifications only to the activation functions (using GELUs [Hendrycks and Gimpel, 2016]) and to parameter initialization (using $\mathcal{N}(0, 0.02)$). The base model contains 6 layers in both encoder and decoder and uses a hidden size of 768. The large model scales this to 12 layers and a hidden size of 1024.

BART utilizes a bidirectional encoder and an autoregressive decoder and can thus be seen as a generalization of BERT [Devlin et al., 2018] and GPT [Radford et al., 2018]. This is in contrast to encoder-only models which are not suitable for autoregressive generation and decoder-only models in which for a given token, the attention layers can only access tokens that are positioned before it in the sequence.

1.4 Related Work

The idea of improving summarization capabilities using reinforcement learning is not new. In this section we discuss some of the relevant previous results.

RLHF

Most notably, previous work has utilized the framework of Reinforcement Learning from Human Feedback (RLHF) [Akrour et al., 2011, Christiano et al., 2017] to bypass proxy objectives and directly optimize for human feedback about the relative quality of summaries [Ziegler et al., 2019, Stiennon et al., 2020, Ouyang et al., 2022, OpenAI, 2022b]. This makes it easier to utilize the capabilities of a model and also improves summarization performance.

Reinforcement Using Automatic Metrics

Paulus et al. [2017] use ROUGE-L as the reward function, improving performance of their attention-based model on the CNN/Daily Mail [Hermann et al., 2015] dataset and achieving state-of-the-art at the time. To achieve improvements both on ROUGE and on human evaluation, they optimize a convex combination of the reinforcement signal

(i.e. policy learning) and the standard maximum-likelihood loss for next-token prediction. The reinforcement learning signal is based on REINFORCE but uses the self-critical approach of Rennie et al. [2017] for baseline estimation. In this approach, the baseline is calculated as the reward of a summary generated by greedy search (as opposed to sampling).

Pasunuru and Bansal [2018] build on this, addressing the issue that ROUGE-L does not reflect important properties of a summary such as salient phrase inclusion and directed logical entailment. To this end they propose two new reward functions – *ROUGESal* which rewards salient phrases by giving them more weight during the ROUGE calculation, and *Entail* which assesses if the summary is logically entailed by the source document. They reach the best results when optimizing both metrics concurrently, alternating between them in successive batches. They present this approach as simpler than optimizing a linear combination of metrics.

Gao et al. [2020] use their reference-free SUPERT metric as a reward function to achieve state-of-the-art at the time results on multiple ROUGE metrics.

Metrics Based on Language Understanding

Arumae and Liu [2019] introduce a reward for extractive summarization based on the helpfulness of a summary in answering Cloze-style [Taylor, 1953] questions about the source document. The question-answer pairs are obtained from reference summaries by masking either a salient word or a named entity from each sentence. The final reward is then calculated as a linear combination of 4 metrics: *QA competency* based on question answering, *adequacy* based on the unigram overlap with the reference summary, *fluency*, and *length*.

Eyal et al. [2019] propose *APES*, Answering Performance for Evaluation of Summaries, measuring quality of a summary based on how helpful it is in answering questions about the original document. The questions are generated either manually or automatically from a reference summary by masking entities identified using a named-entity recognition system. Scialom et al. [2019] extend this to the unsupervised setting where the questions and answers are generated from the source document instead of the reference summary. The resulting reference-free metric is referred to as SummaQA in the SummEval package [Fabbri et al., 2020].

Question answering has also been utilized for summarization evaluation by Chen et al. [2018].

Vasilyev et al. [2020] introduce the BLANC metric based on the ability of BERT [Devlin et al., 2018] to fill in words masked in an article sentence while having access to the summary. Most similar to our approach, their BLANC-help variant uses BERT twice: First, the input is a summary concatenated with a masked sentence from the original article. Second, the summary is replaced with a "filler" where each token is substituted with a period (".") . The BLANC metric is then calculated using the relative success of predicting the masked words when the summary is available versus when it is not. One mask covers every sixth word and in total 6 masks are generated to cover each word once. Their second variant (BART-tune) instead fine-tunes BERT on a small dataset created from the summary and observes how that improves prediction of the article.

Dense Rewards

Another line of work addresses the challenge arising from the sparsity of the reward signal which typically gives only a scalar value for the entire summary. This sparsity leads to less efficient learning [Andrychowicz et al., 2017].

In the case of human feedback, dense token-level or segment-level annotations [Lightman et al., 2023, Wu et al., 2024] are expensive to produce. Bai et al. [2022b] and Lee et al. [2023] explore complementing or substituting the human feedback with AI feedback from a preference model, yielding Reinforcement Learning from AI Feedback (RLAIF). Similarly, Cao et al. [2024] use a critic model prompted to identify problematic spans in a summary, resulting in intrinsic span-level rewards which complement the extrinsic summary-level reward. The critic model is either GPT-3.5 [OpenAI, 2022a] or the same as the policy model (Llama 2 [Touvron et al., 2023]).

Chan et al. [2024] instead introduce Attention Based Credit which redistribute the scalar extrinsic reward to individual token based on the attention map in the reward model. Specifically, they take the last layer and average the attentions over all heads.

2. Datasets and Models

Here we describe the datasets and family of models used in all our experiments. We also discuss the issue of truncating long inputs and outputs to improve efficiency and comply with architectural restrictions of the models.

2.1 Datasets

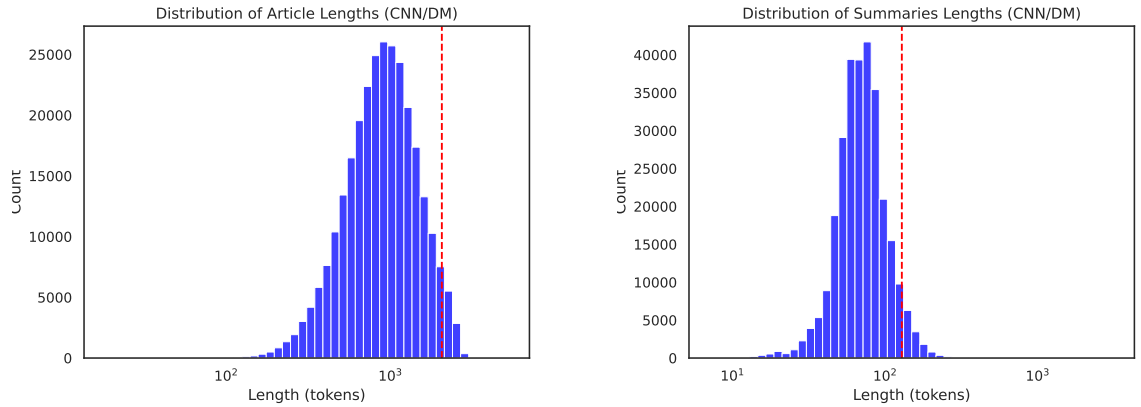
We apply our approach to CNN/Daily Mail and XSum datasets following Lewis et al. [2019].

The CNN/Daily Mail summarization dataset [Hermann et al., 2015, Nallapati et al., 2016] as used in See et al. [2017] is more extractive, meaning the target summaries tend to resemble source sentences. Each data point consists of a news article and its summary, harvested from CNN and Daily Mail. The mean article length is 983 tokens and the mean summary length is 75 tokens. The dataset is split into train (287k examples), validation (13.4k examples) and test (11.5k examples) sets.

In contrast, the XSum summarization dataset [Narayan et al., 2018] is highly abstractive with one-sentence summaries. Each data point is a news article and its summary, harvested from the BBC. The mean article length is 524 tokens and the mean summary length is 30 tokens. The dataset is split into train (204k examples), validation (11.3k examples) and test (11.3k examples) sets.

Because of the quadratic complexity of self-attention with respect to sequence length, we seek to limit the length of training samples. To that end, we truncate each news article to a maximum length of 2048 tokens for CNN/Daily Mail and 1024 for XSum. The number 1024 for XSum is chosen so that we can test models with absolute position embeddings of length 1024, such as BART, without any modification. The distribution of article and summary lengths in train split together with the cutoff line are shown in Figure 2.1 for CNN/Daily Mail, and Figure 2.2 for XSum.

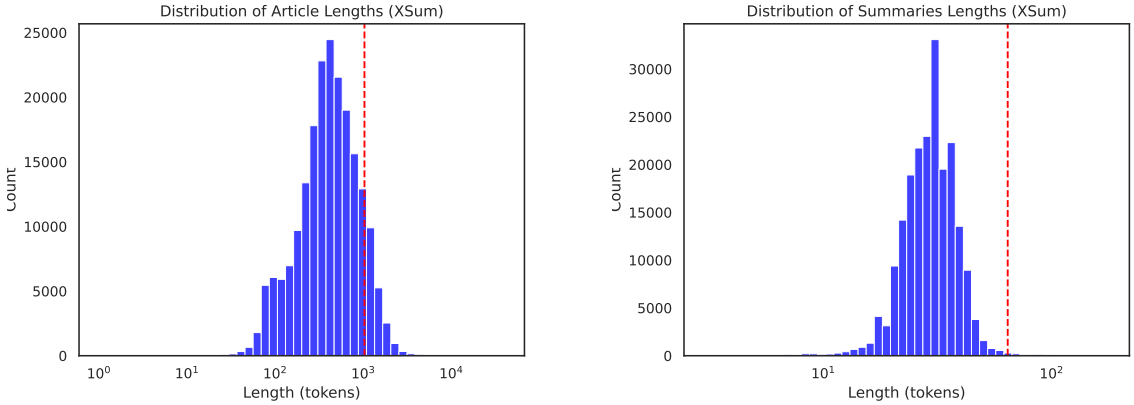
In the train splits, 4.8 % of articles and 5.1 % of summaries are affected in CNN/Daily Mail, and 10.9 % of articles and 0.4 % of summaries are affected in XSum.



(a) Article lengths.

(b) Summary lengths.

Figure 2.1: Token lengths in train split of CNN/Daily Mail with cutoff marked.



(a) Article lengths.

(b) Summary lengths.

Figure 2.2: Token lengths in train split of XSum with cutoff marked.

2.2 Model

We test our approach on BART [Lewis et al., 2019], a widely used family of models which we described in Section 1.3. Concrete pre-trained models are BART-base with 139M parameters and BART-large with 406M parameters, both available at HuggingFace.[1]

Initially, we experimented with the T5 model [Raffel et al., 2020] and we expect our findings to carry over to other similar models, however, we ultimately decided to utilize BART because of its recent successes in summarization [Ghosal et al., 2023].

[1] `https://huggingface.co/facebook/bart-large`

3. Supervised Fine-Tuning

In this chapter, we obtain a baseline as well as a starting checkpoint for reinforcement learning experiments through supervised fine-tuning of pre-trained models. Specifically, we fine-tune BART-base and BART-large models (Section 2.2) for summarization on XSum and CNN/Daily Mail datasets (Section 2.1) using the standard next-token prediction objective. In the case of CNN/Daily Mail dataset, we extend the absolute position embedding size of BART from 1024 to 2048.

The self-supervised training approach described in later chapters can be used with any unlabeled documents, not necessarily related to the dataset used for supervised fine-tuning. However, we use an easier setting where we split the train set of XSum and CNN/Daily Mail in half and use the first half for supervised fine-tuning and the second half for reinforcement learning (where the reference summaries are hidden from the model). With this, we minimize the distribution shift between data in both training stages. This leaves to future work the exploration of using a different data source for the reinforcement learning stage.

3.1 Fine-Tuning

We base our choice of hyperparameters on the official fine-tuning tutorial in Fairseq[1] and tune them with manual experimentation on XSum. The resulting important hyperparameter values are listed in Table 3.1.

We use Adam optimizer [Kingma and Ba, 2014] from the bitsandbytes library with 8-bit quantization of the optimizer state. We also experimented with Adafactor [Shazeer and Stern, 2018] which did not result in performance improvement. The L2 weight decay is applied on all parameters except bias.

When allowed by the GPU architecture, we use 16-bit mixed precision training and PyTorch 2.0's training graph compilation. To fully utilize NVIDIA Tensor Cores in mixed precision training, we follow the official NVIDIA guide[2] and pad all sequence lengths to multiples of 8 along with choosing batch size to be a multiple of 8. We also experimented with gradient checkpointing [Chen et al., 2016] but found that gradient accumulation with smaller batch sizes gives better performance in our case.

3.1.1 Context Length Extension

To enable training on the CNN/Daily Mail dataset, we modify the trained absolute position embedding matrix of BART from length 1024 to length 2048. The new embedding matrix is initialized with two concatenated copies of the original matrix. We also experimented with initializing the second half of the matrix randomly which yielded similar performance. Initializing the whole matrix randomly yielded slightly inferior performance.

[1] `https://github.com/facebookresearch/fairseq/blob/main/examples/bart/README.summarization.md`

[2] `https://docs.nvidia.com/deeplearning/performance/mixed-precision-training/index.html#tensor-core-shape`

Hyperparameter	BART-base	BART-large
Samples per update	64	
Learning rate	1e-4	3e-5
Learning rate schedule	linear decay	
Warmup steps	500	
Label Smoothing	0.1	
L2 weight decay	0.01	
Dropout rate	0.1	
Optimizer	Adam (8-bit quantized)	

Table 3.1: BART fine-tuning hyperparameters.

3.2 Evaluation

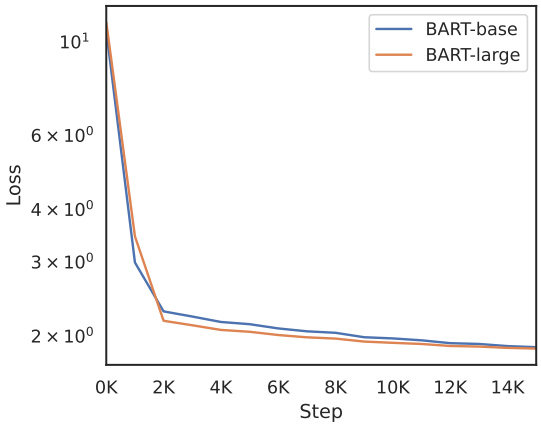
During training, we evaluate ROUGE-1, ROUGE-2, ROUGE-L, and ROUGE-LSum [Lin, 2004] on 10% of the validation split after every 1000 updates. The final evaluation is performed on the whole validation split.

For generation, we use beam search with 4 beams and early stopping. We also ban the repetition of any trigram and limit the maximum generation length to 160 for CNN/Daily Mail and 128 for XSum. Additionally, we use the Porter Stemming Algorithm [Porter, 1980] to strip word suffixes before evaluation. This is to address the fact that multiple word forms (due to tense, plurality, etc.) can refer to the same underlying concept.

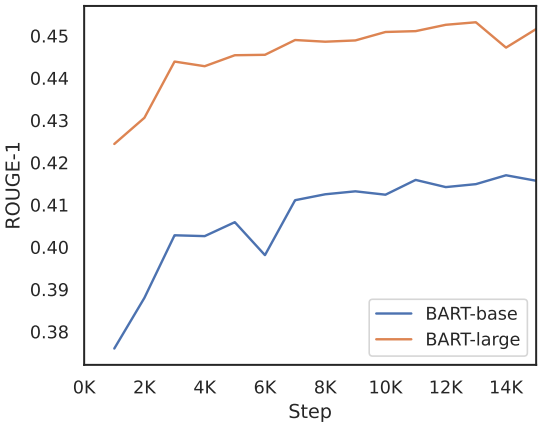
3.3 Results

On XSum, we performed a total of 15,000 updates (10 epochs). On CNN/Daily Mail, we performed 22,000 updates (10 epochs) in the case of BART-base and only 12,000 in the case of BART-large (for computation reasons).

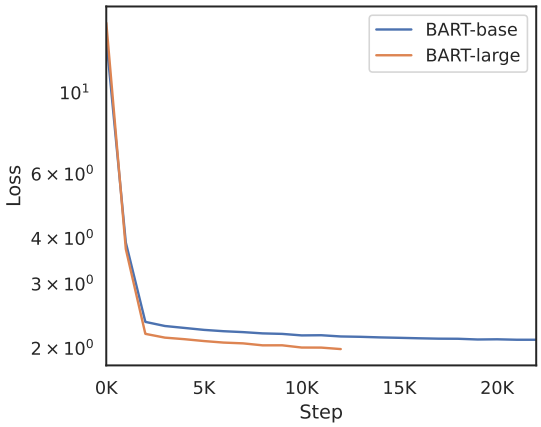
The training curves for BART on both XSum and CNN/Daily Mail are shown in Figure 3.1. Final evaluation scores are shown in Table 3.2 where models are listed by their HuggingFace identifier with our results prefixed by *"ours/"*. We achieve competitive performance to the official fine-tuned BART-large checkpoint released by Meta, which has seen twice the amount of data (this is because we split our train set into two parts as explained in the introduction to this chapter). Note that in the case of CNN/Daily Mail, our models have a context length of 2048 as opposed to the original 1024.



