JOINT GENERATOR-RANKER LEARNING FOR NATURAL LANGUAGE GENERATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Generate-then-rank is a widely used mechanism for text generation, where a generator produces multiple candidates and a ranker chooses the best one. However, existing methods usually train the generator and the ranker separately, which causes a lack of mutual feedback and a misalignment of their objectives. This results in suboptimal generation quality. To address this issue, we propose JGR, a novel joint training algorithm that integrates the generator and the ranker in a single framework. JGR optimizes the generator with a hybrid objective that combines data likelihood and ranker reward, and trains the ranker with a contrastive loss that compares the generator outputs. By alternately updating the generator and the ranker, JGR can effectively harmonize their learning and enhance their quality jointly. We evaluate JGR on various text generation tasks and demonstrate that it surpasses existing methods on four public datasets across three common generation scenarios. Our code and models are publicly available at `https://anonymous.4open.science/r/jgr-anonymous-F597`.

1 INTRODUCTION

The quality of the output texts produced by neural natural language generation (NLG) models, such as those for machine translation (Vaswani et al., 2017) and summarization (Lewis et al., 2019), depends largely on how they are trained and decoded. The conventional approach is to train them with log-likelihood objectives and decode them with greedy or beam search strategies. However, this approach often fails to select the best sample with the highest evaluation score among the generated candidates, as shown by previous studies (Cohen & Beck, 2019; Meister et al., 2020).

To overcome this limitation, some recent works (Liu & Liu, 2021; Liu et al., 2021; Li et al., 2022b; Ravaut et al., 2022) proposed to use a separate ranker model to re-rank the output texts of the generator model, following a generate-then-rank pipeline. This pipeline can improve the quality of the output texts by exploiting the ranker model's ability to evaluate and compare different candidates. However, this pipeline also has a drawback: it requires training the generator and ranker models in two separate phases, which may not fully exploit the potential of the generator model and the feedback from the ranker model.

In this paper, we propose a novel **J**oint training paradigm of both **G**enerator and **R**anker (JGR) for NLG tasks, which aims to overcome the drawback of the generate-then-rank pipeline. Unlike previous works, which train the generator and ranker models separately, we explore a joint and iterative training algorithm that updates both models in turn.

Our main motivation for the joint and iterative training of generator and ranker is twofold: First, the ranker model can provide valuable feedback to the generator model based on the ranking scores of the generated candidates. This encourages the generator model to produce better outputs. Second, the ranker model can also benefit from the outputs of a progressively better generator model, and improve its ranking performance.

The JGR framework consists of a generator and a ranker. During training, the generator and ranker alternate to update their parameters, and each of them involves the other's outputs in its own input signals. Specifically, at the ranker training phase, the ranker model is trained to rank the outputs generated by the generator model for a given input text. At the generator training phase, the generator model uses a combination of the ranker score and the matching score (e.g., BLEU) as the reward for each sample, and trains with policy gradients, which encourages the generator to produce candidates with higher rewards and (Sutton et al., 1999) mitigate the exposure bias issue in the teacher-forcing learning.

To assess the effectiveness of JGR, we conduct experiments on four diverse NLG tasks from different domains, including abstractive summarization (Hermann et al., 2015), conversational summarization (Gliwa et al., 2019), question generation (Rajpurkar et al., 2016), and dialogue (Zhang et al., 2018). The experimental results demonstrate that JGR achieves remarkable performance gains over the conventional MLE training method, with a 3-point increase in ROUGE-2 score on CNN/DailyMail and a 3.5-point increase in BLEU-2 score on PersonaChat.

Furthermore, we make several interesting observations from the results. First, the rewards from the ranker are more effective than the rewards from the direct metrics, but combining them together stabilizes the training and produces a better generator. This simple combination also beats popular RL methods. Second, training the ranker only on the candidates from the generator is better than using ground-truth as positive examples. Third, sampling more candidates during training leads to better performance within a certain range, which is consistent with data augmentation. Finally, the joint training paradigm increases the diversity of the generator outputs, which in turn benefits the ranker training.

2 Related Work

2.1 Natural Language Generation

Natural language generation is a long-standing research topic. RNN-based methods for dialog systems Wen et al. (2015) and convolutional methods for translation Gehring et al. (2016) are some examples of earlier approaches. In the last few years, pre-trained transformer models have advanced the state of the art on many NLG tasks. These models, such as BART (Lewis et al., 2019), ProphetNet (Qi et al., 2020), and T5 (Raffel et al., 2020), use an encoder-decoder architecture and leverage large amounts of unlabeled data. Other models, such as GPT2 (Radford et al., 2019) and UniLM (Dong et al., 2019), use only a decoder or an encoder for natural language generation.

Reinforcement learning can assist the training of NLG models, as shown by several works. Rennie et al. (2017); Paulus et al. (2018) used self-critical methods that measure the reward as the difference between the metric score and the baseline score. Bahdanau et al. (2017); Le et al. (2022) introduced actor-critic frameworks (Konda & Tsitsiklis, 1999), which is also a joint training framework, while they have not considered the contrastive rewards between different candidates given one input. We provided a more detailed comparison in A.1.

Another common approach to NLG is to apply adversarial networks (Goodfellow et al., 2014) in a reinforcement learning-based way. For example, SeqGAN (Yu et al., 2017), RankGAN (Lin et al., 2017), GCN (Lamprier et al., 2022) and SelfGAN (Scialom et al., 2021b). These methods also introduce a joint training framework, however, instead of training a ranker, they trained a discriminator, which distinguishes the ground-truth text and the generator outputs. In Appendix A.2, we detail the main distinctions between these methods and our JGR.

2.2 Generate-then-Rank Framework

The generate-then-rank framework generates some candidate texts with a generator and then ranks them with a ranker. SimCLS (Liu & Liu, 2021), RefSum (Liu et al., 2021), and SumRanker (Ravaut et al., 2022) train rankers separately to rank the outputs of summarization models such as BART (Lewis et al., 2019). In other domains, such as code generation and math problem solving, rankers are also used to evaluate the generated outputs, as shown by AlphaCode (Li et al., 2022b) and Verifier (Cobbe et al., 2021). There are also some works trying to compress the generate-then-rank pipeline to one single model using extra training objectives, for example, MATCHSUM (Zhong et al., 2020), CoLo (An et al., 2022), and BRIO (Liu et al., 2022) with contrastive learning, and Amortized Noisy-Channel NMT (Pang et al., 2021) with Q-learning. However, the above methods do not explore the joint training framework that optimizes both generators and rankers together.

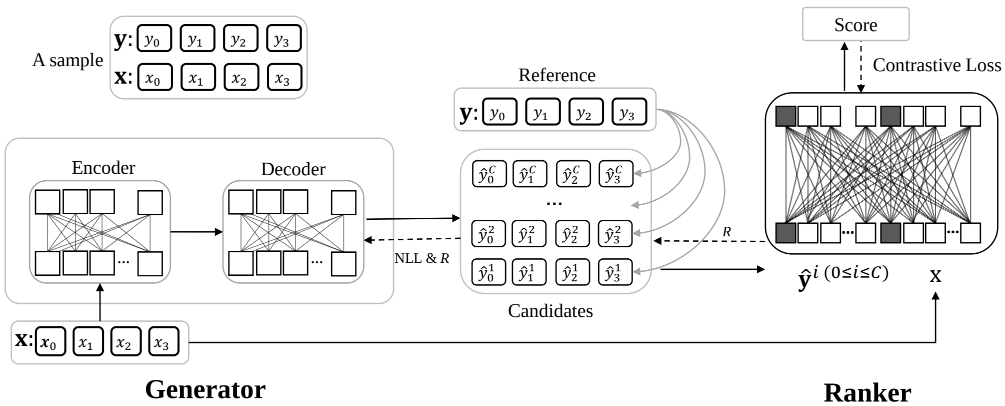

Figure 1: An example to illustrate the generator and ranker in JGR. The input text is first fed into the encoder-decoder generator model to sample candidates, then the candidates are sent to ranker together with the input text to output ranker scores and feedback rewards.