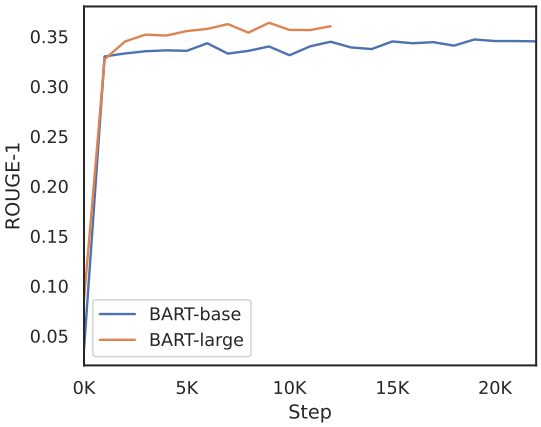
(a) Training Loss (XSum)



(b) Validation ROUGE-1 (XSum)



(c) Training Loss (CNN/DM)



(d) Validation ROUGE-1 (CNN/DM)

Figure 3.1: Supervised fine-tuning of BART.

Model	ROUGE-1	ROUGE-2	ROUGE-L
		XSum	
facebook/bart-large	17.2	3.2	11.0
facebook/bart-large-xsum	45.4	22.2	37.2
ours/bart-large-xsum *	44.7	21.4	36.6
ours/bart-base-xsum *	41.3	18.4	33.5
		CNN/Daily Mail	
facebook/bart-large †	36.5	16.2	22.6
facebook/bart-large-cnn †	44.8	21.7	31.2
ours/bart-large-cnn *	42.4	19.8	29.3
ours/bart-base-cnn *	41.8	19.2	29.2

* Trained on just 50% of data.

† Model is limited to context length 1024, increasing the effect of truncation.

Table 3.2: Comparison of performance on XSum.

4. Reinforcement Learning for Language Models

In this chapter we provide an overview of existing Reinforcement Learning (RL) libraries, develop our own implementation, and run experiments to decide which training configuration to use in the following chapters. We use an artificial test task for experimenting with reinforcement learning for language models – maximizing the average number of output words that start with the letter `T`.

4.1 Existing RL Libraries

First, we explore available RL libraries and evaluate their suitability for our task in the context of language models.

4.1.1 TRL: Transformer Reinforcement Learning

TRL [von Werra et al., 2020] is a mature library integrated with HuggingFace Transformers. It is mostly intended for Reinforcement Learning from Human Feedback (RLHF) and provides tools for Supervised Fine-tuning, Reward Modeling, and Proximal Policy Optimization (PPO) [Schulman et al., 2017]. It is attractive because of its large community and active development, with over 7.5K stars on GitHub as of February 2024. Unfortunately, it doesn't support assigning rewards to individual tokens (i.e. dense rewards) and instead only works with per-sequence scalar rewards. This makes it inapplicable for our purposes. When inspecting the codebase, we found no easy way to circumvent this limitation.

4.1.2 trlX

Inspired by TRL, trlX [Havrilla et al., 2023] was designed for RLHF at scale, providing Accelerate[1]-backed and NVIDIA NeMo[2]-backed trainers. It implements Proximal Policy Optimization (PPO) and Implicit Language Q-Learning (ILQL).

Based on the documentation and type signature of the main `train` method which serves as the library's entry point, trlX only supports per-sequence rewards similarly to TRL. We suspect that this limitation could be circumvented by directly using the `AcceleratePPOTrainer` class. Since this is not done in any of the provided examples, we did not pursue this direction for time reasons.

4.1.3 RL4LMs

RL4LMs [Ramamurthy et al., 2022] is a RL library based on Stable Baselines 3 (SB3) [Raffin et al., 2021], originally designed for RLHF. Unfortunately, RL4MLs is not actively developed, with no updates in nearly a year and outdated dependencies. We forked and updated the codebase, notably migrating to current versions of Stable Baselines 3 and

[1] `https://huggingface.co/docs/accelerate`
[2] `https://github.com/NVIDIA/NeMo`

replacing the deprecated OpenAI gym library with gymnasium. We then created a pull request back to the original repository.

Thanks to its modularity, RL4LMs can be easily extended with custom reward functions, on-policy algorithms, metrics, and datasets. Notably, since each produced token is taken as an individual action, it is assigned an individual reward (as opposed to assigning just one scalar reward per sequence). However, because the PPO implementation in RL4LMs is based on `OnPolicyAlgorithm` of SB3, the token reward has to be computed before generating the next token. This is in conflict with our approach which only computes the rewards after the whole sequence has been rolled out.

We suspect that this limitation could be circumvented by making the reward function return proxy objects instead of actual numbers. This could work since in SB3 the rewards are initially simply stored in a rollout buffer and are only evaluated for returns calculation after the episode has ended. The only obstacle seems to be that the rollout buffer is a NumPy array which does not support storing object references, necessitating a simple modification of the SB3 code. We did not pursue this direction for time reasons.

4.2 Custom Implementation

We implement a baseline RL approach based on the REINFORCE algorithm, which we briefly described in Section 1.2.1.

4.2.1 Reinforcement Learning Formulation

We formulate the summary generation as a Markov decision process (MDP) [Bellman, 1957] where an agent (i.e. summarizer) observes the so-far generated prefix together with the state of the encoder (i.e. a representation of the article being summarized) and produces a single token as its action. It then receives a scalar reward associated with the taken action. This environment is deterministic, discrete, and fully observable.

An episode ends when the agent generates an end-of-sequence token or when the maximum number of actions (i.e. generated tokens) is reached. The maximum number of actions is set to 128 for XSum and 160 for CNN/DM. More precisely, the rewards are only calculated after the episode ends. This is required by the reward function that will be introduced in Chapter 5.

Our formulation is consistent with the RL4LMs library. It is in contrast with TRL and trlX libraries in which an action corresponds to an entire summary instead of just one token.

4.2.2 Action Generation

To generate an action (i.e. a token), we sample from the summarizer's output distribution. We disallow the end-of-sequence token in the first 16 steps to enforce a minimum episode length.

Importantly, the output distribution is generated deterministically with dropout turned off. This is necessary for a stable calculation of the reference KL-divergence described in Section 4.2.4. This behavior is consistent e.g. with the TRL library.

We also experimented with using contrastive search [Su et al., 2022] instead of sampling. Contrastive search extends greedy search by introducing a degeneration penalty for token T equal to the maximum cosine similarity between the embedding of T and

embeddings of all tokens preceding T. This is to limit undesirable repetitions. Additionally, contrastive search only considers the top k predictions of the model. In our case, we used a degeneration penalty coefficient $\alpha = 0.6$ and $k = 6$. This approach achieved competitive performance to sampling on our experiments but ultimately did not lead to improved results.

4.2.3 Calculation of Cumulative Discounted Returns

We follow the standard definition of returns as the expected discounted sum of future rewards. Returns are then taken as estimates of state values and used by REINFORCE. With a discount factor $\gamma \in [0, 1]$ and rewards $R_1, \ldots, R_n$ the definition of returns $r_1, \ldots, r_n$ is given in Equation (4.1).

$$r_t = \sum_{i=t}^{n} \left(\gamma^{i-t} R_i\right) + \gamma^{n-t+1} e \tag{4.1}$$

The term e serves as a baseline estimate of the final state value to offset the effect that episode cut-off has on the estimated value of states near episode end. We experimented with setting e to the average of all rewards in the current batch which yielded marginally better performance compared to $e = 0$. We use this approach in all experiments where $\gamma \neq 0$.

4.2.4 Reference KL-Divergence Anchoring

If the only objective being optimized was an arbitrary reward function, the model would have no incentive to keep producing natural language. Indeed, this is the case even in our test task of maximizing the number of words starting with the letter `T`: the model quickly starts only repeating the token *"the"*.

We use the standard remedy of adding another minimization criterion calculated as the Kullback-Leibler divergence (KL-divergence) [Kullback and Leibler, 1951] between output distributions of the trained model and a frozen reference model. The KL-divergence then also acts as an entropy regularizer in that it encourages exploration [Stiennon et al., 2020]. In our case, the frozen model is a copy of the initial version of the trained model.

In the context of RLHF, the KL-divergence anchoring approach was first used by Jaques et al. [2019]. It was then employed in InstructGPT [Ouyang et al., 2022] and is implemented in all the open-source libraries listed in Section 4.1. It was also used by Anthropic [Bai et al., 2022a] and DeepMind [Glaese et al., 2022] for their RLHF implementations.

However, various details differ across implementations. Notably, there are several ways to calculate the deviance between the model and reference distributions. We follow TRL (described in Section 4.1.1) and implement four different strategies listed below. For notation, let π be the trained model, π^{ref} the frozen reference model, x an input to the model (i.e. input prompt and already generated prefix), and k the argmax over $\pi(x)$ (i.e. the most probable next token).

- `full_kl` – KL-divergence $D_{\text{KL}}(\pi(x)|\pi^{\text{ref}}(x)) = \sum_{i=1}^{n} \pi(x)_i \log \frac{\pi(x)_i}{\pi^{\text{ref}}(x)_i}$
- `diff` – Difference of log-probabilities $\log \pi(x)_k - \log \pi^{\text{ref}}(x)_k = \log \frac{\pi(x)_k}{\pi^{\text{ref}}(x)_k}$
- `abs_diff` – Absolute difference of log-probabilities $\left|\log \pi(x)_k - \log \pi^{\text{ref}}(x)_k\right|$

- `rmse` – Root-mean-square error between log-probabilities $(\log \pi(x)_k - \log \pi^{\text{ref}}(x)_k)^2$

Traditionally, the resulting deviance is then subtracted from the per-token reward. However, using this approach we weren't able to find a choice of hyperparameters leading to summaries that are both coherent and meaningfully maximize the reward in our test task. We hypothesize that the difficulty stems from the sparsity of both the anchoring signal and the reward signal.

To solve this, we observe that all the deviance strategies are differentiable. Therefore, they provide a richer signal than a classic RL reward in that they show how the model's output should be improved instead of just giving its quality. To utilize this, we minimize the deviance directly with back-propagation rather than subtracting it from the reward. In PyTorch, we do this by adding the deviance to the loss with a suitable coefficient.

We use this approach with the `full_kl` deviance which differs from others in that it is calculated from the full output distribution instead of just the highest log-probabilities. We call this new strategy `full_kl_loss`. The resulting loss can then be written as:

$$L(x) = L_{\text{REINFORCE}}(x) + \beta D_{\text{KL}}(\pi(x) \parallel \text{stop\_gradient}(\pi^{\text{ref}}(x))) \tag{4.2}$$

Notice that in the second term of Equation (4.2), gradient updates are not propagated to the frozen reference model. The first term of Equation (4.2) is the REINFORCE loss described in Section 1.2.1.

A potential disadvantage of the `full_kl_loss` strategy is that for cases when $\gamma > 0$, adding KL-divergence directly to the loss means that this signal does not propagate to the preceding tokens. When it is subtracted from the reward instead, preceding tokens are affected through the cumulative returns. In other words, `full_kl_loss` prevents tokens from "seeing" their effect on the coherence of the following text.

4.2.5 Qualitative Analysis

Along with logging metrics using Tensorboard[3] [Abadi et al., 2016] we build a custom visualization application using Streamlit[4] to monitor the evolution of rewards on summary tokens over time. The application tracks the first 25 validation set samples, one of which is shown in Figure 4.1. Visible are the article (cropped), a slider for timestep selection, and the summary with positive rewards marked red and negative rewards marked blue.[5]

4.3 Test Task

We evaluate different approaches using a test task of maximizing the number of output words starting with the letter `T` (case-insensitive). This task is designed to be simple and easy to evaluate, allowing us to experiment with our reinforcement learning implementation in isolation before applying it with our custom reward which is more complex. With this, we obtain a reasonable configuration of all hyperparameters which we then use as a starting point in Chapter 6.
The test reward for token t is defined as:

[3] `https://www.tensorflow.org/tensorboard`

[4] `https://streamlit.io/`

[5] In this example we use the reward function introduced in Chapter 5.

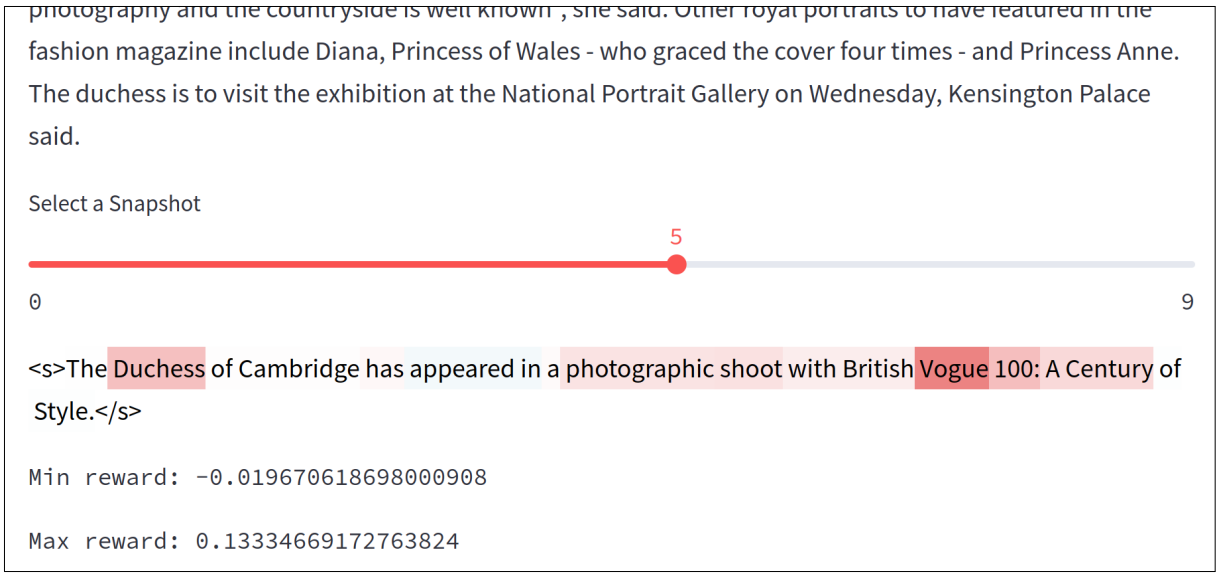


Figure 4.1: Interactive browser application for monitoring rewards during training.

$$r(t) = \begin{cases} 0, & \text{if } t \text{ is not the first token of a word} \\ 1, & \text{if } t \text{ is the first token of a word and begins with a } \texttt{T} \\ -\frac{c}{1-c}, & \text{if } t \text{ is the first token of a word and does not begin with a } \texttt{T} \end{cases} \tag{4.3}$$

In Equation (4.3), $c = 0.16$ is the approximate frequency of English words starting with `T`. The negative reward for the beginnings of word starting with a different letter is chosen so that for an average English sentence, the sum of rewards is 0.

4.3.1 Dataset

We initialize the model from checkpoints that we obtained in Chapter 3 through supervised fine-tuning. We also use the XSum dataset described in Section 2.1 with labels removed. However, we discard the first 50% of train data, because the model has already seen it during the supervised fine-tuning.

4.3.2 Results

We performed a manual search over hyperparameters to find an approximate local optimum on the XSum dataset. The final choice of hyperparameters is listed in Table 4.1. The training curves with reward and KL-divergence are shown in Figure 4.2. BART-large yields higher reward and higher reference KL-divergence than BART-base. Note that the KL-divergence cannot be directly compared since the reference models differ in both cases.

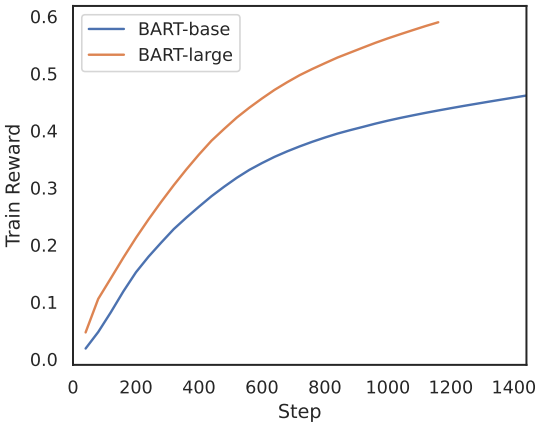
Finally, Table 4.2 lists a few output samples from the evaluation split, together with gold data and a comparison to the output of Llama 2 70B [Touvron et al., 2023]. BART does not attain the creativity and fluency of Llama 2 70B but is able to increase the number of words starting with `T` while preserving some of the summary's message and coherency, although it frequently employs fixed phrases carrying no meaning like *"has taken to the top two"*. Interestingly, Llama 2 70B fails to accomplish the goal in examples (2) and (3),

Hyperparameter	**Value**
Samples per update	16
Learning rate	1e-5
Optimizer	Adam
KL-divergence strategy	`full_kl_loss`
β (KL-divergence coefficient)	0.3
γ (discount factor)	0
Normalize returns	False
Number of updates	~1200

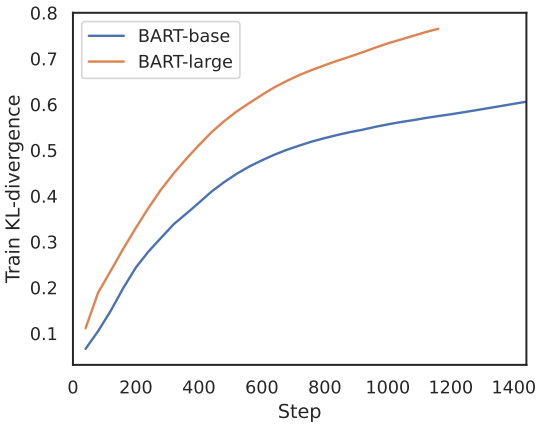
Table 4.1: Test task hyperparameters.

hallucinating unrelated information or using only a normal number of words starting with `T`. We believe this could be rectified with more complex prompting.

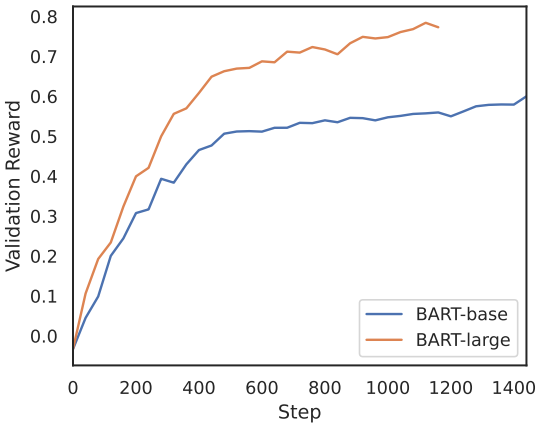
These quantitative and qualitative results give us the confidence to move forward with using more complex reward functions.



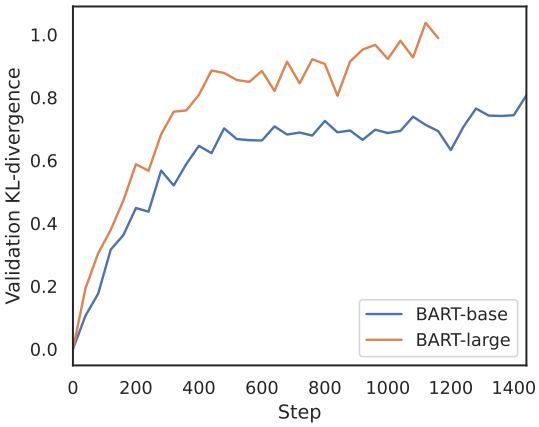
(a) Training Reward



(b) Train KL-divergence



(c) Validation Reward



(d) Validation KL-divergence

Figure 4.2: Test task RL training of BART on XSum.

Gold summary
(1) Former Premier League footballer Sam Sodje has appeared in court alongside three brothers accused of charity fraud.
(2) The Duchess of Cambridge will feature on the cover of British Vogue to mark the magazine's centenary.
(3) Google has hired the creator of one of the web's most notorious forums - 4chan.
BART-base after supervised fine-tuning
(1) Sam Sodje has been remanded in custody after appearing in court accused of embezzling £1m from a sports charity in Nigeria.
(2) The Duchess of Cambridge has been photographed for the first time in British Vogue's 100th anniversary issue.
(3) Google has appointed the former administrator of notorious social networking site 4chan to lead its social networking efforts.
BART-base after 1040 T-maximizing RL updates
(1) Sam Sodje has taken to the top two to the tens of thousands of pounds to try to take to the streets of the town to raise money for Nigerian sport.
(2) The Duchess of Cambridge has taken to the top two to appear on the cover of British Vogue to mark the publication of the magazine's 100th anniversary.
(3) Google has taken to the top two to try to take the time to think through the threat posed by the social networking site 4chan.
Llama 2 70B prompted for T-maximizing *
(1) Trial set for tarnished footballers accused of tainted trading tactics totaling tens of thousands.
(2) Two tributes to tennis talent take time to transform into timeless treasures.
(3) Chris Poole, aka "moot", creator of controversial imageboard website 4chan, joins Google as an executive, bringing his expertise in building online communities to the tech giant's social networking efforts.

* The exact prompt: *Please summarize the following article in one sentence using only words that start with the letter t*

Table 4.2: Samples of test task output on validation data.

5. Reward Function for Summarization

In this chapter, we define a per-token reward function that rewards summaries based on their quality. For our purposes, a high-quality summary is one that satisfies the following:

1. Is short, coherent, and written in English.
2. Contains as many important facts from the summarized text as possible.
3. Contains as few wrong or misleading facts as possible.

Ultimately, our goal is to reward a model for all these traits using the reinforcement learning approach described in Chapter 4. The reward function that we develop in this chapter is meant to target the last two traits. We assume the first trait is already present and only needs to be preserved as much as possible during RL training using the KL-divergence method described in Section 4.2.4.

5.1 Our Approach

We introduce a separate instance of BART called *predictor*. For disambiguation, we call the trained model *summarizer*. Like the summarizer, the predictor is meant for sequence-to-sequence prediction on the summarization dataset. However, its inputs are summaries and its outputs are the original articles being summarized (i.e. the direction is flipped with respect to the summarizer).

The idea behind our reward function is to approximate the importance of a summary token T by how much harder it is for the predictor to predict the summarized article when we mask out T. We hypothesize that masking out an important informative token T should make it harder to predict the article. On the other hand, masking out a misleading token should make the prediction easier. Finally, masking out an irrelevant "filler" token should not have a big influence on the prediction.

The ease of prediction is measured by considering the model's cross-entropy on the article tokens. The exact details and their evaluation are the subject of the rest of this chapter.

5.1.1 Masking Strategies

For every generated summary $S = T_1 \dots T_n$, we create a set of binary masks $\{M_1, \dots, M_m\}$, each of length n. Let $P(M_j)$ be the indices where mask M_j is set (i.e. where masking occurs). The result of the application of a mask $M_j(S)$ is an altered summary S' which is equal to S except for the positions $k \in P(M_j)$ where S'_k is set to a masking token K.

The predictor is run individually with each of $M_1(S), \dots, M_m(S)$ as input to predict the summarized article (the executions are batched), details of which are described in Section 5.1.2. This gives us *"confusion coefficients"* $c_1, \dots, c_m$ representing how much the predictor *"got confused"* by each of the individual masks.

The reward for a token T_i is then dependent on those c_j where $i \in P(M_j)$, i.e. on results from masks where T_i is masked. If T_i is not masked by any mask, we set its reward

to 0. By the nature of the REINFORCE algorithm, such tokens are disregarded during gradient computation.

On the other hand, suppose that T_i is masked at least once and possibly multiple times, obtaining the corresponding subset of *confusion coefficients* $c_1, \ldots, c_p$. To obtain a single scalar reward R_i we use one of the following strategies, calling this hyperparameter `samples_reducer`:

- `average`, where $R_i = \frac{1}{p} \sum_{l=1}^{p} c_l$,
- `max`, where $R_i = \max c_1, \ldots, c_p$.
- `min`, where $R_i = \min c_1, \ldots, c_p$,

As for the masking token K, we experiment with BART tokens `<unk>`, `<mask>`, and `<pad>`, together with a `random` strategy which selects from all tokens at random for each occurrence. However, we discarded the `<pad>` early in the process because of its inferior performance.

The currently implemented mask-generation strategies are as follows. An example of a mask for each of these strategies is shown in Figure 5.1.

- `random` – Generate `mask_count` masks. For each mask and each position, decide independently randomly with probability `mask_prob` whether to mask the position or not.
- `tokenwise` – Create a separate mask for each position where only that one position is masked. This means that the mask count is equal to the summary length.
- `random_bursts` – Generate `mask_count` masks as follows. First, select some positions at random. For each selected position, mask it together with the following `burst_length`-1 tokens to form a masked burst. The number of bursts is calculated so that the ratio of masked tokens is approximately `mask_prob`.
- `bursts` – For each possible burst (substring of length `burst_length`), create a mask where nothing but this burst is masked. Furthermore, create shorter masks on both ends of the summary so that each token is masked exactly `burst_length`-times. This means that for reasonably long summaries, the mask count is equal to $n + 2(\texttt{burst\_length} - 1)$ where n is the summary length.
- `words` – Define *words* by splitting the de-tokenized summary along spaces (with no additional preprocessing). Each mask is then constructed by masking a burst of `word_burst_length` consecutive words. As in the `bursts` strategy, both ends of the summary are additionally masked so that each word is masked exactly `word_burst_length`-times.
- `random_variable_bursts` – This strategy is only used for predictor training as described in Section 5.1.4. It works by choosing the length of a burst at random between 1 and `burst_length` and generating a mask where nothing but this burst is masked.

Importantly, the beginning-of-sequence (BOS) and end-of-sequence (EOS) tokens are never masked since their absence causes BART to have radically higher cross-entropy and obscures any signal.

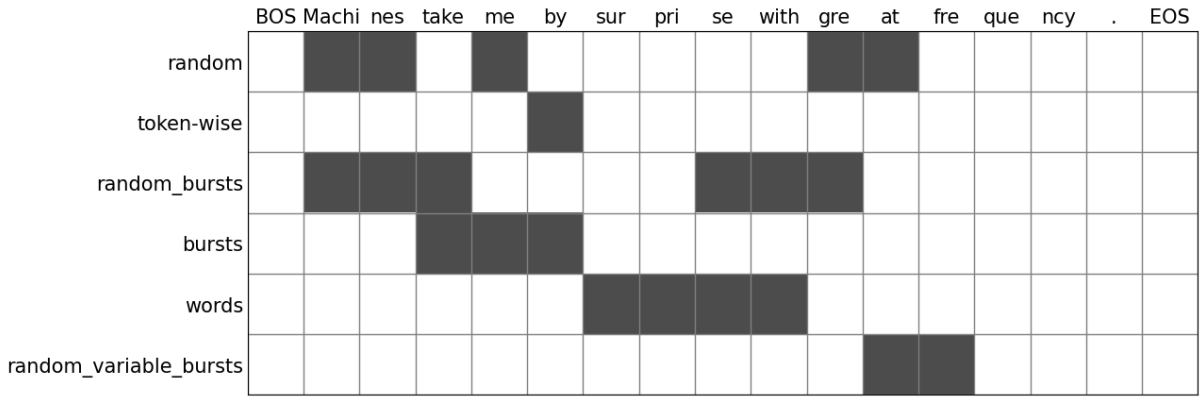


Figure 5.1: An example of a mask for each mask-generation strategy. The tokenization is only illustrative, unrelated to the BART tokenizer. In all examples, `burst_length` = 3 and `word_burst_length` = 2.

5.1.2 Confusion Coefficients

Here we discuss the computation of the *confusion coefficients* introduced and used in the previous Section 5.1.1. A scalar confusion coefficient is calculated for each mask. Intuitively, it expresses the amount of "confusion" caused to the predictor by obscuring the masked tokens.

After applying a mask, we start with predictor's perplexities on each of the original article tokens as compared to perplexities before masking. These are then weighted and reduced either by averaging or by taking the maximum. An example of the relative perplexities before reduction is shown in Figure 5.8.

We now express this idea mathematically. Let S be a summary, M a mask, and $M(S)$ the application of mask M on summary S as in the previous section. Furthermore, let A be the summarized article, w_t be arbitrary weights, and $\text{predictor}(S)_t$ be the predictor's output distribution on the t-th token with $\text{predictor}(S)_{t,A_t}$ being the predicted probability of the correct token. An example of weights w_t will be described in Section 5.1.3. Finally, let $\mathcal{R}$ be either `max` or `average`. We call this hyperparameter `loss_reducer`.

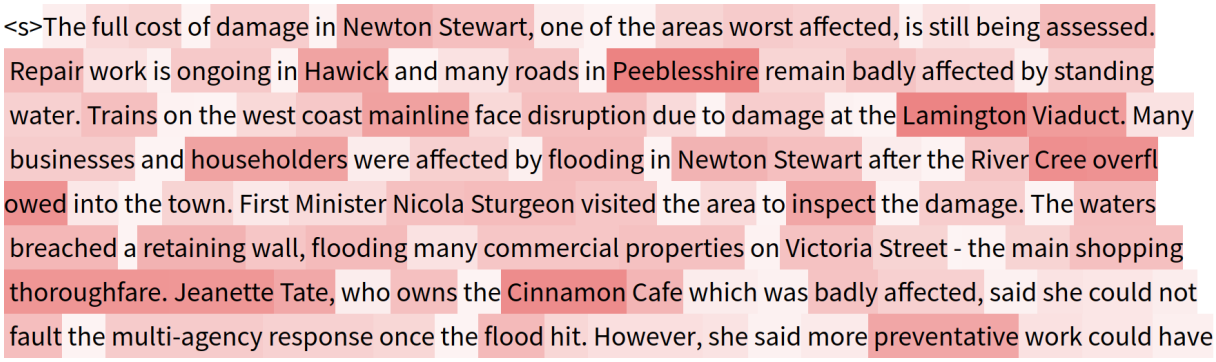
The *confusion coefficient* c_j can then be expressed as:

$$c_j = \mathcal{R}_{t=1}^{|A|} \left(-w_t \log \frac{\text{predictor}(M(S))_{t,A_t}}{\text{predictor}(S)_{t,A_t}} \right) \tag{5.1}$$

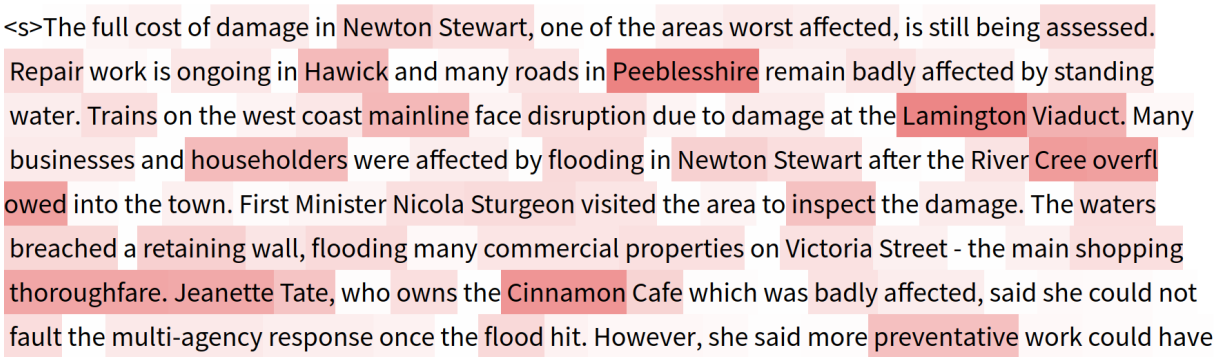
In our implementation, we split the logarithm into difference of logarithms so that the logarithm and predictor's softmax can be fused to make back-propagation more numerically stable. The bulk of the calculation is then the standard cross-entropy loss with a custom reduction, as available in PyTorch. However, we use our custom implementation which exploits the fact that the labels (i.e. the article A) are the same for all invocations of the loss function and thus don't have to be duplicated.

5.1.3 IDF Weights

Confusion coefficients defined in Equation (5.1) are designed so that tokens of article A can be arbitrarily weighted to have more or less significance in the calculation. We implement weighting based on IDF (inverse document frequency) of *words* (not tokens). This puts more emphasis on rare words in the article such as names of people and places.



(a) $\omega = 1$



(b) $\omega = 2$

Figure 5.2: IDF weights on a part of an article from XSum. Darker colors correspond to higher weights.

However, its efficacy is to be tested since not all information-dense utterances consist of rare words.

We start by calculating IDF of words from the second 50% of training data articles. Words are defined by the regular expression `(?u)\b[a-zA-Z]+\b`. Words seen in fewer than 3 documents are disregarded (i.e. cut-off is 3).