In the retrieve-then-rank framework for dense retrieval (Karpukhin et al., 2020), a retriever first finds relevant documents from a large collection, then a ranker reorders them according to their scores. Our JGR is partially motivated by this framework, we think in the generate-then-rank framework, the generation can be viewed as a retrieval process. Therefore, during training and inference, the generator should sample enough candidates for the ranker to re-rank. Several works have proposed to jointly train the retriever and the ranker to improve generate-then-rerank. Such as RocketQA v2 (Ren et al., 2021) and AR2 (Zhang et al., 2021). However, to our knowledge, JGR is the first work applying the joint training paradigm to the generate-then-rank framework for natural language generation.

3 METHODOLOGY

3.1 PRELIMINARIES

In this section, we introduce the basic elements of conditional text generation, including problem definition, model architecture, and model training.

Given a text pair $(\mathbf{x}, \mathbf{y})$, $\mathbf{x}$ is the input text sequence, $\mathbf{y}$ is the target text sequence. The conditional text generation tasks ask the model to generate a high-quality output $\hat{\mathbf{y}}$ that close the the ground-truth $\mathbf{y}$ based on the input $\mathbf{x}$. We adopt the Transformer (Vaswani et al., 2017) encoder-decoder model as a general framework for conditional text generation. The encoder part transforms $\mathbf{x}$ into a tensor representation $\mathcal{H}_e$ using the Transformer model, as shown in Eqn. 1.

$$\mathcal{H}_e = \textbf{Encoder}(\mathbf{x}), \tag{1}$$

The decoder part uses $\mathcal{H}_e$ as input and produces a text sequence via the auto-regressive fashion.

$$\hat{\mathbf{y}} \sim \textbf{Decoder}(\hat{\mathbf{y}}, \mathcal{H}_e) = \prod_{t=1}^{|\hat{y}|} p(\hat{y}_t | \hat{y}_{<t}, \mathcal{H}_e) \tag{2}$$

To simplify the notation, we write $G_\theta(\cdot)$ for the encoder-decoder generation model with parameters θ, and $p_{G_\theta}(\hat{\mathbf{y}}|\mathbf{x})$ for the probability of generating $\hat{\mathbf{y}}$ given $\mathbf{x}$. The standard way to train the encoder-decoder sequence generation model is to minimize the negative log-likelihood of the ground-truth target sequence:

$$\mathcal{L}_{\text{NLL}} = -\sum_{t=1}^{|\mathbf{y}|} \log p_{G_\theta}(y_t | y_{<t}, \mathbf{x}), \tag{3}$$

When inference, the generator adopts a decoding strategy such as beam search. However, previous studies (Cohen & Beck, 2019; Meister et al., 2020) observed that the top-scored candidate from decoding strategies is often not the candidate with the highest evaluation score. Therefore we design JGR, which alleviates this problem through joint training of generator and ranker.

3.2 JOINT GENERATOR-RANKER TRAINING

The model architecture of JGR, shown in Figure 1, has two components: a generator that outputs several text candidates for an input text using an encoder-decoder model, and a ranker that scores these text candidates. The JGR workflow works as follows: a) the generator decodes multiple text candidates from the input text; b) the input text and the text candidates are combined and sent to the ranker; c) the ranker learns to rank the text candidates via a contrastive learning objective; d) the ranker gives a reward to each text candidate, which in turn is used to train the generator.

We use $G_\theta(\cdot)$ and $D_\phi(\cdot)$ to represent the generator model and ranker model respectively, where $G_\theta(\cdot)$ is a text generation model with an encoder-decoder structure as explained in section 3.1, and $D_\phi(\cdot)$ works as scoring model that takes the concatenation of input text $\mathbf{x}$ and and generated text $\hat{\mathbf{y}}$ as input, and outputs a scalar value $s_{\hat{y}}$ reprensenting the quality of the generated text:

$$s_{\hat{y}} = D_\phi([\mathbf{x}, \hat{\mathbf{y}}]) \tag{4}$$

During the training stage, the generator and ranker are trained alternatively and iteratively. Algorithm 1 shows the training procedure of JGR. We first warm up the generator G_θ with a standard negative log-likelihood (NLL) loss according to Eqn 3. Then, we iteratively update the ranker and generator:

Fix $G_\theta(\cdot)$, Train $D_\phi(\cdot)$: the goal of the ranker model $D_\phi(\cdot)$ is to choose the best sample from a set of candidates generated by the generator model, which we denote as $\hat{\mathcal{Y}} = \{\hat{\mathbf{y}}^1, \hat{\mathbf{y}}^2, ..., \hat{\mathbf{y}}^C\}$

$$\{\hat{\mathbf{y}}^1, \hat{\mathbf{y}}^2, ..., \hat{\mathbf{y}}^C\} \sim p_{G_\theta}(\cdot|\mathbf{x}), \tag{5}$$

C is the number of sampled candidates. For each $\hat{\mathbf{y}}^i$, we calculate the matching score (e.g., BLEU or ROUGE) with the ground-truth text $\mathbf{y}$, denoted as $\Delta(\mathbf{y}, \hat{\mathbf{y}}^i)$. Then, we pick up the positive and negative samples in the candidate set by their matching scores, for the training of the ranker. Specifically, $\hat{\mathbf{y}}^+$ is the text candidate with the highest matching score, and $\hat{\mathcal{Y}}^-$, whose size is a hyper-parameter, is the negative candidate set containing a certain number of candidates with the lowest scores. The ranker model can be trained by minimizing contrastive loss:

$$\mathcal{L}^\phi = -\log p_{D_\phi}(\hat{\mathbf{y}}^+|\hat{\mathcal{Y}}^-, \mathbf{x}), \tag{6}$$

where $p_{D_\phi}(\hat{\mathbf{y}}^+|\hat{\mathcal{Y}}^-, \mathbf{x})$ is the probability of selecting $\hat{y}^+$ from the set $\{\hat{\mathbf{y}}^+\} \cup \mathcal{Y}^-$, which is computed by applying softmax function on the ranking scores:

$$p_{D_\phi}(\hat{\mathbf{y}}^+|\hat{\mathcal{Y}}^-, \mathbf{x}) = \frac{\exp^{s_{\hat{y}^+}}}{\exp^{s_{\hat{y}^+}} + \sum_{\hat{\mathbf{y}}^- \in \hat{\mathcal{Y}}^-} \exp^{s_{\hat{y}^-}}}, \tag{7}$$

After several training steps of updating the ranker, we turn to fix the ranker and update the generator.

Fix $D_\phi(\cdot)$, Train $G_\theta(\cdot)$: the generator model is trained in two ways. The first one is $\mathcal{L}_{\text{NLL}}$, which uses a teacher-forcing mechanism to minimize the negative log-likelihood loss function over the training instances as discussed in section 3.1 (Eqn. 3). The second one is $\mathcal{L}_{\text{RL}}$ - a reinforcement learning-based approach in which the generator model acts as a policy network to produce a list of text samples $\hat{\mathcal{Y}}$ given the input $\mathbf{x}$, and the ranker model gives a reward to each text sample in $\hat{\mathcal{Y}}$ based on its ranking score. The generator model can be trained by maximizing (minimizing) the expected (negative) reward (Sutton et al., 1999).

$$\mathcal{L}_{\text{RL}} = -\sum_{\hat{\mathbf{y}} \in \hat{\mathcal{Y}}} (\mathcal{R}(\hat{\mathbf{y}}) - b) \sum_t \log p_{G_\theta}(\hat{y}_t|\hat{y}_{<t}, \mathbf{x}), \tag{8}$$

where $\mathcal{R}(\hat{\mathbf{y}})$ is the reward for sample $\hat{\mathbf{y}}$, calculated by combining the matching score $\Delta(\hat{\mathbf{y}}, \mathbf{y})$ and the ranking score $s_{\hat{\mathbf{y}}}$: $\mathcal{R}(\hat{\mathbf{y}}) = \Delta(\hat{\mathbf{y}}, \mathbf{y}) + s_{\hat{\mathbf{y}}}$, baseline b is used to reduce the variance in RL training, which is computed by averaging the rewards of all samples in the candidate set: $b = \sum_{\hat{\mathbf{y}} \in \hat{\mathcal{Y}}} \mathcal{R}(\hat{\mathbf{y}})/C$.

We then combine $\mathcal{L}_{\text{NLL}}$ and $\mathcal{L}_{\text{RL}}$ to form the final objective function for generator model training :

$$\mathcal{L}^\theta = \mathcal{L}_{\text{NLL}} + \mathcal{L}_{\text{RL}}. \tag{9}$$

After updating the generator for several steps, we go back to fixing the generator and updating the ranker. This iteration will continue until the entire JGR framework converges.

Algorithm 1 Joint Training of Generator and Ranker (JGR)

Require: Generator G_θ; Ranker D_ϕ; Training data $\mathbb{D}$.
Initialize G_θ and D_ϕ from the pre-trained language models.
Train the warm-up generator G_θ^0 on $\mathbb{D}$.
while model has not converged **do**
  for training steps A **do**
    Sample candidates $\hat{\mathcal{Y}} \sim p_{G_\theta}(\cdot|\mathbf{x})$ for each $\mathbf{x}$ in the mini-batch.
    Select $\hat{\mathbf{y}}^+$ and $\hat{\mathcal{Y}}^-$ from $\hat{\mathcal{Y}}$
    Update parameters of D_ϕ with Eq 6.
  end for
  for training steps B **do**
    Sample candidates $\hat{\mathcal{Y}} \sim p_{G_\theta}(\cdot|\mathbf{x})$ for each $\mathbf{x}$ in the mini-batch.
    Compute rewards $\mathcal{R}(\hat{\mathbf{y}})$ for each $\hat{\mathbf{y}} \in \hat{\mathcal{Y}}$.
    Update parameters of G_θ with Eq 9.
  end for
end while

4 EXPERIMENT

We begin the experimental section by first introducing the implementation details of JGR, including the datasets and experimental settings. Then we show the overall performance of JGR and other compared methods. After that, we conduct sever analyses: First, we compare the ranker of JGR with other rankers. Second, we examine several types of rewards to see how they impact the training. Third, we investigate how the different types and numbers of sampled candidates will affect the performance of JGR. And finally, we conduct an analysis to show the necessity of joint training.

4.1 DATASETS

We evaluate the proposed method on four publicly available benchmarks: CNN/DailyMail, SAMSum SQuAD 1.1, and PersonaChat. The statistics of these benchmarks and the details of evaluation metrics are in Appendix E.

CNN/DailyMail (Hermann et al., 2015) is a benchmark for summarization. Both extractive and abstractive summarization models can be applied on CNN/DailyMail. Since our JGR focuses on text generation, we treat CNN/DailyMail as an abstractive summarization task. There are two versions: anonymized and non-anonymized. We use the non-anonymized dataset See et al. (2017). The evaluation metrics are Rouge-1, Rouge-2, and Rouge-L.

SAMSum (Gliwa et al., 2019) is a benchmark for conversational summarization, whose inputs are the concatenation of dialog context. The evaluation metrics are Rouge-1, Rouge-2, and Rouge-L.

SQuAD 1.1 (Rajpurkar et al., 2016) is originally an machine reading comprehension dataset. We follow the data split and pre-processing as done by Du et al. (2017); Zhao et al. (2018); Liu et al. (2020), to make it a question generation dataset, which treats the concatenation of the answer span and article as the input, and the question as the target output. The evaluation metrics are Rouge-L, Bleu-4, and METEOR.

PersonaChat Zhang et al. (2018) contains about 160K utterances. Given the multi-turn conversations and persona profile, the model learns to generate the response. The evaluation metrics are Bleu-1, Bleu-2, and the ratio of distinct unigrams and bigrams in the generated responses (Distinct-1 and Distinct-2).

4.2 SETTINGS

We use BART-large (Lewis et al., 2019) as the backbone model for the generator. The backbone of the ranker is based on RoBERTa-large (Liu et al., 2019). The generator and ranker models are

Table 1: Overall results on CNN/DailyMail and SAMSum. The results with "†" means from our implementation. The results with "*" are the results of backbone models for JGR-G.

Method	CNN/DailyMail				SAMSum			
	R-1	R-2	R-L	AVG	R-1	R-2	R-L	AVG
Lead-3	40.42	17.62	36.67	31.57	-	-	-	-
PTGEN (See et al., 2017)	36.44	15.66	33.42	28.51	-	-	-	-
PTGEN-COV (See et al., 2017)	39.53	17.28	36.38	31.06	-	-	-	-
BART (Lewis et al., 2019)	44.16*	21.28*	40.90*	35.45*	52.86†*	28.24†*	48.57†*	43.22†*
PEGASUS (Zhang et al., 2020)	44.17	21.47	41.11	35.58	51.99	27.59	47.56	42.38
ProphetNet (Qi et al., 2021)	44.20	21.17	41.30	35.56	52.62	27.77†	48.33	42.91
GSUM (Dou et al., 2021)	45.94	22.32	42.48	36.91	-	-	-	-
BRIO (Liu et al., 2022)	47.48*	23.55*	44.57*	38.53*	-	-	-	-
JGR-G	46.86	23.18	43.74	37.93	53.85	29.22	49.93	44.33
JGR-R	47.63	**23.59**	44.50	38.57	**54.30**	**29.48**	**50.51**	**44.76**
JGR-G$_{\text{init w. BRIO}}$	48.39	23.22	46.11	39.24	-	-	-	-
JGR-R$_{\text{init w. BRIO}}$	**48.86**	23.35	**46.56**	**39.59**	-	-	-	-

Table 2: Overall results on SQuAD 1.1.

	R-L	B-4	MTR
MASS (Song et al., 2019)	50.98	23.14	25.36
BART (Lewis et al., 2019)	51.46*	23.14*	26.56*
UNILM (Dong et al., 2019)	52.04	23.75	25.61
ProphetNet (Qi et al., 2020)	51.50	22.50	26.00
JGR-G	52.79	24.52	26.46
JGR-R	**53.57**	**24.73**	**26.97**

Table 3: Overall results on PersonaChat.

	B-1	B-2	D-1	D-2
BART (Lewis et al., 2019)	49.9*	40.0*	1.3*	8.0*
PLATO$_{\text{w/o lantent}}$ (Bao et al., 2020)	40.6	31.5	**2.1**	**12.1**
PLATO (Bao et al., 2020)	45.8	35.7	1.2	6.4
ProphetNet (Qi et al., 2020)	46.7	39.0	1.3	7.5
DialogVED (Chen et al., 2022)	48.2	39.9	1.5	9.4
JGR-G	52.5	43.2	1.4	6.2
JGR-R	**53.3**	**43.5**	1.5	8.0

initialized with the off-the-shelf checkpoints[1]. On CNN/DailyMail , apart from initializing JGR with the language models, we also evaluate JGR that initializes the generator using the previous state-of-the-art model BRIO (Liu et al., 2022)[2].

In the model training, the generator model adopts a nucleus sampling approach to generate the candidate set with temperature $= 1.0$ and top(p) $= 1.0$. In inference, we apply beam search decoding strategy with beam size $= 16$, and length penalty = 1.0; The details of other hyper-parameters (e.g., learning rate and training epochs, etc) are listed in Appendix F. The JGR model is implemented based on the open-source Huggingface Transformers framework (Wolf et al., 2020). We conduct experiments on a single node of 8 NVIDIA A100 GPUs.

It is worth noting that in order to initialize the ranker with a more general and reasonable ranking function, we increase the number of training steps and add a certain number of warm-up steps at the first ranker training iteration. The hyper-parameters of the first ranker training iteration are also introduced in Appendix F.

4.3 Overall Results

Table 1 shows the results of JGR and other baseline methods on summarization tasks CNN/DailyMail and SAMSum. "Lead-3" is an ad-hoc summarization approach that uses the first three sentences in the article as the summary. "PTGEN" and "PTGEN-COV" are sequence-to-sequence generation methods without pre-training. Other baselines are pre-trained language models fine-tuned on the benchmarks. "JGR-G" indicates the generator model in JGR, and "JGR-R" is using the ranker of JGR to re-rank the outputs of JGR-G. "JGR-G/R$_{\text{init w. BRIO}}$" are our JGR with the generator initialized from BRIO. As shown in Table 1, the generator model (JGR-G) itself achieves a considerable performance gain compared with its backbone models on both the two benchmarks, which verifies the effectiveness of the proposed JGR training to obtain a better generator. On both CNN/DailyMail and SAMSum, the ranker (JGR-R) can further improve the performance upon JGR-G. Both JGR-G and JGR-R can reach state-of-the-art on SAMSum. If initialized with BRIO, both our JGR-G and JGR-R can surpass the state-of-the-art on CNN/DailyMail with a significant margin over the average ROUGE score.