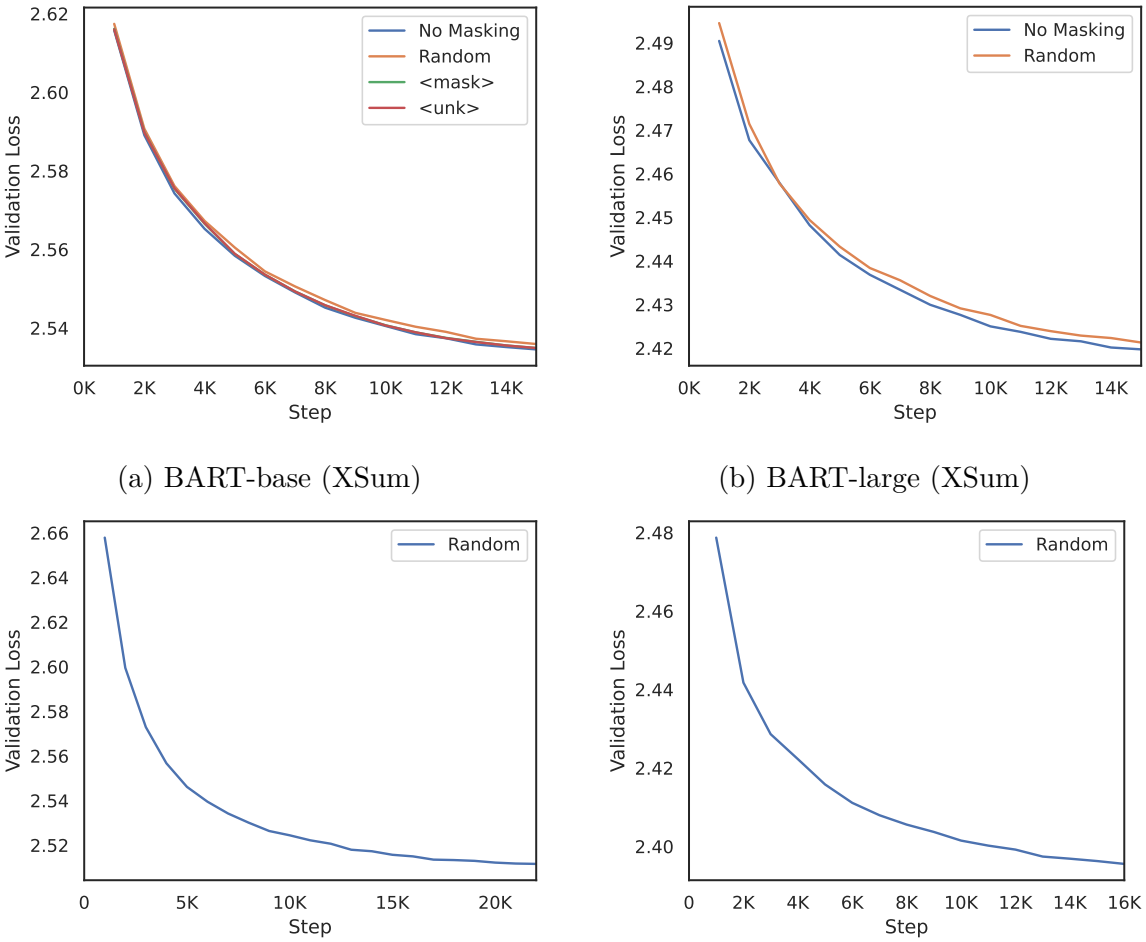
IDF values are then used when calculating the confusion coefficients as follows. Suppose the t-th token belongs to a word with IDF I. In this case, words are obtained simply by splitting the text on spaces and newlines. To combat punctuation and other characters not considered during IDF computation, we additionally find all occurrences of `(?u)\b\w+\b` in each word to perform the IDF search. When more occurrences are found, we take their average IDF. Words not found in the training corpus are assigned the maximum from all IDF values.

We first set $w'_t = I^\omega$ where ω is a hyperparameter. Then, the values w'_t across all tokens are normalized so that the average value is equal to 1, obtaining the final weights w_t.

Figure 5.2 shows an example of the resulting weights on a part of an article from XSum with different values of ω.

5.1.4 Supervised Fine-Tuning of Predictor

We obtained the predictor checkpoints by fine-tuning BART on the first 50% of each corresponding dataset with inputs and outputs swapped. Hyperparameters were chosen identically to summarizer fine-tuning as listed in Table 3.1. We call the resulting checkpoints



(a) BART-base (XSum)

(b) BART-large (XSum)

(c) BART-base (CNN/DM)

(d) BART-large (CNN/DM)

Figure 5.3: Validation loss during predictor fine-tuning.

`predictor_base` (based on BART-base) and `predictor_large` (based on BART-large). Without this fine-tuning, the performance of our metric was radically worse.

Additionally, we perform this fine-tuning with masked versions of the inputs using the `random_variable_bursts` masking strategy described in Section 5.1.1. This is to minimize the distribution shift between the fine-tuning data and the data that the predictor will ultimately be used on. In the case of BART-base we experimented with all the `<mask>`, `<unk>`, and `random` mask tokens, obtaining checkpoints `predictor_base-mask`, `predictor_base-unk`, and `predictor_base-random`. However, `predictor_base-random` dominated all our evaluations and was chosen as the only predictor for the follow-up grid search in Section 5.2.3. Therefore, in the case of BART-large, we only used the `random` strategy, obtaining checkpoint `predictor_large-random`.

Plots of all the fine-tuning runs are shown in Figure 5.3. All of these runs are stable and close to convergence.

5.2 Quantitative Analysis

To quickly identify promising hyperparameter configurations we design quantitative metrics of the reward function's performance, i.e. how effective it is in discerning relevant

summary tokens from irrelevant or misleading tokens. We present two such metrics and argue that both are useful.

5.2.1 Paired Test Using a Distractor Article

First, we describe a metric which is based on a distractor article. For each data sample consisting of an article A and a summary S we choose another *distractor article* A' at random. We use the article A to compute per-token rewards $p_1, \ldots, p_n$ on S (positive samples) and article A' to compute per-token rewards $n_1, \ldots, n_n$ on S (negative samples). With this, we obtain a set of positive measurements P and a set of negative measurements N on the same population, i.e. the same summary tokens. We then employ paired one-sided statistical tests with the null hypothesis that P is not stochastically greater than N. The following tests are performed:

- One-sided Wilcoxon signed-rank test. The test works by computing the differences $X_i = P_i - N_i$, assigning ranks to them based on their absolute value, and computing the following statistic (R_i denotes rank of X_i):

$$T = \sum_{i=1}^{n} \operatorname{sgn}(X_i) R_i. \tag{5.2}$$

- One-sided Student's t-test, which computes the following statistic:

$$t = \frac{\bar{d}}{\sqrt{s^2/n}}, \tag{5.3}$$

  where $\bar{d}$ is the mean difference and s^2 is the sample variance. This test assumes that the differences are approximately normally distributed.

Through statistical tests, we obtain a p-value. However, because of the large number of samples and relatively large difference between P and N, the p-values are too close to zero to be distinguishable. Since we are only interested in the relative ordering of reward functions, not in the absolute p-values, we only consider the T and t statistics themselves. If one reward function yields substantially larger statistic than another one, we use that as a signal about their relative quality. In our experiments, an average of about 90% reward function pairs are ordered identically when using T or t metrics.

The Distractor Article approach is theoretically pleasing because of its simplicity and paired measurements, i.e. the positive and negative samples are obtained on the same tokens. There are fewer arbitrary decisions to be made when designing this test compared to the one introduced in the next Section 5.2.2. The result gives us a hint about a reward function's ability to assign greater rewards to summaries that contain some relevant information about the article. This is mainly affected by the predictor architecture and pre-training, mask token, and `loss_reducer`.

However, this metric does not take into account the distribution of rewards among individual tokens in a summary, which makes it unusable for testing hyperparameters like `burst_length`, `masking_strategy`, or `samples_reducer`. For example, increasing `burst_length` and setting `samples_reducer` to `max` will typically increase P more than N, thus improving the metric even though this makes the reward function lose its ability to correctly allocate reward inside a summary. In the limit, a uniform function assigning the same reward to all summary tokens might seem to perform well under this metric while being useless in practice.

5.2.2 Independent Test Using a Distractor Summary

To address shortcomings of the Distractor Article metric discussed in the previous Section 5.2.1, we introduce a different approach based on a distractor summary. For each data sample consisting of an article A and a summary S we choose another *distractor summary* S' at random. We then construct an artificial summary T by selecting individual words from either S or S' at random. In this context, words are defined by simply splitting along spaces and newlines.

The words are not sampled independently; instead, the first token is taken either from S or from S', and for each following token, the chance of it being sampled from the same summary as the previous one is 60 %. This approach was chosen by manually inspecting the resulting summaries and trying to maximize their coherence while still switching between S and S' often enough. Importantly, all the random choices have a fixed seed and are therefore identical across evaluation runs.

Afterward, we compute per-token rewards $r_1, \ldots, r_{n+m}$ on T and split them into positive samples $p_1, \ldots, p_n$ and negative samples $n_1, \ldots, n_m$ based on whether the corresponding token came from S or S'. With this, we again obtain a set of positive measurements P and a set of negative measurements N. However, the measurements are now not on the same population (i.e. the same summary tokens) and in general have different sizes (although in our case the sizes are similar). We therefore perform independent one-sided statistical tests instead of paired tests. We use the following tests with the null hypothesis that P is not stochastically greater than N:

- One-sided Mann–Whitney U test (also known as Wilcoxon rank-sum test). Works by combining both sets of numbers, summing ranks of samples from each set to get R_1 and R_2, and computing the following statistic:

$$U_1 = nm + \frac{n(n-1)}{2} - R_1. \tag{5.4}$$

- One-sided Welch's t-test, which is an adaptation of Student's t-test without the assumption of equal variance of P and N. However, the assumption that the tested values are normally distributed is maintained. The statistic t is given by:

$$t = \frac{\bar{X}_1 - \bar{X}_2}{\sqrt{s^2_{\bar{X}_1} + s^2_{\bar{X}_2}}}, \tag{5.5}$$

  where $\bar{X}_i$ are population means and $s_{\bar{X}_1}$ are its standard errors.

Again, we only consider U_1 and t statistics themselves instead of calculating p-values. It is important to note that since the artificial summary T is created by concatenating *words* from S and S', we expect it to favor the `words` masking strategy.

5.2.3 Grid Search

After manually experimenting with all the different hyperparameters, we designed and performed the following grid search:

- predictor checkpoint: `predictor_base-random`, `predictor_base-mask`,
- mask token: `random`, `<mask>`,

- `loss_reducer`: `average`, `max`,
- `samples_reducer`: `average`, `max`, `min`,
- `mask_strategy`: `bursts` (using `burst_length` $\in \{2, 3, 4\}$), `words` (using `word_burst_length` $= 2$),
- ω (IDF weight power): 0.0, 1.0, 2.0.

We visualize the results of the grid search separately for each combination of predictor checkpoint and mask token. Out of all 4 combinations the (`predictor_base-random`, `random`) performed best with results shown in Figure 5.4. Each run is plotted in a 2-dimensional space corresponding to the two metrics introduced in Section 5.2.1 and Section 5.2.2. Each run is drawn as a single symbol so that `mask_strategy` corresponds to shape, `samples_reducer` corresponds to outer color, `loss_reducer` corresponds to inner color, and ω (IDF weight power) corresponds to size. More specifically, the correspondence is as follows:

- `loss_reducer`
  - `average`: green
  - `max`: red
- `samples_reducer`
  - `average`: blue
  - `max`: orange
  - `min`: purple
- `mask_strategy`
  - `bursts`
    - `burst_length` $= 2$: circle
    - `burst_length` $= 3$: triangle
    - `burst_length` $= 4$: square
  - `words`, `word_burst_length` $= 2$: star
- ω (IDF weight power)
  - 0.0: small
  - 1.0: medium
  - 2.0: large

The resulting Figure 5.4 suggests that `loss_reducer` strategy `average` is more effective in discerning relevant summary tokens from irrelevant ones than the `max` strategy, indicated by the red cluster on the left. We hypothesize that this is because `max` is more prone to noise in predictor's perplexities than `average`.

The same figure also shows a clear trend when increasing `burst_length` from 2 to 3 to 4 – the reward becomes less effective in discerning relevant tokens (symbols move to the

Hyperparameter	Value
Predictor checkpoint	`predictor_base-random`
Mask token	`random`
`loss_reducer`	`average`
`samples_reducer`	`min`
`mask_strategy`	`words` (`word_burst_length` = 2)
ω (IDF weight power)	$\{0.0, 1.0\}$ (*inconclusive*)

Table 5.1: Best reward function configuration based on grid search.

left). Furthermore, the effect of IDF weight power is relatively small when `loss_reducer` is `max`, indicated by the different sizes of the same symbol being generally close together.

We choose the Distractor Summary Mann-Whitney statistic as the primary measure of quality and report the best hyperparameter configuration based on this metric in Table 5.1. However, we only use this configuration as a starting point and experiment with all the hyperparameters further in Chapter 6.

The (`predictor_base-random`, `mask`) and (`predictor_base-mask`, `mask`) combinations are competitive while (`predictor_base-mask`, `random`) performs substantially worse. Their visualizations are provided in Appendix A.

Scaling the Predictor to BART-large

We then run some of the best performing and some of the worst performing configurations with a larger predictor model `predictor_large-random`. The resulting differences are shown in Figure 5.5. To our surprise, using a stronger predictor model hurts performance on our metrics in all tested cases. This suggests there might be an optimal middle point in predictor size where weaker models do not provide enough signal and stronger models are not *"confused"* enough by masking important words.

We observe that if the predictor model is too strong, it can be better than a human in inferring a masked section of a summary from the non-masked context, leading to small reward for phrases that a human reader might consider useful. For example, consider a summary containing the phrase "Welsh cyclist Luke Rowe". When the phrase "Welsh cyclist" is masked, a strong predictor might be able to infer it with relative certainty from the context "Luke Rowe". This means that parts of the original article pertaining to Wales and cycling would not suffer any detriment in prediction perplexities, yielding small reward for the phrase "Welsh cyclist" which a human reader might consider important. Our metrics capture this human preference because "Welsh cyclist" is contained in the reference summary.

We use the results from this section in that we avoid training a BART-large version of the predictor in the case of CNN/Daily Mail, which would be relatively expensive due to long article lengths compared to XSum. In the case of XSum we still train and experiment with both BART-base and BART-large versions of the predictor.

5.3 Qualitative Analysis

On top of the general training monitoring tools described in Section 4.2.5, we develop tools for debugging reward functions which we describe in this section.

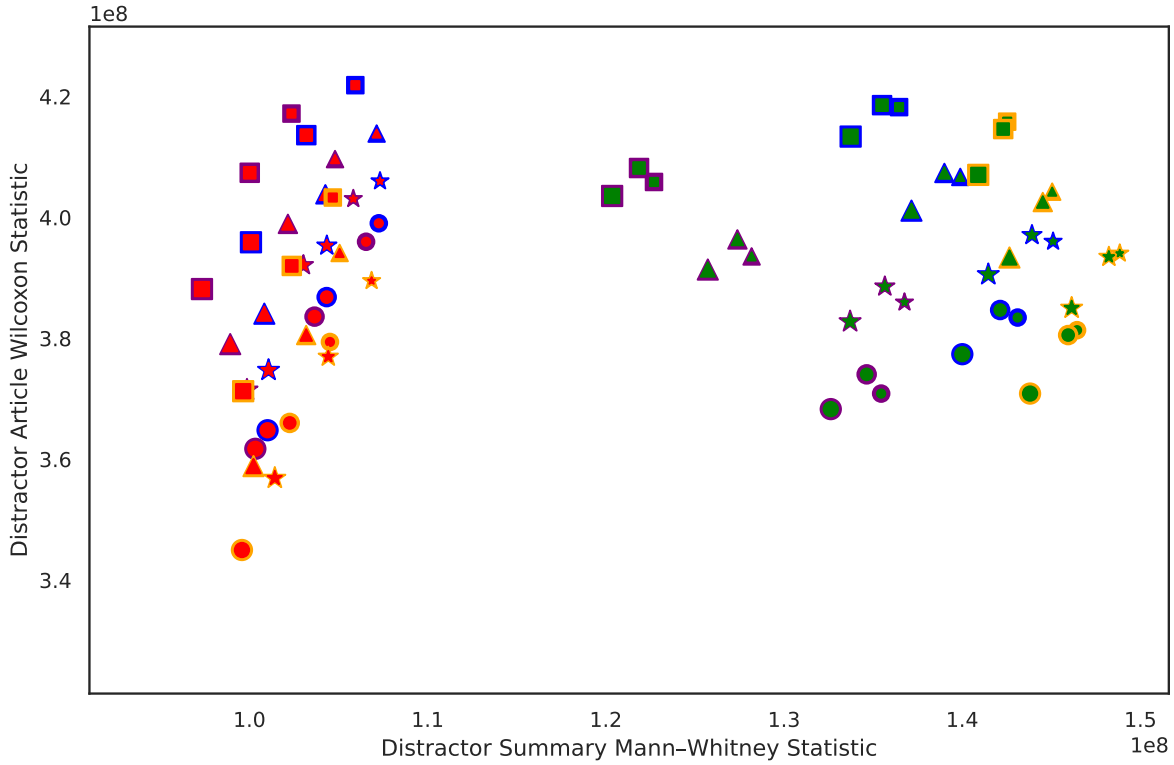


Figure 5.4: Grid search results for parameters `mask_token` = `<random>`, predictor = `predictor_base-random`.

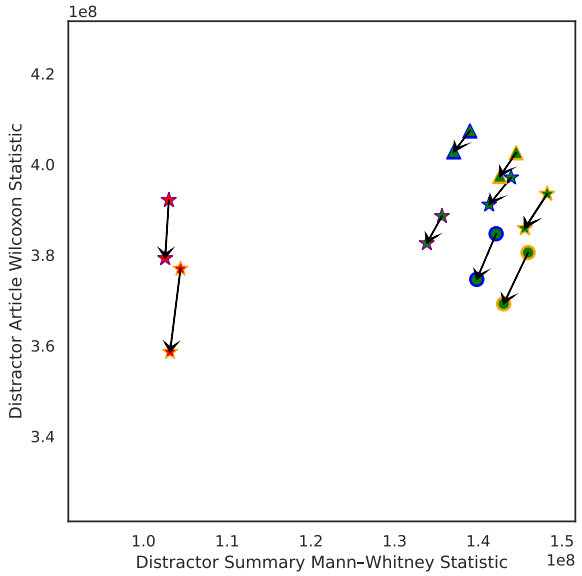


Figure 5.5: Comparison between `predictor_base-random` and `predictor_large-random` on select configurations.