In Table 2, we compare the performance of JGR with four pre-trained language models (Song et al., 2019; Lewis et al., 2019; Dong et al., 2019; Qi et al., 2020) on SQuAD 1.1, since they have reported

[1]RoBERTa: `https://huggingface.co/roberta-large`, BART: `https://huggingface.co/facebook/bart-large`

[2]BRIO: `https://github.com/yixinL7/BRIO`

the results finetuned and evaluated in the same data split as in Liu et al. (2020). With a relatively weak backbone model, BART, our JGR-G can still outperform all the compared baselines. And JGR-R can also further improve the results of JGR-G.

Table 3 shows the results of compared methods in persona-based response generation. As shown in the results, our JGR-G and JGR-R can surpass the baselines significantly on the metrics of BLEU-1 and BLEU-2. However, both JGR-G and JGR-R can only perform the same level of the baselines on Distinct-1 and Distinct-2. It is noteworthy that PLATO and DialogVED are the only two language models that are pre-trained using a conversational corpus among these baselines. They achieved high scores on Distinct-1 and Distinct-2, showing the importance of pre-training corpus.

4.4 Performance of Ranker

Recently, several works adopt the generate-then-rerank framework, especially on the summarization tasks (Liu & Liu, 2021; Liu et al., 2021; Ravaut et al., 2022; Liu et al., 2022). We compare these methods with that our JGR-R on CNN/DailyMail. Since all the above methods train the ranker separately with the fine-tuned BART as the generator on CNN/DailyMail, we only report their results in this setting.

Table 4: The results of different rankers on CNN/DailyMail. "Gain" represents the performance gain of ranker compared with the used generator over the average score.

Generator	Ranker	R-1	R-2	R-L	Gain
G^0	-	44.16	21.28	40.90	0.00
G^0	SimCLS	46.67	22.15	43.54	2.00
G^0	RefSum	45.15	21.70	42.00	0.83
G^0	SumRanker	46.62	22.39	43.59	2.08
G^0	BRIO	47.28	22.93	44.15	**3.84**
G^0	D^0	45.54	22.27	42.25	1.24
JGR-G	-	46.86	23.18	43.74	0.00
JGR-G	JGR-R	**47.63**	**23.59**	**44.50**	0.64

The experimental results are shown in Table 4, where G^0 denotes the base generator, i.e. BART, and D^0 is the ranker after the first ranker training iteration, as described in Section 4.2. Several observations can be seen in the results. First, our JGR can achieve the highest score with the inference pipeline. Second, on CNN/DailyMail, the performance gain brought by JGR-R is not as big as other related methods which introduced some extra modules to their models. Third, on CNN/DailyMail, after the joint training in JGR, the performance gain brought by the ranker drops. We think this is because as the generator's performance grows, the quality of candidates rises, causing the ranker harder to tell which on is the best among all candidates.

4.5 Impact of Rewards

In this section, we investigate the impact of rewards. We compare different reward settings on CNN/DailyMail. The compared methods are as follows: 1) **RL** is the conventional self-critical reinforcement-learning method where the rewards are the metric scores $\Delta(\hat{\mathbf{y}}, \mathbf{y})$, and the greedy search output is used as baseline Rennie et al. (2017); Paulus et al. (2018). 2) **JGR-G**$_{\text{only mr}}$/**JGR-G**$_{\text{only rr}}$ are our JGR where the generator is trained without the rewards from generator/metrics. The standard NLL loss is added in all the compared methods. The results are shown in Table 5.

Table 5: Results generator trained with different type of rewards on CNN/DailyMail.

	R-1	R-2	R-L	AVG
BART	44.16	21.28	40.90	35.45
RL	44.14	21.20	40.95	35.43
JGR-G	**46.86**	**23.18**	43.74	**37.93**
JGR-G$_{\text{only mr}}$	44.20	21.37	41.04	35.54
JGR-G$_{\text{only rr}}$	46.76	22.99	**43.81**	37.85

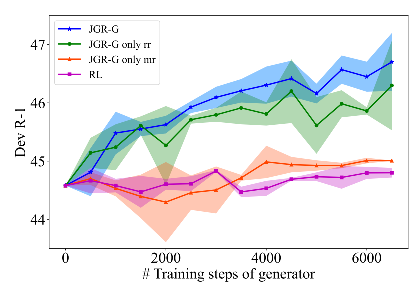

Figure 2: Dev scores for methods with different type of rewards.

From the results, we can see that our JGR can outperform traditional RL significantly. In addition, both JGR-G$_{\text{only mr}}$ and JGR-G$_{\text{only rr}}$ suffer a performance decline compared to standard JGR-G, and the performance of JGR-G$_{\text{only mr}}$ is far worse than that of JGR-G$_{\text{only rr}}$, indicating that using rewards from ranker contributes more than using rewards from metrics, and it is better to combine them. In Figure 2, we plot the curves of the dev scores under 3 random runs for the compared methods. As illustrated in the figure, although the standard RL method appears to have a small variance under different random runs, its dev scores are hard to grow while training. The JGR-$G_{\text{only rr}}$ has a smaller variance than JGR-$G_{\text{only mr}}$, however, it fails to achieve a high dev score. Our standard JGR, which combines metric rewards and ranker rewards, not only shows the relatively small variance in randomized trials but also can steadily improve the dev score during training.

Table 6: Results of JGR with different candidate picking strategies on CNN/DailyMail.

	Generator				Ranker			
	R-1	R-2	R-L	AVG	R-1	R-2	R-L	AVG
$\hat{\mathbf{y}}^+$=GT	45.64	22.27	42.55	36.82	44.20	21.46	41.22	35.63
$\hat{\mathcal{Y}}^- =$ BOT($\hat{\mathcal{Y}}$)	**46.86**	**23.18**	**43.74**	**37.93**	**47.63**	**23.59**	**44.50**	**38.57**
$\hat{\mathcal{Y}}^- =$ TOP($\hat{\mathcal{Y}}$)	44.16	21.31	41.00	35.49	44.07	21.23	40.91	35.40
$\hat{\mathcal{Y}}^- =$ RAND($\hat{\mathcal{Y}}$)	44.68	21.65	41.42	35.92	45.80	22.68	42.56	37.01
$\hat{\mathcal{Y}}^- =$ TOP-BOT($\hat{\mathcal{Y}}$)	44.86	21.80	41.64	36.10	46.12	22.76	42.91	37.26

Table 7: Results of JGR with numbers of sampled candidates on CNN/DailyMail.

	Generator				Ranker			
	R-1	R-2	R-L	AVG	R-1	R-2	R-L	AVG
$C = 2$	44.59	21.63	41.32	35.85	46.15	22.76	42.86	37.26
$C = 4$	45.44	22.19	42.80	36.81	46.70	23.10	43.81	37.87
$C = 6$	46.36	22.77	42.94	37.37	47.32	23.49	44.29	38.37
$C = 8$	**46.86**	**23.18**	**43.74**	**37.93**	**47.63**	23.59	**44.50**	**38.57**
$C = 16$	46.34	22.97	43.11	37.47	47.34	**23.64**	44.13	38.37
$C = 32$	46.14	22.78	42.87	37.26	47.25	23.48	43.98	38.24
$C = 40$	46.29	22.98	43.00	37.42	47.26	23.60	44.00	38.29

4.6 Candidate Picking Strategies

We examine how different types and numbers of candidates can affect the performance of JGR. We first compare different methods of picking positive candidates and negative candidates when training the ranker. The results are shown in Table 6. The $\hat{\mathbf{y}}^+$=GT denotes that the best candidate $\hat{\mathbf{y}}^+$ is not sampled from the generator, but always the reference. The result shows that if the best candidate is always the reference, the performance of the generator is not as good as the standard JGR, and the ranker's performance is even worse than the generator. We think the reason is that the ranker is misled by the reference, thus it may always misclassify the references as the positive candidates, while other candidates sampled by the generator as the negative candidates. As a result, neither the ranker is well-trained, nor it can pass proper rewards to train the generator.

The last four lines of Table 6 show the results of methods for picking negative samples, i.e., with the lowest matching scores (BOT($\hat{\mathcal{Y}}$), our standard setting), with the highest matching scores (TOP($\hat{\mathcal{Y}}$)), randomly pick (RAND($\hat{\mathcal{Y}}$)), and half has the highest matching scores and the second half has the lowest matching score (TOP-BOT($\hat{\mathcal{Y}}$)). From the results, we can see that our standard setting (BOT($\hat{\mathcal{Y}}$)) significantly outperforms other negative candidate picking strategies.

In Table 7, we show the performance of JGR with different numbers of sampled candidates when training the generator. As shown in the table, under a certain range ($C = 2 \sim 8$), the performance of JGR rises as the number of candidates increases. We attribute this to the fact that increasing the number of candidates means that the generator can be optimized on more probabilities from candidates, which is to some extent a way of data augmentation. However, the performance does not grow as desired when the number of candidates becomes too large.

Table 8: Results of JGR and JGR without joint training on CNN/DailyMail.

	Generator				Ranker			
	R-1	R-2	R-L	AVG	R-1	R-2	R-L	AVG
JGR	**46.86**	**23.18**	**43.74**	**37.93**	**47.63**	**23.59**	**44.50**	**38.57**
w/o joint	45.02	21.83	42.40	36.42	45.10	21.81	42.47	36.46

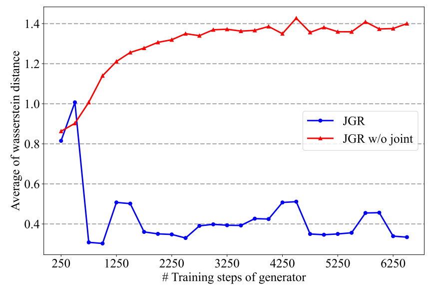

Figure 3: The average of Wasserstein distances between ranker rewards and metrics rewards.

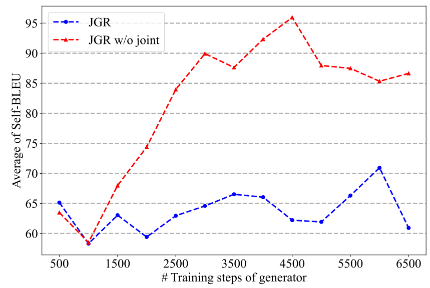

Figure 4: The average of self-BLEU at each training interval.

4.7 Does Joint Training Matter?

To see how the joint training affects our method, we compare the performance of our JGR and the variant that only trains the generator in JGR while fixing the ranker after the first ranker training iteration ($\text{JGR}_{\text{w/o joint}}$). As the results shown in Table 8, $\text{JGR}_{\text{w/o joint}}$ is far worse than JGR, and $\text{JGR-R}_{\text{w/o joint}}$ achieves no performance gain over $\text{JGR-G}_{\text{w/o joint}}$. To take an in-depth look, we analyze the distribution of rewards. We first draw the curves of the Wasserstein distance between ranker rewards and metrics rewards at each training interval for JGR and $\text{JGR}_{\text{w/o joint}}$. As illustrated in Figure 3, the Wasserstein distances of JGR are hovering within a range, while the Wasserstein distances of $\text{JGR}_{\text{w/o joint}}$ are growing extremely high, which means the distribution of ranker rewards and metrics rewards are quite different in $\text{JGR}_{\text{w/o joint}}$. Therefore we think that $\text{JGR-R}_{\text{w/o joint}}$ might not assign the proper rewards to the sampled candidates, due to it not being jointly trained.

We also analyse the diversity of sampled candidates for JGR-G and $\text{JGR-G}_{\text{w/o joint}}$. We use self-BLEU[3] to measure the diversity of sampled candidates. A larger self-BLEU score means a lower diversity of the sampled candidates. We show the curves of the average self-BLEU score for generated candidates at each training interval in Figure 4. From the figure, we can see that the self-BLEU of $\text{JGR}_{\text{w/o joint}}$ increases rapidly after the generator is trained 1000 steps, while the same situation never happens in JGR. It indicates that if the ranker is not jointly trained with the generator, the rewards it feeds back to the generator will cause the generator to sample candidates that are more and more similar to each other, making the training of JGR harder. On the contrary, joint training can erase this phenomenon and help to keep a certain level of diversity in sampled candidates, thus leading to better training.

4.8 More Discussions

Due to the page limit, we show more discussions about JGR compared to reinforcement learning, GAN, data augmentation in Appendix A, the evaluation on advanced metrics in Appendix C and the impact of decoding strategies in Appendix D.

5 Conclusion

In this paper, we propose a novel Joint training of Generator and Ranker framework, namely JGR, for natural language generation. Both the generator and ranker of our JGR can achieve state-of-the-art results on several benchmarks in the areas of summarization, question generation, and dialog. We also analyze our JGR in several aspects and find that: First, the rewards from the ranker work better than the rewards from the direct metrics such as BLEU, but combining them together helps the training become more stable. Second, during training, letting the ranker be trained on the candidates generated by the generator exclusively is even better than previous approaches using ground-truth as positive examples. Third, more candidates being sampled during training can lead to better performance, which is consistent with the findings from data augmentation. Finally, the joint training paradigm helps the generator sample candidates with higher diversity, which in turn contribute to the training.

[3]We introduce the computation of self-BLEU in Appendix B.

REFERENCES

Chenxin An, Ming Zhong, Zhiyong Wu, Qin Zhu, Xuanjing Huang, and Xipeng Qiu. CoLo: A contrastive learning based re-ranking framework for one-stage summarization. In *Proceedings of the 29th International Conference on Computational Linguistics*, pp. 5783–5793, Gyeongju, Republic of Korea, October 2022. International Committee on Computational Linguistics. URL https://aclanthology.org/2022.coling-1.508.

Dzmitry Bahdanau, Philemon Brakel, Kelvin Xu, Anirudh Goyal, Ryan Lowe, Joelle Pineau, Aaron Courville, and Yoshua Bengio. An actor-critic algorithm for sequence prediction. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=SJDaqqveg.

Siqi Bao, Huang He, Fan Wang, Hua Wu, and Haifeng Wang. PLATO: Pre-trained dialogue generation model with discrete latent variable. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 85–96, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.9. URL https://aclanthology.org/2020.acl-main.9.

Wei Chen, Yeyun Gong, Song Wang, Bolun Yao, Weizhen Qi, Zhongyu Wei, Xiaowu Hu, Bartuer Zhou, Yi Mao, Weizhu Chen, Biao Cheng, and Nan Duan. DialogVED: A pre-trained latent variable encoder-decoder model for dialog response generation. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 4852–4864, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-long.333. URL https://aclanthology.org/2022.acl-long.333.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems, 2021. URL https://arxiv.org/abs/2110.14168.

Eldan Cohen and Christopher Beck. Empirical analysis of beam search performance degradation in neural sequence models. In Kamalika Chaudhuri and Ruslan Salakhutdinov (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 1290–1299. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/cohen19a.html.

Li Dong, Nan Yang, Wenhui Wang, Furu Wei, Xiaodong Liu, Yu Wang, Jianfeng Gao, Ming Zhou, and Hsiao-Wuen Hon. Unified language model pre-training for natural language understanding and generation. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/c20bb2d9a50d5ac1f713f8b34d9aac5a-Paper.pdf.

Zi-Yi Dou, Pengfei Liu, Hiroaki Hayashi, Zhengbao Jiang, and Graham Neubig. GSum: A general framework for guided neural abstractive summarization. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 4830–4842, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.384. URL https://aclanthology.org/2021.naacl-main.384.

Xinya Du, Junru Shao, and Claire Cardie. Learning to ask: Neural question generation for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1342–1352, Vancouver, Canada, July 2017. Association for Computational Linguistics. doi: 10.18653/v1/P17-1123. URL https://www.aclweb.org/anthology/P17-1123.