5.3.1 Interactive Browser Application

To facilitate a quick qualitative feedback loop when experimenting with different reward functions we implement an interactive browser application using Streamlit. The application allows the user to choose values for all relevant hyperparameters and run the reward function on a selected article-summary pair. Along with the resulting reward values, the user can also choose to visualize either the IDF weights or all the masks with their corresponding cross-entropy errors on the article tokens. A screenshot of the application is shown in Figure 5.6.

5.3.2 Reward Examples

Concrete rewards on the first four summaries from XSum validation set[1] are shown in Figure 5.7. We include the beginning-of-sequence and end-of-sequence tokens for completeness, although they are ignored during reinforcement updates. The figure also shows a tampered version of each summary where a piece of correct information is replaced with a false one.

We also monitor the aggregate distributions of positive and negative rewards over all data samples. An example of such distributions taken from the best-performing grid search run on XSum is shown in Figure 5.9. The positive rewards are clearly stochastically greater than the negative rewards.

Analysis of the Provided Examples

As hinted by Figure 5.7, in many cases the reward function correctly identifies salient pieces of information such as names of people and organizations (e.g. Sam Sodje, Adam Voges, Vogue, 4chan, Google) and assigns to them a positive reward. Similarly, the reward detects obviously wrong information in the tampered summaries (Jan Novak, Czech, Nature, fired), assigning them a negative reward.

However, not all important information is rewarded and not all false information is penalized. For example, the substring "Former Premier League footballer" in example (A) is not rewarded even though a human reader might find it relevant. That is because the corresponding article pertains more to a court case than to football and does not mention the Premier League, meaning that masking the phrase "Premier League" in the summary does not make it substantially harder to predict the article.

Similarly, the phrase "torn calf muscle" in example (B) is not rewarded because the corresponding article only mentions an "injury" without specifying any details. In general, our reward function cannot reward relevant context not present in the article. In fact, there is fundamentally no way to detect if such a piece of information is correct if we only have access to the document which does not mention it. This unfortunate property of XSum is caused by the process of its creation where the first sentence of a BBC article is taken to be its summary but is often crucial for the understanding of the whole article.

Furthermore, different parts of one information piece such as a name can be rewarded differently, as is the case for "Sam Sodje" in example (A) and "Adam Voges" in example (B). In both cases, masking a person's surname causes higher predictor perplexity on the article than masking their forename. This is partly because predicting a common forename is easier than predicting a less common surname. It is also because both full

[1]The hyperparameter configuration is `mask_token = random`, `loss_reducer = average`, `samples_reducer = min`, with IDF weighting turned on.

names are also present in the training data, so the model can infer the forename from the surname.

Other Known Problems

Another issue with the design of our reward function is that incorrect information in the summary might still be helpful in predicting the article. For example, consider the summary "A taxi driver has been involved in a collision with a pedestrian in Inverness." Although the incident took place in Dundee, masking the mention of Inverness hurts article prediction because both cities are located in Scotland. The mention of a Scottish city is a good hint in predicting various parts of the article, such as the name Caird Hall (a concert auditorium in Dundee). At the same time, since the article does not explicitly mention Dundee, listing the wrong city in the summary has little negative effect. We discuss a possible mitigation of this problem in Section 6.4.

5.3.3 Cross-Entropy Error Examples

For an example of how masking part of a summary affects the prediction of an article, Figure 5.8 shows cross-entropy errors on the article tokens when the words "Sam Sodje" are masked. The predictor is expectedly "confused" when predicting a name in the article that has been masked in the summary.

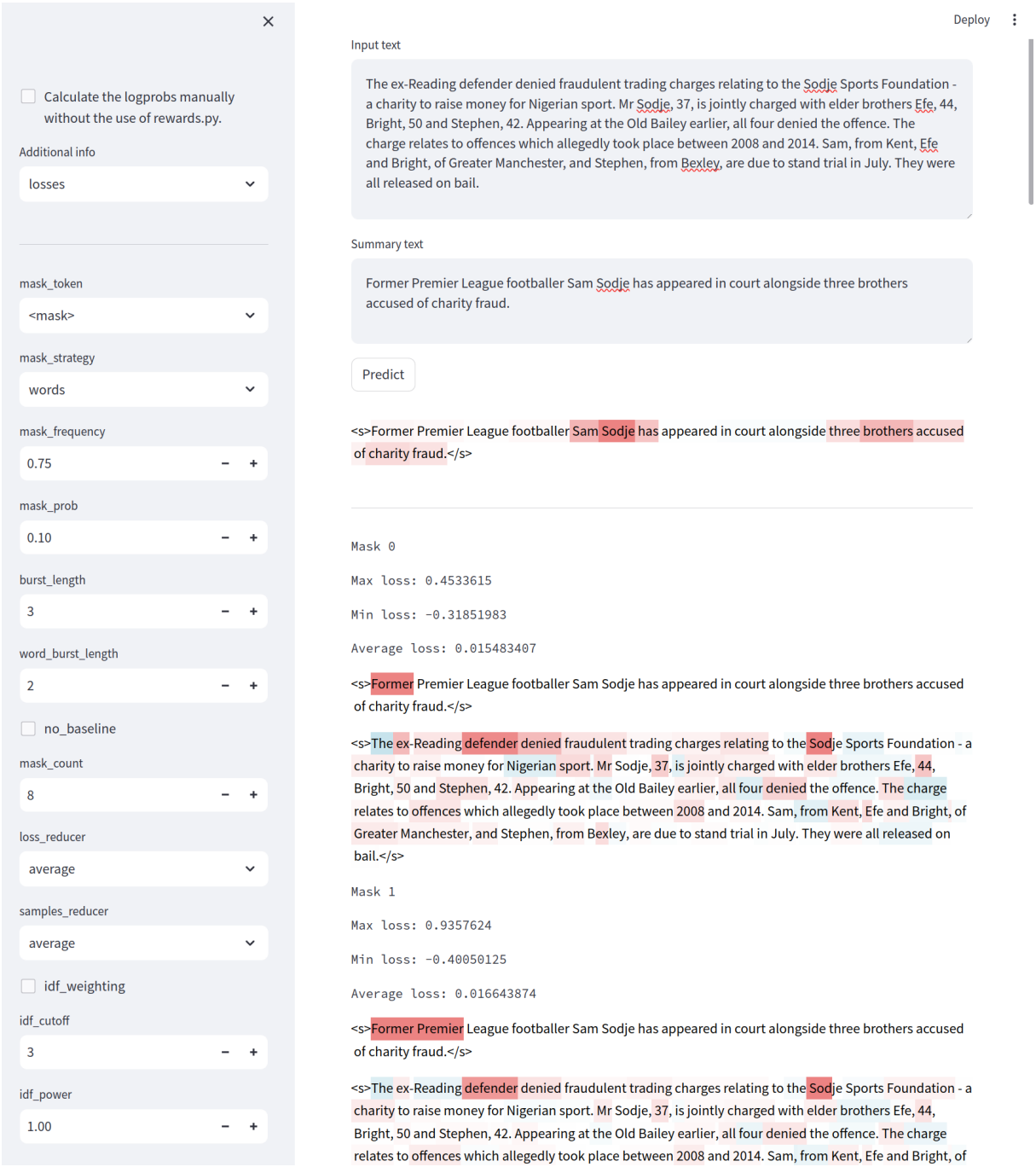


Figure 5.6: Interactive browser application for experimenting with different reward settings.

(A)

<s>Former Premier League footballer Sam Sodje has appeared in court alongside three brothers accused of charity fraud.</s>

<s>Former Premier League footballer Jan Novak has appeared in court alongside three brothers accused of charity fraud.</s>

(B)

<s>Middlesex batsman Adam Voges will be out until August after suffering a torn calf muscle in his right leg.</s>

<s>Czech batsman Adam Voges will be out until August after suffering a torn calf muscle in his right leg. </s>

(C)

<s>The Duchess of Cambridge will feature on the cover of British Vogue to mark the magazine's centenary .</s>

<s>The Duchess of Cambridge will feature on the cover of Nature to mark the magazine's centenary.</s>

(D)

<s>Google has hired the creator of one of the web's most notorious forums - 4chan.</s>

<s>Google has fired the creator of one of the web's most notorious forums - 4chan.</s>

Figure 5.7: Examples of rewards on XSum validation set. Red colors correspond to positive values, blue to negative. Every second sample is altered to contain false information.

<s>Former Premier League footballer Sam Sodje has appeared in court alongside three brothers accused of charity fraud.</s>

<s>The ex-Reading defender denied fraudulent trading charges relating to the Sodje Sports Foundation - a charity to raise money for Nigerian sport. Mr Sodje, 37, is jointly charged with elder brothers Efe, 44, Bright, 50 and Stephen, 42. Appearing at the Old Bailey earlier, all four denied the offence. The charge relates to offences which allegedly took place between 2008 and 2014. Sam, from Kent, Efe and Bright, of Greater Manchester, and Stephen, from Bexley, are due to stand trial in July. They were all released on bail.</s>

Figure 5.8: Cross-entropy errors on article tokens when some summary tokens are masked. Summary is shown above article with the masked tokens marked red. High cross-entropy on article corresponds to red colors.

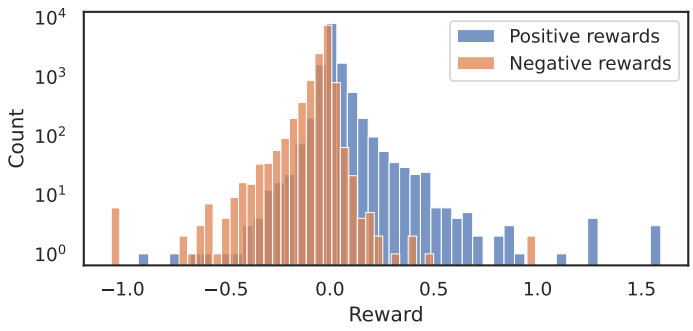


Figure 5.9: Positive and negative rewards from an independent test using a distractor summary.

6. Self-Supervised Summarization Task

This chapter combines all of the work described in previous chapters into a final set of experiments. Specifically, we use pre-trained checkpoints from Chapter 3 and attempt to improve their summarization abilities using the reinforcement learning approach from Chapter 4 with the self-supervised reward function from Chapter 5. We start with the hyperparameter configurations tuned on test tasks in previous chapters and adapt them for this specific case.

6.1 Training

During training, we monitor the reinforcement rewards and returns, KL-divergence with respect to a reference model, gradient norm, policy entropy, and summarization metrics ROUGE and BertScore. We conduct a manual hyperparameter search on the XSum dataset using BART-base, evaluating various configurations of β, γ, KL-divergence minimization strategies, masking strategies, returns normalization, learning rate values, mask tokens, `loss_reducer` strategies, and `samples_reducer` strategies (for a description of these hyperparameters refer to Chapter 4 and Chapter 5). Then we use the best found configuration to train BART-large, altering only β, learning rate, and batch size. The same configuration is then used for the CNN/Daily Mail dataset with minimal modifications.

The final hyperparameter configurations are listed in Table 6.1. For runs with these configurations, the progression of reward and reference KL-divergence on the validation set during training is shown in Figure 6.1.

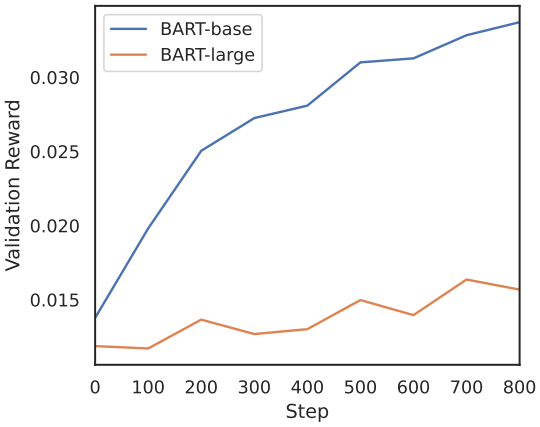
6.2 Automatic Evaluation

We evaluate the trained models using SummEval [Fabbri et al., 2020], a collection of 14 automatic evaluation metrics. Because of technical difficulties, we exclude the S^3 metric, which is a model-based metric that uses other metrics as input features. The model is unfortunately serialized using an old version of Python not available in our execution environment. More details including the installation process of SummEval are available in Appendix B.1.

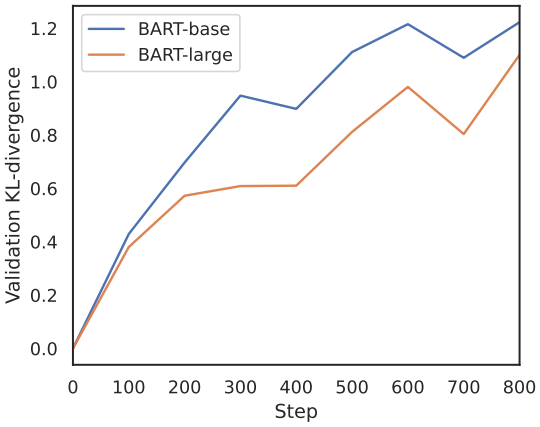
We also exclude the Sentence Mover's Similarity (SMS) as it reports the value of 1.0 for all our experiments. Since we did not modify the default configuration for this metric we suspect it might be a problem with the implementation. We do not investigate this further.

We therefore report scores on 12 automatic evaluation metrics. Following [Fabbri et al., 2020] we split these into metrics designed for summarization and general text generation metrics. We further split the former based on whether they use the gold reference into *reference-based* and *reference-free*.

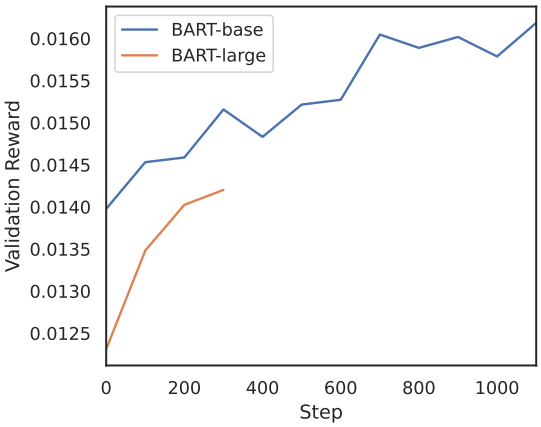
Reference-based metrics generally rely on measuring the overlap of n-grams, words, sub-sequences, or sentences between the reference and the model's output. They employ various strategies for alignment, stemming, soft comparison in embedding spaces, and



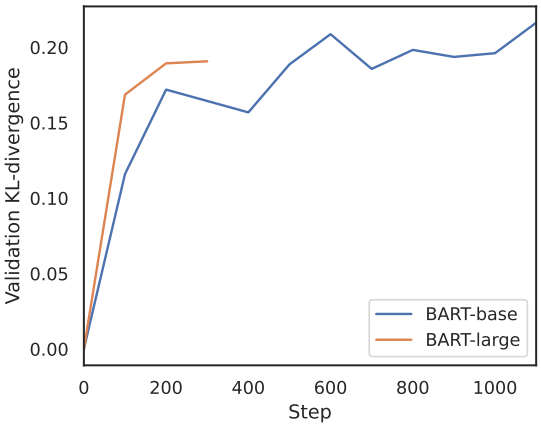
(a) Validation reward (XSum)



(b) Validation reference KL-divergence (XSum)



(c) Validation reward (CNN/DM)



(d) Validation reference KL-divergence (CNN/DM)

Figure 6.1: Validation reward and reference KL-divergence during BART training.

Hyperparameter	XSum		CNN/Daily Mail	
	BART-base	**BART-large**	**BART-base**	**BART-large**
Batch size	16	12	12	5
Learning rate	1e-5	5e-6	1e-5	1e-5
Number of updates	800	800	1100	300
KL-divergence strategy	`full_kl_loss`			
β (KL-divergence coefficient)	0.014	0.015	0.03	0.03
γ (discount factor)	0.9	0.95	0.9	0.9
ω (IDF weight power)	1.0			
Loss reducer	`average`			
Samples reducer	`min`			
Masking strategy	`words`			
Masking token	`random`			
`words_burst_length`	2			
Returns normalization	False			
Predictor	BART-base	BART-large	BART-base	BART-base
Predictor SFT masking	`random`			

Table 6.1: Final hyperparameters.

more. The problem with reference-based metrics is that even high-quality summaries might be penalized because they might be very different from the reference summary. Reference-free metrics work either by measuring the usefulness of the summary when performing language understanding tasks on the article (SummaQA [Scialom et al., 2019], BLANC [Vasilyev et al., 2020]) or by generating pseudo-reference summaries from the article, converting the problem to the reference-based case (SUPERT [Gao et al., 2020]).

The final test split scores of BART-base and BART-large trained using our approach on XSum and CNN/DailyMail are listed in Table 6.2 (marked RL). For comparison, we also include scores for the models before reinforcement learning after only supervised fine-tuning on the first 50% of the train split (marked SFT). To account for the effect of training the SFT models on just half of the data compared to the RL models, we additionally train BART on the whole train split of XSum and provide their evaluation as well (marked full-SFT). This additional training is not done in the case of CNN/Daily Mail.