Jonas Gehring, Michael Auli, David Grangier, and Yann N Dauphin. A convolutional encoder model for neural machine translation. *arXiv preprint arXiv:1611.02344*, 2016.

Bogdan Gliwa, Iwona Mochol, Maciej Biesek, and Aleksander Wawer. Samsum corpus: A human-annotated dialogue dataset for abstractive summarization. In *Proceedings of the 2nd Workshop on New Frontiers in Summarization*, pp. 70–79, 2019.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger (eds.), *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014. URL https://proceedings.neurips.cc/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf.

Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching machines to read and comprehend. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pp. 1693–1701, 2015. URL https://proceedings.neurips.cc/paper/2015/hash/afdec7005cc9f14302cd0474fd0f3c96-Abstract.html.

Vladimir Karpukhin, Barlas Oğuz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. *arXiv preprint arXiv:2004.04906*, 2020.

Vijay Konda and John Tsitsiklis. Actor-critic algorithms. *Advances in neural information processing systems*, 12, 1999.

Wojciech Kryscinski, Bryan McCann, Caiming Xiong, and Richard Socher. Evaluating the factual consistency of abstractive text summarization. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 9332–9346, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.750. URL https://aclanthology.org/2020.emnlp-main.750.

Sylvain Lamprier, Thomas Scialom, Antoine Chaffin, Vincent Claveau, Ewa Kijak, Jacopo Staiano, and Benjamin Piwowarski. Generative cooperative networks for natural language generation. *arXiv preprint arXiv:2201.12320*, 2022.

Hung Le, Yue Wang, Akhilesh Deepak Gotmare, Silvio Savarese, and Steven CH Hoi. Coderl: Mastering code generation through pretrained models and deep reinforcement learning. *arXiv preprint arXiv:2207.01780*, 2022.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*, 2019.

Bohan Li, Yutai Hou, and Wanxiang Che. Data augmentation approaches in natural language processing: A survey. *AI Open*, 3:71–90, 2022a. ISSN 2666-6510. doi: https://doi.org/10.1016/j.aiopen.2022.03.001. URL https://www.sciencedirect.com/science/article/pii/S2666651022000080.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, Cyprien de Masson d'Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, James Molloy, Daniel J. Mankowitz, Esme Sutherland Robson, Pushmeet Kohli, Nando de Freitas, Koray Kavukcuoglu, and Oriol Vinyals. Competition-level code generation with alphacode, 2022b.

Kevin Lin, Dianqi Li, Xiaodong He, Zhengyou Zhang, and Ming-ting Sun. Adversarial ranking for language generation. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/bf201d5407a6509fa536afc4b380577e-Paper.pdf.

Dayiheng Liu, Yu Yan, Yeyun Gong, Weizhen Qi, Hang Zhang, Jian Jiao, Weizhu Chen, Jie Fu, Linjun Shou, Ming Gong, et al. Glge: A new general language generation evaluation benchmark. *arXiv preprint arXiv:2011.11928*, 2020.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

Yixin Liu and Pengfei Liu. SimCLS: A simple framework for contrastive learning of abstractive summarization. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pp. 1065–1072, Online, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-short.135. URL https://aclanthology.org/2021.acl-short.135.

Yixin Liu, Zi-Yi Dou, and Pengfei Liu. RefSum: Refactoring neural summarization. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 1437–1448, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.113. URL https://aclanthology.org/2021.naacl-main.113.

Yixin Liu, Pengfei Liu, Dragomir Radev, and Graham Neubig. BRIO: Bringing order to abstractive summarization. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2890–2903, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-long.207. URL https://aclanthology.org/2022.acl-long.207.

Clara Meister, Ryan Cotterell, and Tim Vieira. If beam search is the answer, what was the question? In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 2173–2185, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.170. URL https://aclanthology.org/2020.emnlp-main.170.

Richard Yuanzhe Pang, He He, and Kyunghyun Cho. Amortized noisy channel neural machine translation. *arXiv preprint arXiv:2112.08670*, 2021.

Romain Paulus, Caiming Xiong, and Richard Socher. A deep reinforced model for abstractive summarization. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=HkAClQgA-.

Weizhen Qi, Yu Yan, Yeyun Gong, Dayiheng Liu, Nan Duan, Jiusheng Chen, Ruofei Zhang, and Ming Zhou. Prophetnet: Predicting future n-gram for sequence-to-sequence pre-training. *arXiv preprint arXiv:2001.04063*, 2020.

Weizhen Qi, Yeyun Gong, Yu Yan, Can Xu, Bolun Yao, Bartuer Zhou, Biao Cheng, Daxin Jiang, Jiusheng Chen, Ruofei Zhang, et al. Prophetnet-x: large-scale pre-training models for english, chinese, multi-lingual, dialog, and code generation. *arXiv preprint arXiv:2104.08006*, 2021.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 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. URL http://jmlr.org/papers/v21/20-074.html.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 2383–2392, Austin, Texas, November 2016. Association for Computational Linguistics. doi: 10.18653/v1/D16-1264. URL https://www.aclweb.org/anthology/D16-1264.

Mathieu Ravaut, Shafiq Joty, and Nancy F Chen. Summareranker: A multi-task mixture-of-experts re-ranking framework for abstractive summarization. *arXiv preprint arXiv:2203.06569*, 2022.

Ruiyang Ren, Yingqi Qu, Jing Liu, Wayne Xin Zhao, Qiaoqiao She, Hua Wu, Haifeng Wang, and Ji-Rong Wen. Rocketqav2: A joint training method for dense passage retrieval and passage re-ranking. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 2825–2835, 2021.

Steven J. Rennie, Etienne Marcheret, Youssef Mroueh, Jerret Ross, and Vaibhava Goel. Self-critical sequence training for image captioning. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1179–1195, 2017. doi: 10.1109/CVPR.2017.131.

Thomas Scialom, Paul-Alexis Dray, Sylvain Lamprier, Benjamin Piwowarski, Jacopo Staiano, Alex Wang, and Patrick Gallinari. QuestEval: Summarization asks for fact-based evaluation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 6594–6604, Online and Punta Cana, Dominican Republic, November 2021a. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.529. URL https://aclanthology.org/2021.emnlp-main.529.

Thomas Scialom, Paul-Alexis Dray, Jacopo Staiano, Sylvain Lamprier, and Benjamin Piwowarski. To beam or not to beam: That is a question of cooperation for language gans. *Advances in neural information processing systems*, 34:26585–26597, 2021b.

Abigail See, Peter J Liu, and Christopher D Manning. Get to the point: Summarization with pointer-generator networks. *arXiv preprint arXiv:1704.04368*, 2017.

Shiqi Shen, Yong Cheng, Zhongjun He, Wei He, Hua Wu, Maosong Sun, and Yang Liu. Minimum risk training for neural machine translation. *CoRR*, abs/1512.02433, 2015. URL http://arxiv.org/abs/1512.02433.

Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. MASS: Masked sequence to sequence pre-training for language generation. In Kamalika Chaudhuri and Ruslan Salakhutdinov (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 5926–5936. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/song19d.html.

Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In S. Solla, T. Leen, and K. Müller (eds.), *Advances in Neural Information Processing Systems*, volume 12. MIT Press, 1999. URL https://proceedings.neurips.cc/paper/1999/file/464d828b85b0bed98e80ade0a5c43b0f-Paper.pdf.

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.

Ashwin K. Vijayakumar, Michael Cogswell, Ramprasaath R. Selvaraju, Qing Sun, Stefan Lee, David J. Crandall, and Dhruv Batra. Diverse beam search: Decoding diverse solutions from neural sequence models. *CoRR*, abs/1610.02424, 2016. URL http://arxiv.org/abs/1610.02424.

Tsung-Hsien Wen, Milica Gasic, Nikola Mrksic, Pei-Hao Su, David Vandyke, and Steve Young. Semantically conditioned lstm-based natural language generation for spoken dialogue systems. *arXiv preprint arXiv:1508.01745*, 2015.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, 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*, pp. 38–45, Online, October 2020. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/2020.emnlp-demos.6.

Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. Seqgan: Sequence generative adversarial nets with policy gradient. *Proceedings of the AAAI Conference on Artificial Intelligence*, 31(1), Feb. 2017. doi: 10.1609/aaai.v31i1.10804. URL https://ojs.aaai.org/index.php/AAAI/article/view/10804.

Hang Zhang, Yeyun Gong, Yelong Shen, Jiancheng Lv, Nan Duan, and Weizhu Chen. Adversarial retriever-ranker for dense text retrieval. *arXiv preprint arXiv:2110.03611*, 2021.

Jingqing Zhang, Yao Zhao, Mohammad Saleh, and Peter Liu. PEGASUS: Pre-training with extracted gap-sentences for abstractive summarization. In Hal Daumé III and Aarti Singh (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 11328–11339. PMLR, 13–18 Jul 2020. URL https://proceedings.mlr.press/v119/zhang20ae.html.

Saizheng Zhang, Emily Dinan, Jack Urbanek, Arthur Szlam, Douwe Kiela, and Jason Weston. Personalizing dialogue agents: I have a dog, do you have pets too? In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2204–2213, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-1205. URL https://www.aclweb.org/anthology/P18-1205.

Tianyi Zhang*, Varsha Kishore*, Felix Wu*, Kilian Q. Weinberger, and Yoav Artzi. Bertscore: Evaluating text generation with bert. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=SkeHuCVFDr.

Yao Zhao, Xiaochuan Ni, Yuanyuan Ding, and Qifa Ke. Paragraph-level neural question generation with maxout pointer and gated self-attention networks. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 3901–3910, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1424. URL https://www.aclweb.org/anthology/D18-1424.

Ming Zhong, Pengfei Liu, Yiran Chen, Danqing Wang, Xipeng Qiu, and Xuanjing Huang. Extractive summarization as text matching. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 6197–6208, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.552. URL https://aclanthology.org/2020.acl-main.552.

A DISCUSSION

In this section, we discuss the relations between our JGR and several popular methods, including reinforcement learning (RL), generative adversarial networks (GAN), and data augmentation.

A.1 JGR & RL

Some previous RL works, i.e., (Shen et al., 2015; Rennie et al., 2017; Paulus et al., 2018) proposed to use $\Delta(\hat{\mathbf{y}}, \mathbf{y})$ to compute reward $\mathcal{R}(\hat{\mathbf{y}})$ directly which doesn't combine ranking scores as feedback signals. However, we argue that the ranking score calculated by the ranker model can provide more semantic-relevant information than the matching scores (e.g., BLEU or ROUGE) which are purely based on the surface match. In the ablation study, we also demonstrate that the proposed approach is superior to other configurations in terms of training stability and performance.

Some other RL works Bahdanau et al. (2017); Le et al. (2022) introduced actor-critic frameworks (Konda & Tsitsiklis, 1999), which jointly train an actor and a critic, are similar to our JGR framework. while they have not considered the contrastive rewards between different candidates given one input. However, different from these works, JGR allows the generator to sample several i.i.d. candidates and be optimized simultaneously on these candidates at each training step. This improvement makes the reward of a sampled candidate contain contrastive information from the candidates from the same candidate set. Furthermore, it effectively raises the number of diverse chains of probabilities on which the generator can be optimized. In Table 9, we compare our JGR-G with the simple RL baseline that uses metric rewards, and the actor-critic baseline that the critic is trained to fit the metric score $\Delta(\hat{\mathbf{y}}, \mathbf{y})$. The empirical results show that trained with the JGR framework, the generator model can surpass those trained with previous RL-based methods well used in the NLG area.

Table 9: Performance of generators in JGR and two kinds of RL-based method.

	R-1	R-2	R-L	AVG
BART	44.16	21.28	40.90	35.45
RL (self-critical)	44.14	21.20	40.95	35.43
Actor-critic	45.04	21.99	41.71	36.25
JGR-G	**46.86**	**23.18**	**43.74**	**37.93**

A.2 JGR & GAN

From the perspective of the composition of a framework, both JGR and GAN contain a generator and a critic. In GAN, the critic is the discriminator that aims at discriminating the real candidate from the candidate pool. While in JGR, the critic is the ranker that aims to re-rank the candidates generated by the generator.

The main difference between JGR and GAN comes from the training objective. Let the G_θ denotes the generator, and D_ϕ denotes the discriminator/ranker. GAN trains G_θ and D_ϕ with the min-max objective:

$$\mathcal{J}_{G_\theta D_\phi} = \min_\theta \max_\phi E_{\mathbf{y}^+ \sim p_{\text{true}}(\cdot|\mathbf{x})}[\log p_{D_\phi}(\mathbf{y}^+, \mathbf{x})] + E_{\hat{\mathbf{y}}^- \sim p_{G_\theta}(\cdot|\mathbf{x})}[\log(1 - p_{D_\phi}(\hat{\mathbf{y}}^-, \mathbf{x}))] \quad (10)$$

In Eq 10, $\mathbf{y}^+$ is the ground-truth output of input $\mathbf{x}$, and $\hat{\mathbf{y}}^-$ is the candidate texts sampled by the generator. This is different from the setting of JGR, where both $\mathbf{y}^+$ (denoted as $\hat{\mathbf{y}}^+$ in JGR) and $\hat{\mathbf{y}}^-$ are sampled from $p_{G_\theta}(\cdot|\mathbf{x})$.

To implement GAN in NLG, according to Yu et al. (2017), the policy gradient is used and the reward assigned to $\hat{\mathbf{y}}^-$ is $\log p_{D_\phi}(\hat{\mathbf{y}}^-, \mathbf{x})$. Note that the reward is always positive, therefore GAN essentially raises the probability of the generator outputs, regardless of the quality of the outputs. On contrary, as computed in Eq. 8, there are both positive and negative rewards in JGR, which means that JGR not only encourages the generator to generate good candidates but also punishes the generator when generating bad candidates.

Table 10 shows the performance of generators in JGR and GAN on CNN/DailyMail, where GAN_{std} is the standard GAN setting that $\mathbf{y}^+$ is the ground-truth text and GAN_{mod} is our modified version

Table 10: Results generator in JGR and two kinds of GANs.

	R-1	R-2	R-L	AVG
BART	44.16	21.28	40.90	35.45
GAN_{std}	43.68	20.81	40.45	34.98
GAN_{mod}	42.93	20.66	39.87	34.49
JGR-G	**46.86**	**23.18**	**43.74**	**37.93**

of GAN that $\mathbf{y}^+$ is replaced by the best candidate sampled by the generator, i.e., $\hat{\mathbf{y}}^+$. As shown in the table, our JGR surpasses the GAN methods, and the performance of GAN_{std} and GAN_{mod} can not even surpass the model trained on optimizing the standard NLL loss, indicating that the GAN methods are not suitable for all NLG tasks. The GAN_{mod} performs worse than GAN_{std}, showing that for the min-max objective of GAN, it is not a good choice to letting $\hat{\mathbf{y}}^+$ as the positive sample, which is contrary to what we found in JGR.

A.3 JGR & Data Augmentation

Data augmentation methods aim to improve the models' performance by adding modified or synthesized data to the existing training data (Li et al., 2022a). For natural language generation tasks, denote the augmented dataset as $\hat{\mathcal{D}}$, where $\hat{\mathcal{D}}$ contains several augmented samples $(\hat{\mathbf{x}}, \hat{\mathbf{y}})$, the training object for model in the augmented data is:

$$\mathcal{L}_{\text{DA}} = -\sum_{(\hat{\mathbf{x}},\hat{\mathbf{y}})\in\hat{\mathcal{D}}} \sum_{t} \log p_{G_\theta}(\hat{y}_t | \hat{y}_{<t}, \hat{\mathbf{x}}) \tag{11}$$

The above equation is similar to JGR's reinforcement learning loss in Eq 8. Both of them optimize the generator by maximizing the log-likelihood of synthesized data. Therefore, from this perspective, we can regard our JGR as a way of data augmentation where the synthesized data is sampled from the generator and the log-likelihood is re-scaled by the rewards.

Table 11: Results generator in JGR and two kinds of GANs.