Together with the human evaluation, this is the only time when the test split is used. All experiments performed on the test split are reported.

During evaluation, we utilize the `length_penalty`[1] parameter of the Transformers library to encourage some models to generate shorter output sequences. This is important because some metrics are incomparable across models if the average output length differs considerably. Specifically, for the marked models we use a `length_penalty` of -2. This means that during beam search, the score (i.e. log-probability) of a sequence of length n is divided by n^{-2}. Since these scores are negative, this penalizes longer sequences. This technique is enabled by default when using BART in Transformer's summarization pipeline on CNN/Daily Mail.[2]

Afterward, we experimented with the `exponential_decay_length_penalty` param-

[1]The naming of this parameter is unfortunate since positive values encourage longer sequences and negative values encourage shorter sequences.

[2]This can be verified by inspecting the `task_specific_params` field of BART's config in Transformers.

Model	*Reference-based metrics*				*Reference-free metrics*		
	ROUGE-1/2/3/4/L/su*/w	**ROUGE-WE-3**	**BertScore**	**MoverScore**	**SummaQA**	**BLANC**	**SUPERT**
XSum							
base SFT	0.410/0.182/0.096/0.056/0.331/0.173/0.197	0.229	0.429	0.247	0.059	0.025	0.454
base full-SFT	0.427/0.198/0.109/0.064/0.347/0.187/0.206	0.244	0.446	0.267	0.059	0.025	0.454
base RL	0.333/0.119/0.053/0.027/0.258/0.115/0.155	0.158	0.308	0.124	**0.097**	**0.057**	**0.494**
large SFT	**0.445**/**0.212**/**0.119**/**0.071**/**0.363**/**0.202**/**0.216**	**0.261**	**0.467**	**0.291**	0.058	0.025	0.454
large full-sft	**0.458**/**0.225**/**0.129**/**0.080**/**0.375**/**0.213**/**0.224**	**0.274**	**0.480**	**0.307**	0.057	0.025	0.453
large RL *	0.379/0.152/0.075/0.041/0.297/0.145/0.179	0.197	0.376	0.193	**0.084**	**0.054**	**0.493**
CNN/Daily Mail							
base SFT	**0.412**/**0.188**/**0.109**/0.073/**0.384**/**0.158**/**0.211**	**0.228**	**0.404**	**0.197**	0.146	0.099	0.626
base RL *	0.391/0.172/0.099/0.066/0.363/0.143/0.201	0.209	0.378	0.172	**0.158**	**0.112**	0.643
base RL †	0.391/0.172/0.099/0.066/0.363/0.143/0.200	0.210	0.377	0.173	0.154	0.108	0.642
large SFT	**0.417**/**0.191**/**0.111**/**0.074**/**0.384**/**0.160**/**0.213**	**0.232**	**0.405**	**0.198**	0.150	0.103	**0.649**
large RL *	0.411/0.187/**0.109**/**0.074**/0.375/0.154/**0.211**	0.227	0.395	0.188	**0.170**	**0.121**	**0.690**
large RL †	0.411/0.184/0.107/0.072/0.374/0.153/0.209	0.224	0.389	0.187	0.162	0.114	0.681

* Evaluated using `length_penalty` $= -2$.

† Evaluated using `exponential_decay_length_penalty` $= (55, 0.9)$.

(a) Model scores from summarization-specific evaluation metrics.

Model	BLEU	chrF	CIDEr	METEOR	Length	Stats (cov/comp/den)	Repeated (1/2/3)	Novelty (1/2/3)
XSum								
base SFT	12.015	36.182	1.291	0.184	**21.228**	0.763/**20.916**/1.637	**0.068**/0.003/0.000	0.250/0.720/0.883
base full-SFT	13.482	37.775	1.433	0.194	**21.663**	0.756/**20.530**/1.602	0.069/0.003/0.000	0.258/0.729/0.889
base RL	7.584	31.730	0.836	0.151	25.113	**0.890**/17.996/**3.731**	0.089/0.004/0.000	0.119/0.482/0.667
large SFT	**14.692**	**39.373**	**1.512**	**0.206**	22.260	0.748/20.030/1.549	0.069/0.003/0.000	**0.267**/**0.741**/**0.897**
large full-SFT	**15.860**	**40.617**	**1.637**	**0.214**	22.419	0.743/19.898/1.526	**0.068**/0.003/0.000	**0.272**/**0.745**/**0.899**
large RL *	9.699	35.442	1.019	0.177	25.808	**0.864**/17.601/**4.452**	0.083/0.003/0.000	0.145/0.492/0.652
CNN/Daily Mail								
base SFT	**15.378**	39.348	**0.705**	0.189	**56.986**	0.957/**14.272**/12.900	0.183/0.025/0.000	**0.031**/**0.139**/**0.241**
base RL *	14.222	39.478	0.552	0.186	64.216	0.963/13.107/16.397	0.176/0.023/0.000	0.024/0.097/0.171
base RL †	14.379	38.733	0.617	0.181	60.519	0.963/13.261/16.586	0.175/0.023/0.000	0.025/0.095/0.167
large SFT	**15.753**	**40.231**	**0.647**	**0.194**	**59.516**	**0.966**/**14.206**/**18.316**	**0.170**/**0.018**/0.000	**0.030**/**0.109**/**0.183**
large RL *	14.595	**41.483**	0.482	**0.201**	68.693	**0.977**/12.917/**26.105**	**0.172**/**0.016**/0.000	0.020/0.051/0.088
large RL †	14.916	40.226	0.605	0.190	62.086	0.976/12.928/25.873	0.165/0.015/0.000	0.022/0.051/0.087

* Evaluated using `length_penalty` $= -2$.

† Evaluated using `exponential_decay_length_penalty` $= (55, 0.9)$.

(b) Model scores from general text generation evaluation metrics.

Table 6.2: Final model scores on 12 SummEval metrics. The two highest scores on each dataset for each metric (lowest for Length and Repeated-1/2/3) are bolded, ignoring the models marked with † and cases where the values are too small to be distinguishable. BART-base is listed as *base* and BART-large as *large*.

eter in Transformers, achieving an average output length even more similar to the SFT models. We include these for completeness.

All other generation parameters are the same as in Section 3.2.

6.2.1 Discussion of the Automatic Results

Table 6.2 shows the final test scores of relevant checkpoints on 12 summarization metrics. The main trend to observe is that our self-supervised approach (models marked RL) improves scores on reference-free metrics in almost all cases. This holds even when compared to the models trained in a supervised fashion on the whole train split (marked full-SFT), offsetting the fact that the SFT models saw only half of the data.

The success on SummaQA and BLANC is somewhat to be expected since these metrics measure language understanding on an article based on a summary, which is similar to our approach. This is not the case for the SUPERT metric which measures the semantic similarity of a summary with a pseudo-reference summary created by selecting salient sentences from the article.

The performance improvement on reference-free metrics is accompanied by a decline on all reference-based metrics. This is to be expected since the self-learning objective does not encourage adhering to the reference summary (it only attempts to not deviate too much from the outputs produced after supervised fine-tuning through the use of reference KL-divergence).

Compared to the scores reported on BART[3] by Fabbri et al. [2020], our checkpoints after supervised fine-tuning on CNN/Daily Mail achieve comparable performance.[4] The only discrepancy is the Repeated-3 metric which is consistently indistinguishable from zero in our case. We suspect there is a problem in the reporting of Repeated-3 by Fabbri et al. [2020] since it is exactly equal to Repeated-2 in all cases.

Overall, these results show that the self-supervised learning improved some of the summarization capabilities of the fine-tuned models.

6.2.2 Discussion of Specific Examples

Table 6.3 lists generated summaries on the first five validation samples from XSum. The top half of the table contains outputs from BART-base fine-tuned in a supervised fashion (experiment BART-base SFT). The bottom half contains outputs generated after our self-supervised approach was applied on top of the supervised fine-tuning (experiment BART-base RL). In this section, we discuss each of these samples. For full context, refer to the corresponding articles listed, for example, on HuggingFace.[5]

This section is meant as a small qualitative example of the effect that our self-learning approach has on the generated summaries but we will not attempt to make any general conclusions from only five data points. Such analysis will be the subject of Section 6.3.

The following list contains comments for summaries in Table 6.3 generated after supervised fine-tuning, marked (SFT), and summaries generated after self-learning, marked (RL).

(A) (SFT) The summary is coherent and relevant. However, the article contains no mention of the £1m figure (nor is it mentioned in connection to Sam Sodje in the XSum train data) and it mentions that the defendants were released on bail (not remanded in custody).

(RL) The summary correctly states that there were actually four actors (brothers), that they are from Kent, Greater Manchester, and Bexley, that they denied charges, that the charges relate to fraudulent trading, and also the name of the Sodje Sports Foundation. However, it lists the name Efe twice, although the fourth brother is named Stephen, hurting readability. It also makes no mention of the court.

(B) (SFT) The summary correctly captures the main point of the article. However, the article does not mention the type of injury suffered (hamstring) and neither the fact that Adam Voges is a batsman, although that could be considered common knowledge. Moreover, the article states that the team hopes to have Adam Voges back in August, not that he will be out for the rest of the season.

(RL) The only difference from (SFT) is the replacement of "hamstring" with "calf", which is equally inconsistent with the article.

[3]Fabbri et al. [2020] do not specify whether they use BART-base or BART-large.

[4]The chrF metric [Popović, 2015] is reported in percentages instead of in the range $[0, 1]$ in our case.

[5]`https://huggingface.co/datasets/EdinburghNLP/xsum/viewer/default/validation`

BART-base after supervised fine-tuning	
(A)	Sam Sodje has been remanded in custody after appearing in court accused of embezzling £1m from a sports charity in Nigeria.
(B)	Middlesex batsman Adam Voges has been ruled out for the rest of the season with a hamstring injury.
(C)	The Duchess of Cambridge has been photographed for the first time in British Vogue's 100th anniversary issue.
(D)	Google has appointed the former administrator of notorious social networking site 4chan to lead its social networking efforts.
(E)	Three people have been charged in connection with an aggravated vehicle theft in north Belfast.
BART-base after 800 RL updates	
(A)	Sam, from Kent, Efe and Bright, of Greater Manchester, and Efe Sodje, of Bexley, have denied fraudulent trading charges relating to the Sodje Sports Foundation.
(B)	Middlesex batsman Adam Voges has been ruled out for the rest of the season with a calf injury.
(C)	The Duchess of Cambridge has been photographed in a photographic shoot for British Vogue 100: A Century of Style.
(D)	Chris Poole, the administrator of 4chan, has joined Google's social networking site, Bradley Horowitz.
(E)	A man, aged 19, and a boy, aged 16, have been charged with six counts of aggravated vehicle taking in Belfast.

Table 6.3: The first five samples on XSum validation set. At the top are outputs after supervised fine-tuning. At the bottom are outputs after 800 updates of self-learning.

(C) (SFT) The summary captures the main point of the article and contains no serious errors. However, as per the article, the duchess was not photographed for the first time ever, rather it was the first time she did so for a magazine.

(RL) Compared to (SFT), the summary specifies the name "British Vogue 100: A Century of Style". According to the article, this is the name of an exhibition where the photos will also be displayed and which collaborated with Vogue on the photographic shoot. However, the formulation in the summary is misleading, because the photos were primarily taken for the magazine. Furthermore, readability is hurt by the redundant repetition in the phrase "photographed in a photographic shoot".

(D) (SFT) The summary captures the main point of the article but lacks detail. Moreover, it is inconsistent with the article in that the article does not mention that the person in question will have a leadership position.

(RL) Compared to (SFT), the summary adds the piece of information that the person in question is named Chris Poole (which is correct as per the article) and discards the inconsistent piece of information about leadership. However, it lacks coherence because of the odd mention of Bradley Horowitz, whom the article quotes welcoming Chris Poole to Google. It is also inconsistent to call

Chris Poole "the administrator of 4chan" because the article mentions that he has already stepped down.

(E) (SFT) The summary captures the basic message of the article but contains factual errors. As per the article, two people have been involved in the incident (not three). Moreover, the article does not mention north Belfast, but rather only the Belfast Magistrates' Court. It is debatable whether that implies that the incident occurred in Belfast.

(RL) Compared to (SFT), the summary correctly lists two actors together with their age. It also adds the correct information that there were six counts of theft. Furthermore, it discards the incorrect adjective "north" when talking about Belfast.

6.3 Human Evaluation

We conduct a human evaluation of the generated summaries on the first 50 test set samples from XSum using our custom browser application built with Streamlit. A screenshot of this application is shown in Figure 6.4. The user is presented with general instructions, a definition for each evaluated metric, an XSum article, and five different summaries generated from the evaluated models. The generation was done in the same way as in Section 6.2. Crucially, the presented summaries are shuffled randomly for each example so that the user cannot tell which one was generated from which model.

We take heavy inspiration from Fabbri et al. [2020] in designing the axes on which summaries are evaluated. However, we discard their *Coherence* metric which measures inter-sentence coherence since XSum summaries mostly consist of just one sentence. In the end, we use the following three metrics:

- *relevance* – measures how well the summary captures key points of the article,
- *consistency* – measures whether the facts in the summary are consistent with the facts in the original article,
- *readability* – measures the grammatical, stylistic, and logical quality of the summary when read by itself (without the original article).

In statistical terms, *relevance* can be roughly thought of as recall and *consistency* as precision.

The whole evaluation was performed by just one annotator (the author of this thesis). The resulting average scores for BART-base and BART-large after supervised fine-tuning and after self-learning are listed in Table 6.4 and plotted in Figure 6.2. In addition to the trained models, we include the score for reference summaries from XSum under the label *reference*. For a better insight into the distribution of scores, we plot the individual scores on relevance and consistency from human evaluation in Figure 6.3.

The results suggest that the self-supervised training improved the amount of relevant information included in the summaries (*relevance*) as well as its accuracy (*consistency*). At the same time, it did not induce any detriment in readability. Another trend to observe is that BART-large consistently gives better scores than BART-base. However, to formulate a definitive conclusion on any of these trends, we would need more annotators evaluating more samples.

Model	Relevance	Consistency	Readability
BART-base SFT	3.54	3.42	4.62
BART-base RL	**3.92**	**3.56**	4.70
BART-large SFT	3.68	3.54	4.64
BART-large RL	**3.98**	**3.94**	**4.84**
reference	3.70	3.46	**4.80**

Table 6.4: Average scores from human evaluation. The two highest scores on each metric are bolded.

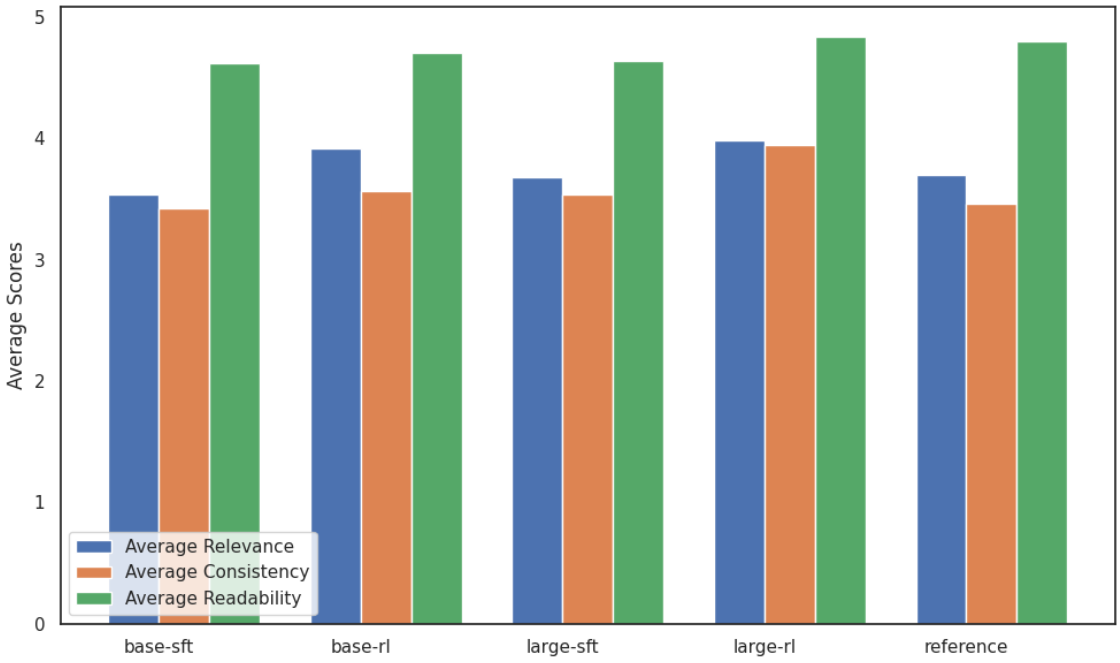
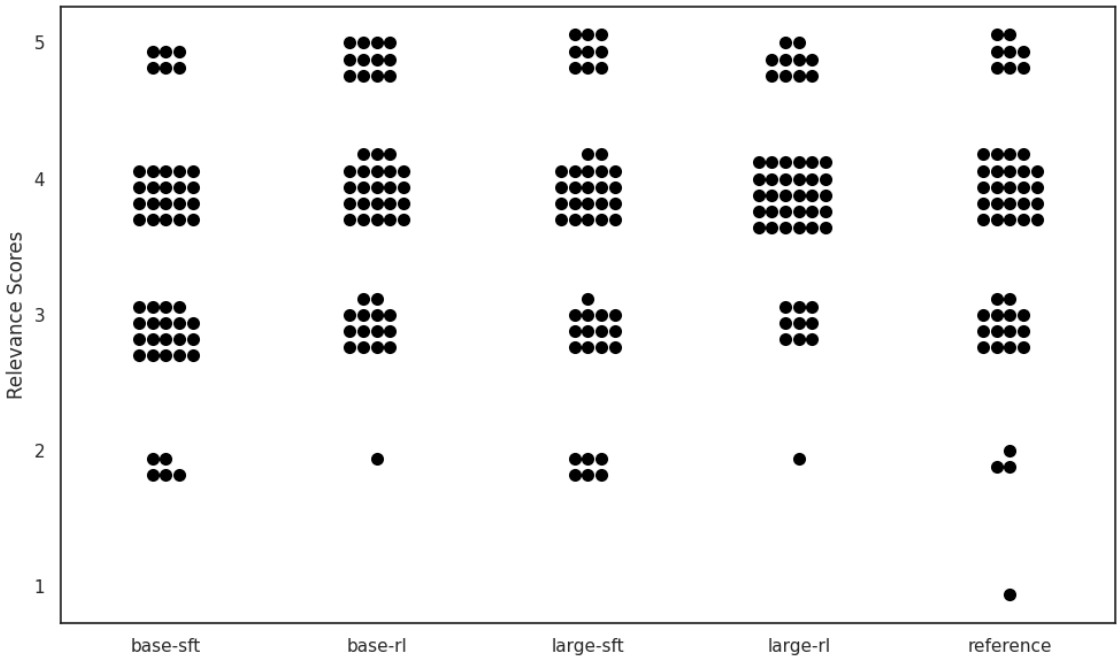


Figure 6.2: Average scores from human evaluation.

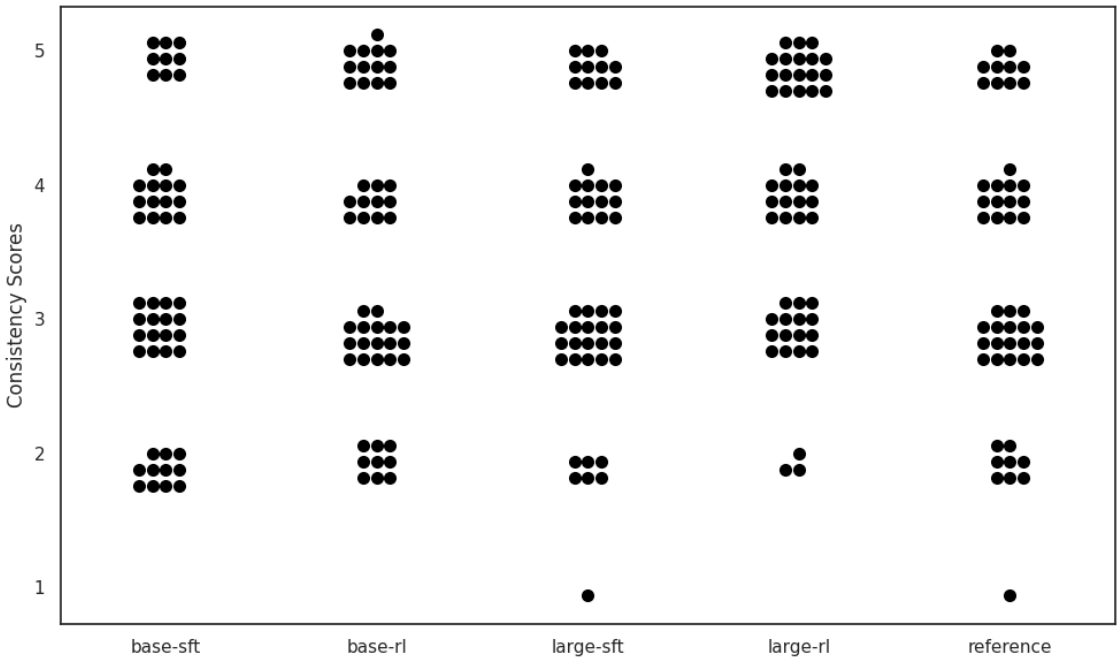
Interestingly, the reference summaries from XSum score relatively low on consistency and also on relevance. After inspecting the samples and scores qualitatively,[6] we believe this is due to the way XSum was created which we already alluded to in Section 5.3. Specifically, the summary was taken to be the first sentence of a BBC news article and was subsequently removed from the article. However, some crucial information is often only contained in this first sentence and therefore the XSum article does not entail the XSum summary. This leads to the reference summary being marked low on consistency. It also means that some parts of the summary might not be relevant with respect to the article, causing a low score on relevance.

Considering these shortcomings of the XSum reference summaries, part of the value of a self-supervised approach is unshackling the models from having to maximize their likelihood.

[6]This analysis was only done after all the scores were recorded.



(a) Relevance scores from human evaluation.



(b) Consistency scores from human evaluation.

Figure 6.3: Individual scores from human evaluation displayed as a scatter plot. Overlapping dots are moved to remove the overlap.

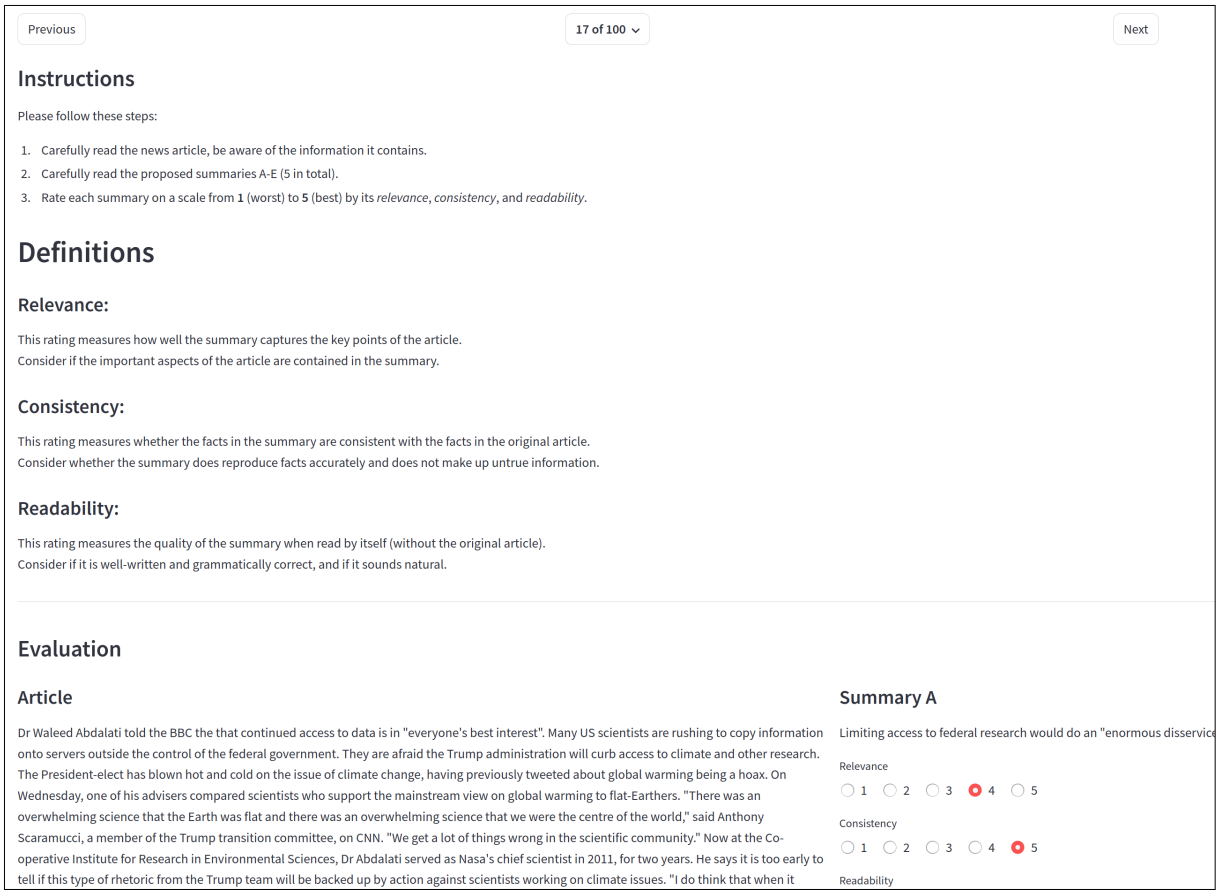


Figure 6.4: Screenshot of the browser application used for human evaluation (cropped).

Conclusion

We introduced a novel dense reference-free reward function for summarization. Its core idea is to estimate the importance of summary tokens based on the impact that masking them has on the prediction of the original article. We then used the reward function for reinforcement learning, improving some summarization capabilities of BART. This improvement was observed both on reference-free automatic metrics and in human evaluation.

Because of the complexity of the final self-learning pipeline, we designed and tested its components in separation. First, we obtained baseline checkpoints via supervised fine-tuning. Then we implemented and tested the reinforcement learning algorithm using an artificial task of maximizing the number of output words that start with the letter `T`. The reward function for summarization was also designed separately, choosing from a number of different strategies and hyperparameters using two custom automatic metrics. Finally, we connected all the pieces together for self-supervised training of BART.

We evaluated the final checkpoints on the test set using 12 automatic metrics from the SummEval package and 3 manual metrics. The results indicate that the self-supervised approach causes models to less adhere to the reference summaries and improves their performance on all reference-free metrics in nearly all cases. We believe that this validates the viability of our approach and opens the door to future research.

6.4 Future Work

There remains a lot of work to be done in the area of summarization via reinforcement learning, including building on the results of this thesis. In this section, we highlight some of the possible avenues of future work.

Reward Function Improvement and Evaluation

While the general idea of approximating the importance of a phrase by masking it and observing the increase of predictor perplexity is simple, there are numerous implementation choices with non-trivial interdependence. Many of them remain unexplored.

For example, instead of masking phrases by replacing them with random tokens, the masking could be done by replacing words with other words that are close in a semantic contextual embedding such as one obtained from BERT [Devlin et al., 2018]. This would be to address the issue introduced in Section 5.3.2 where an incorrect piece of information might still improve predictor performance because it is in some way related to the correct information. The example we gave was that masking the mention of Inverness hurts predictor performance even though the article is about Dundee because both are cities in Scotland. If instead of completely obscuring "Inverness" it was replaced with e.g. Dunfermline (another city in Scotland), the perplexity increase could be negated, eliminating the positive reward on "Inverness". This is because presumably "Dunfermline" is as helpful as "Inverness" when the actual city is Dundee.

Our reinforcement learning implementation might also be tested with an entirely different reward function architecture, for example, the Attention Based Credit introduced by Chan et al. [2024].

Besides the reward function design, there is room for improvement in the evaluation procedure of reward functions. We introduced two evaluation metrics in Section 5.2. Future work might empirically explore their correlation with human judgement regarding the ability of reward function to recognize important summary tokens.

Reinforcement Learning Improvements

Regarding the reinforcement learning logic, we only implemented the REINFORCE algorithm. This leaves room for many future improvements, most notably implementing the Proximal Policy Optimization (PPO) algorithm [Schulman et al., 2017]. Even with REINFORCE, future work might use a value network computing state values for better baseline estimation.

Other Training Strategies

We only explored the *in-domain* self-learning setting where both the supervised fine-tuning and the self-learning were done on different parts of the same dataset. This leaves for future work the *out-of-domain* setting where the self-learning might utilize unlabeled articles from an entirely different data source, thereby also casting our approach as a domain adaptation technique.

Usage as a Summarization Metric

While we employed our reward function only as a training signal for reinforcement learning, it could also be used as a basis for a summarization metric. For this, it is necessary to define a consistent way of reducing the dense per-token reward to a scalar per-sequence reward. The resulting metric should then be rigorously tested with human evaluation.

One benefit of this metric is that it would provide detailed per-token scores for better interpretability.

Work on the Open-Source Ecosystem

During our work, we found that many useful open-source libraries in the field of summarization and reinforcement learning are no longer maintained. This makes them unnecessarily difficult to install and unable to capture recent trends in these fields. Most importantly, this is the case for RL4MLs and SummEval. We created pull requests to both of these libraries to make them easier to install, but there remains the need for more thorough refactoring and an active community of maintainers.

Bibliography

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

Riad Akrour, Marc Schoenauer, and Michele Sebag. Preference-based policy learning. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2011, Athens, Greece, September 5-9, 2011. Proceedings, Part I 11*, pages 12–27. Springer, 2011.

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. *Advances in neural information processing systems*, 30, 2017.

Jason Ansel, Edward Yang, Horace He, Natalia Gimelshein, Animesh Jain, Michael Voznesensky, Bin Bao, Peter Bell, David Berard, Evgeni Burovski, Geeta Chauhan, Anjali Chourdia, Will Constable, Alban Desmaison, Zachary DeVito, Elias Ellison, Will Feng, Jiong Gong, Michael Gschwind, Brian Hirsh, Sherlock Huang, Kshiteej Kalambarkar, Laurent Kirsch, Michael Lazos, Mario Lezcano, Yanbo Liang, Jason Liang, Yinghai Lu, CK Luk, Bert Maher, Yunjie Pan, Christian Puhrsch, Matthias Reso, Mark Saroufim, Marcos Yukio Siraichi, Helen Suk, Michael Suo, Phil Tillet, Eikan Wang, Xiaodong Wang, William Wen, Shunting Zhang, Xu Zhao, Keren Zhou, Richard Zou, Ajit Mathews, Gregory Chanan, Peng Wu, and Soumith Chintala. PyTorch 2: Faster machine learning through dynamic Python bytecode transformation and graph compilation. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '24)*. ACM, 4 2024. doi: 10.1145/3620665.3640366. URL `https://pytorch.org/assets/pytorch2-2.pdf`.

Kristjan Arumae and Fei Liu. Guiding extractive summarization with question-answering rewards. *arXiv preprint arXiv:1904.02321*, 2019.

Yuntao Bai, Andy Jones, Kamal Ndousse, Amanda Askell, Anna Chen, Nova DasSarma, Dawn Drain, Stanislav Fort, Deep Ganguli, Tom Henighan, et al. Training a helpful and harmless assistant with reinforcement learning from human feedback. *arXiv preprint arXiv:2204.05862*, 2022a.

Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, et al. Constitutional AI: Harmlessness from AI feedback. *arXiv preprint arXiv:2212.08073*, 2022b.

Richard Bellman. A markovian decision process. *Journal of mathematics and mechanics*, pages 679–684, 1957.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Meng Cao, Lei Shu, Lei Yu, Yun Zhu, Nevan Wichers, Yinxiao Liu, and Lei Meng. Beyond sparse rewards: Enhancing reinforcement learning with language model critique in text generation. *arXiv preprint arXiv:2401.07382*, 2024.

Thomas Carta, Clément Romac, Thomas Wolf, Sylvain Lamprier, Olivier Sigaud, and Pierre-Yves Oudeyer. Grounding large language models in interactive environments with online reinforcement learning. In *International Conference on Machine Learning*, pages 3676–3713. PMLR, 2023.

Alex J Chan, Hao Sun, Samuel Holt, and Mihaela van der Schaar. Dense reward for free in reinforcement learning from human feedback. *arXiv preprint arXiv:2402.00782*, 2024.

Ping Chen, Fei Wu, Tong Wang, and Wei Ding. A semantic qa-based approach for text summarization evaluation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174*, 2016.

Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. *Advances in neural information processing systems*, 30, 2017.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Matan Eyal, Tal Baumel, and Michael Elhadad. Question answering as an automatic evaluation metric for news article summarization. *arXiv preprint arXiv:1906.00318*, 2019.

Alexander R Fabbri, Wojciech Kryściński, Bryan McCann, Caiming Xiong, Richard Socher, and Dragomir Radev. SummEval: Re-evaluating summarization evaluation. *arXiv preprint arXiv:2007.12626*, 2020.

Kavita Ganesan. ROUGE 2.0: Updated and improved measures for evaluation of summarization tasks. *arXiv preprint arXiv:1803.01937*, 2018.

Yang Gao, Wei Zhao, and Steffen Eger. SUPERT: Towards new frontiers in unsupervised evaluation metrics for multi-document summarization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1347–1354, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.124. URL `https://aclanthology.org/2020.acl-main.124`.

Tirthankar Ghosal, Ondřej Bojar, Marie Hledíková, Tom Kocmi, and Anna Nedoluzhko. Overview of the second shared task on automatic minuting (AutoMin) at INLG 2023. In *Proceedings of the 16th International Natural Language Generation Conference: Generation Challenges*, pages 138–167, 2023.

Amelia Glaese, Nat McAleese, Maja Trębacz, John Aslanides, Vlad Firoiu, Timo Ewalds, Maribeth Rauh, Laura Weidinger, Martin Chadwick, Phoebe Thacker, et al. Improving alignment of dialogue agents via targeted human judgements. *arXiv preprint arXiv:2209.14375*, 2022.

Tanya Goyal, Junyi Jessy Li, and Greg Durrett. News summarization and evaluation in the era of GPT-3. *arXiv preprint arXiv:2209.12356*, 2022.

Alexander Havrilla, Maksym Zhuravinskyi, Duy Phung, Aman Tiwari, Jonathan Tow, Stella Biderman, Quentin Anthony, and Louis Castricato. trlX: A framework for large scale reinforcement learning from human feedback. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 8578–8595, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.530. URL https://aclanthology.org/2023.emnlp-main.530.

Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (GELUs). *arXiv preprint arXiv:1606.08415*, 2016.

Karl Moritz Hermann, Tomás Kociský, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching machines to read and comprehend. *CoRR*, abs/1506.03340, 2015. URL http://arxiv.org/abs/1506.03340.

Natasha Jaques, Asma Ghandeharioun, Judy Hanwen Shen, Craig Ferguson, Agata Lapedriza, Noah Jones, Shixiang Gu, and Rosalind Picard. Way off-policy batch deep reinforcement learning of implicit human preferences in dialog. *arXiv preprint arXiv:1907.00456*, 2019.

Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

Harrison Lee, Samrat Phatale, Hassan Mansoor, Kellie Lu, Thomas Mesnard, Colton Bishop, Victor Carbune, and Abhinav Rastogi. RLAIF: Scaling reinforcement learning from human feedback with AI feedback. *arXiv preprint arXiv:2309.00267*, 2023.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. BART: denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *CoRR*, abs/1910.13461, 2019. URL http://arxiv.org/abs/1910.13461.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. *arXiv preprint arXiv:2305.20050*, 2023.

Chin-Yew Lin. ROUGE: A package for automatic evaluation of summaries. In *Text summarization branches out*, pages 74–81, 2004.

Elena Lloret and Manuel Palomar. Text summarisation in progress: a literature review. *Artificial Intelligence Review*, 37:1–41, 2012.

Hans Peter Luhn. The automatic creation of literature abstracts. *IBM Journal of research and development*, 2(2):159–165, 1958.

Ramesh Nallapati, Bing Xiang, and Bowen Zhou. Sequence-to-sequence RNNs for text summarization. *CoRR*, abs/1602.06023, 2016. URL `http://arxiv.org/abs/1602.06023`.

Shashi Narayan, Shay B. Cohen, and Mirella Lapata. Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization. volume abs/1808.08745, 2018.

OpenAI. GPT-3.5 Turbo. `https://platform.openai.com/docs/models/gpt-3-5`, 2022a. Accessed: 2024-04-19.

OpenAI. Introducing ChatGPT. `https://openai.com/blog/chatgpt`, 2022b. Accessed: 2024-04-09.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744, 2022.

Ramakanth Pasunuru and Mohit Bansal. Multi-reward reinforced summarization with saliency and entailment. *arXiv preprint arXiv:1804.06451*, 2018.

Romain Paulus, Caiming Xiong, and Richard Socher. A deep reinforced model for abstractive summarization. *arXiv preprint arXiv:1705.04304*, 2017.

Maja Popović. chrf: character n-gram f-score for automatic mt evaluation. In *Proceedings of the tenth workshop on statistical machine translation*, pages 392–395, 2015.

M.F. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, Jul 1980. URL `https://tartarus.org/martin/PorterStemmer/def.txt`. Accessed: 2024-04-09.

Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140): 1–67, 2020.

Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-Baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021. URL `http://jmlr.org/papers/v22/20-1364.html`.

Rajkumar Ramamurthy, Prithviraj Ammanabrolu, Kianté Brantley, Jack Hessel, Rafet Sifa, Christian Bauckhage, Hannaneh Hajishirzi, and Yejin Choi. Is reinforcement learning (not) for natural language processing?: Benchmarks, baselines, and building

blocks for natural language policy optimization. 2022. URL https://arxiv.org/abs/2210.01241.

Steven J Rennie, Etienne Marcheret, Youssef Mroueh, Jerret Ross, and Vaibhava Goel. Self-critical sequence training for image captioning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7008–7024, 2017.

Natalie Schluter. The limits of automatic summarisation according to ROUGE. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics*, pages 41–45. Association for Computational Linguistics, 2017.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Thomas Scialom, Sylvain Lamprier, Benjamin Piwowarski, and Jacopo Staiano. Answers unite! unsupervised metrics for reinforced summarization models. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan, editors, *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3246–3256, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1320. URL https://aclanthology.org/D19-1320.

Abigail See, Peter J. Liu, and Christopher D. Manning. Get to the point: Summarization with pointer-generator networks. *CoRR*, abs/1704.04368, 2017. URL http://arxiv.org/abs/1704.04368.

Noam Shazeer and Mitchell Stern. Adafactor: Adaptive learning rates with sublinear memory cost. In *International Conference on Machine Learning*, pages 4596–4604. PMLR, 2018.

Nisan Stiennon, Long Ouyang, Jeffrey Wu, Daniel Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul F Christiano. Learning to summarize with human feedback. *Advances in Neural Information Processing Systems*, 33:3008–3021, 2020.

Yixuan Su, Tian Lan, Yan Wang, Dani Yogatama, Lingpeng Kong, and Nigel Collier. A contrastive framework for neural text generation. *Advances in Neural Information Processing Systems*, 35:21548–21561, 2022.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

Wilson L Taylor. "cloze procedure": A new tool for measuring readability. *Journalism quarterly*, 30(4):415–433, 1953.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

Oleg Vasilyev, Vedant Dharnidharka, and John Bohannon. Fill in the BLANC: Human-free quality estimation of document summaries. *arXiv preprint arXiv:2002.09836*, 2020.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Leandro von Werra, Younes Belkada, Lewis Tunstall, Edward Beeching, Tristan Thrush, Nathan Lambert, and Shengyi Huang. TRL: transformer reinforcement learning. `https://github.com/huggingface/trl`, 2020.

Ronald J. Williams. A class of gradient-estimating algorithms for reinforcement learning in neural networks. In *Proceedings of the IEEE First International Conference on Neural Networks*, San Diego, CA, 1987.

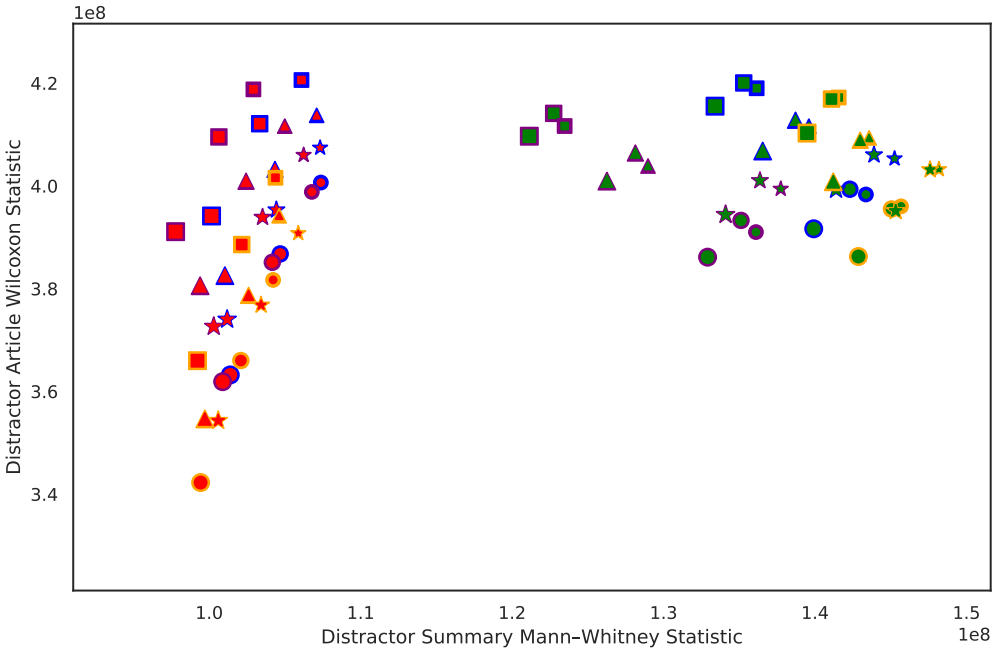
Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3):229–256, 1992. ISSN 1573-0565. doi: 10.1007/BF00992696. URL `https://doi.org/10.1007/BF00992696`.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Perric Cistac, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, October 2020. Association for Computational Linguistics. URL `https://www.aclweb.org/anthology/2020.emnlp-demos.6`.

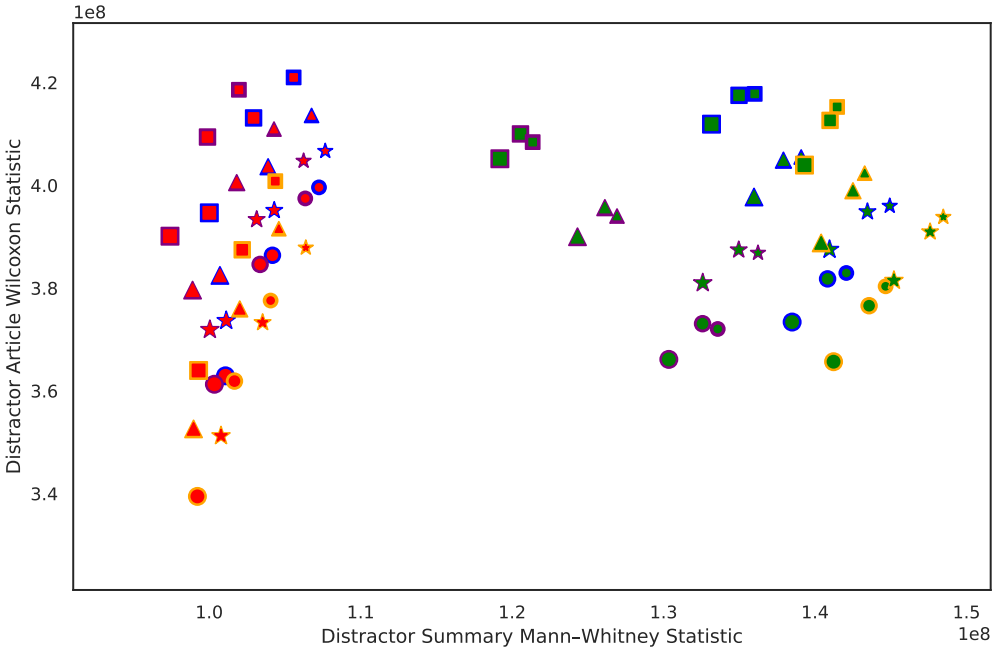
Zeqiu Wu, Yushi Hu, Weijia Shi, Nouha Dziri, Alane Suhr, Prithviraj Ammanabrolu, Noah A Smith, Mari Ostendorf, and Hannaneh Hajishirzi. Fine-grained human feedback gives better rewards for language model training. *Advances in Neural Information Processing Systems*, 36, 2024.

Daniel M Ziegler, Nisan Stiennon, Jeffrey Wu, Tom B Brown, Alec Radford, Dario Amodei, Paul Christiano, and Geoffrey Irving. Fine-tuning language models from human preferences. *arXiv preprint arXiv:1909.08593*, 2019.

A. Grid Search Results

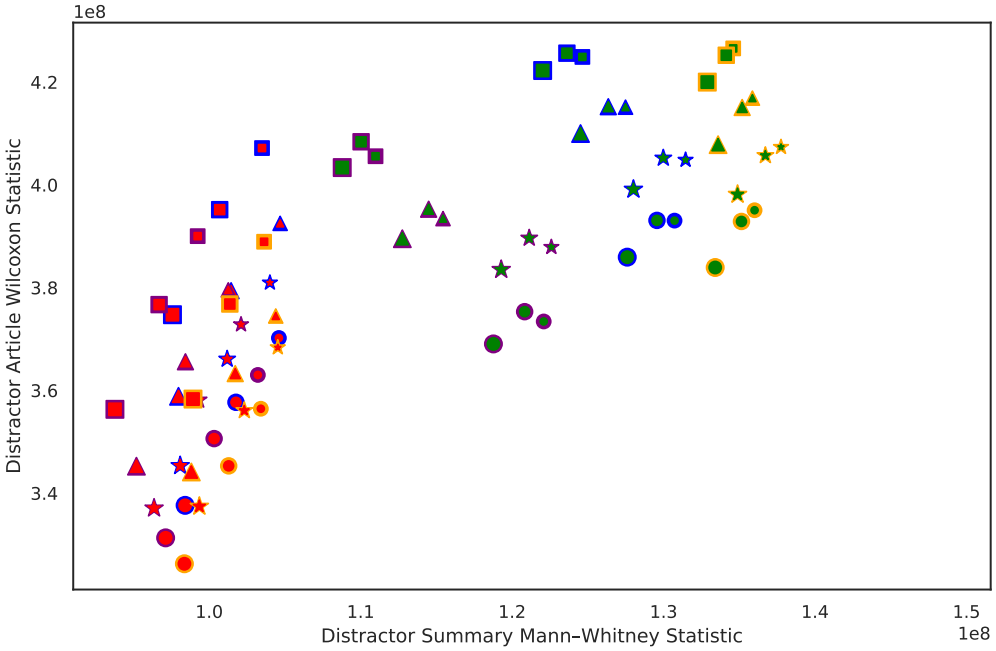


(a) `mask_token` = `<mask>`, predictor = `predictor_base-random`



(b) `mask_token` = `<mask>`, predictor = `predictor_base-mask`

Figure A.1: Paired and independent test statistics of grid search results.



(c) `mask_token` = `<random>`, predictor = `predictor_base-mask`

Figure A.1: Paired and independent test statistics of grid search results (cont.)

B. Technical Details

B.1 SummEval Package Installation

We use the official Python package released together with the SummEval paper [Fabbri et al., 2020] available at `https://github.com/Yale-LILY/SummEval`. Along with standard installation using `pip` we found the following steps to be necessary. Note that they are not listed in the package's `setup.py` file and thus need to be executed manually.

- Install the `pyrouge` package from `https://github.com/bheinzerling/pyrouge.git`.
- Install the `wmd` package available at PyPi.
- Install the `scikit-learn` in the exact version `0.21.3`. The version is important because of the S^3 metric since its associated model cannot be loaded using newer versions. Unfortunately, this package version is not compatible with Python versions newer than 3.7. Since our environment only offers Python versions from 3.8 up we decided to skip this step and not use the S^3 metric.
- Download NLTK data with the following command:
  `python -c "import nltk; nltk.download('stopwords')"`
- Modify the following environment variables. We assume that `<summ_eval>` is the absolute path where the SummEval repository was cloned.
  - `ROUGE_HOME=<summ_eval>/evaluation/summ_eval/ROUGE-1.5.5/`
  - `CORENLP_HOME=<summ_eval>/evaluation/summ_eval/stanford-corenlp-full-2018-10-05/`
  - `PYTHONPATH=$PYTHONPATH:<summ_eval>/evaluation/summ_eval`

The installation unfortunately constraints many packages to their old versions, notably forcing `torch` version `1.13`. This is incompatible with the rest of our codebase. For this reason, we set up SummEval in a separate Python virtual environment and run the metric calculation as a separate process, using inter-process communication to exchange JSON messages between the environments. To this end, we modify the package's `calc_scores.py` to simplify its interface and make it produce machine-readable output.

Along with these steps we had to make the following modifications to the codebase. We create pull requests back to the original repository for both of these issues.

- In `sentence_movers_utils.py` and `calc_scores.py`, substitute all occurrences of `stderr` with `sys.stderr`.
- In `rouge_we_metric.py` on line 17, use the HTTPS protocol instead of HTTP. Specifically, substitute `http://u.cs.biu.ac.il/` with `https://u.cs.biu.ac.il/`. Without this modification, the download fails on timeout.

We created a Dockerfile which automatically performs all the steps listed.

The metric calculation is then controlled using a Gin[1] configuration file. We use the provided default configuration, modifying only batch sizes.

[1] `https://github.com/google/gin-config`

C. Contents of Electronic Attachment

- code.zip
  - src - source codes for the whole training pipeline and experiments
    - train.py
    - eval.py
    - rewards.py
    - masks.py
    - dataset.py
    - . . .
    - summ_eval_wrapper
      - calc_scores.py - modified version of the SummEval entry script
    - config - configuration files for SummEval
      - summ_eval.config
      - summ_eval-small_bs.config
    - experiments - various stand-alone experiments and visualizations
      - predictor_viz.py
      - grid_search.py
      - dataset_stats.py
      - . . .
  - human_eval
    - human_eval.py - processing and visualization of human evaluation data
    - human_eval_app.py - browser application for human evaluation
    - summaries.jsonl - sampled summaries for human evaluation
    - ratings.json - scores recorded from human evaluation
  - Dockerfile
  - requirements.txt
  - requirements-summ_eval.txt