	R-1	R-2	R-L	AVG
BART	44.16	21.28	40.90	35.45
DA_{sep}	44.37	21.24	41.18	35.60
DA_{mix}	44.27	21.38	41.04	35.56
JGR-G	**46.86**	**23.18**	**43.74**	**37.93**

We designed two simple but effective data augmentation methods named DA_{sep} and DA_{mix}. Both of DA_{sep} and DA_{mix} use a fine-tuned generator G^0 to generate one summary $\hat{\mathbf{y}}$ for each input $\mathbf{x}$ in original training set $\mathcal{D}$ using beam search, the collection of all $(\mathbf{x}, \hat{\mathbf{y}})$ is treat as the augmented training data $\hat{\mathcal{D}}$. After that, 1) DA_{sep} fine-tunes G^0 firstly on $\hat{\mathcal{D}}$ and then on $\mathcal{D}$, 2) DA_{mix} further fine-tunes G^0 on the mixture of $\hat{\mathcal{D}}$ and $\mathcal{D}$. We compare the performance of DA_{sep} and DA_{mix} with our JGR on CNN/DailyMail, with BART as the generator, the results are shown in Table 11. As shown in the results, both DA_{sep} and DA_{mix} can further improve the performance of BART, verifying the effect of data augmentation. However, the performance gain brought by data augmentation is far less than that brought by JGR.

B Computation of Self-BLEU

Given a candidate set $\hat{\mathcal{Y}} = \{\hat{\mathbf{y}}^1, \hat{\mathbf{y}}^2, ..., \hat{\mathbf{y}}^C\}$ sampled from the generator, the self-BLEU score for $\hat{\mathcal{Y}}$ is computed as the average of mutual BLEU scores of all candidate pairs:

$$\text{self-BLEU}(\hat{\mathcal{Y}}) = \frac{\sum_{\hat{\mathbf{y}}^i,\hat{\mathbf{y}}^j\in\hat{\mathcal{Y}};i\neq j} \text{BLEU}(\hat{\mathbf{y}}^i, \hat{\mathbf{y}}^j)}{C(C-1)} \tag{12}$$

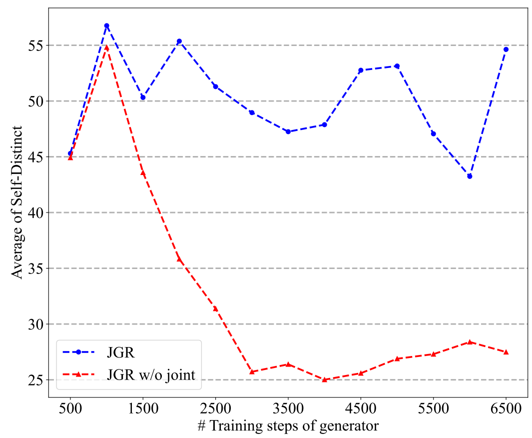

Figure 5: The average of self-Distinct-2 at each training interval.

A higher self-BLEU score means the sampled candidates are more similar to each other, in other words, a lower diversity of the sampled candidates.

It is another way to assess the diversity of sampled candidates by computing the proportion of the number of distinct n-grams in the total number of tokens for the sampled candidates of an input sequence. We refer to this metric as self-Distinct-n where n refers to n-grams. The higher self-Distinct-n corresponds to the higher diversity of sampled candidates. Like Figure 4, we show the curves of the average self-Distinct-2 for generated candidates at each training interval in Figure 5. From the figure, we can see that the self-Distinct-2 of $\text{JGR}_{\text{w/o joint}}$ drops rapidly after the generator is trained 1000 steps, while the self-Distinct-2 keeps hovering in a relatively high range for JGR. This phenomenon aligns with what we found when applying self-BLEU and further enhances our conclusion in Section 4.7.

C Evaluating JGR on Advanced Metrics

A model trained with RL objective may succeed in the metrics it uses as the reward function but perform poorly in other metrics. We hope to investigate whether JGR, which uses the RL objective to train its generator, suffers from the same problem. We use three advanced metrics, namely BERTScore (Zhang* et al., 2020), FactCC (Kryscinski et al., 2020), and QuestEval (Scialom et al., 2021a), to evaluate JGR on CNN/DailyMail. BERTScore measures the semantic similarity of the predicted summary and ground-truth reference. FactCC and QuestEval use a trained language model to measure the factual consistency between the generated summary and input source document. The results are shown in Table 12.

Table 12: Performance on BERTScore, FactCC, and QuestEval.

	BERTScore	FactCC	QE
BART	88.47	57.54	59.41
JGR-G	88.90	60.33	61.53
JGR-R	88.96	61.59	61.62

As shown in the table, JGR-G and JGR-R both achieve higher BERTScore than BART, indicating that they can generate summaries with better semantic quality. For FactCC and QuestEval, which measure factual consistency, JGR-G and JGR-R also surpass the BART baseline. These results demonstrate that JGR, despite optimizing for ROUGE, does not compromise on other aspects of summary quality, such as semantic similarity and factual consistency, and can even improve on them compared to the generator trained with NLL.

Moreover, JGR is not limited to ROUGE and BLEU as the ranking criteria. It could potentially perform better if we incorporate other metrics like FactCC into $\Delta(\hat{\mathbf{y}}, \mathbf{y})$. We plan to explore this direction in the future.

D Decoding Strategies

We study the impact of different decoding strategies during inference. Two decoding strategies are compared, namely beam search and group beam search (Vijayakumar et al., 2016). We also compare different beam sizes. The results of ROUGE-1 score with beam search on CNN/DailyMail are shown in Figure 6.

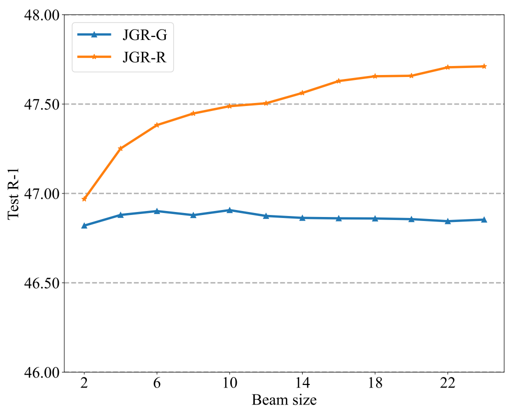

Figure 6: ROUGE-1 score using beam search with different on CNN/DailyMail test set.

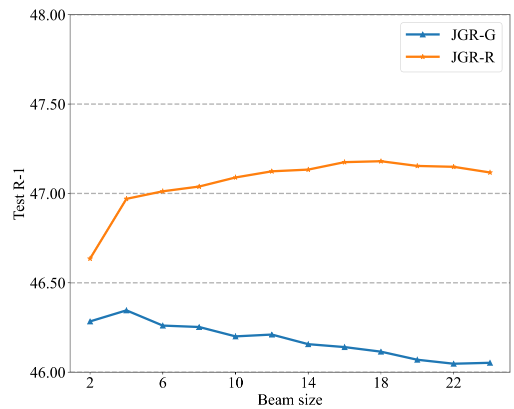

Figure 7: ROUGE-1 score using diverse beam search with different on CNN/DailyMail test set.

As shown in Figure 6, increasing the beam size does not contribute to the performance of JGR-G when using the normal beam search. However, the performance of JGR-R can rise as the beam size increases. This indicates that increasing the beam size can raise the probability of JGR-R ranking a better candidate to the top among all the candidates decoded by JGR-G.

Figure 7 shows the results with diverse beam search. Firstly we can find that with diverse beam search the JGR system can not achieve comparable results with JGR using normal beam search, and the performance of JGR-G begins to drop when beam size exceeds 4. We can still observe that the performance of JGR-R rises as the beam size increases. However, since the performance of JGR-G keeps declining, the performance ascent of JGR-R is not as significant as that of JGR-R with the normal beam search.

E Statistics of the Benchmarks

The statistics of all benchmarks are shown in Table 13.

Table 13: The statistics of the benchmarks. |Src.| means the average number of tokens for each source input. |Tgt.| means the average number of tokens for each target text.

Benchmark	|Train|	|Dev|	|Test|	|Src.|	|Tgt.|
CNN/DailyMail	287,113	13,368	11,490	822.3	57.9
SAMSum	14,731	818	819	124.1	23.4
SQuAD 1.1	75,722	10,570	11,877	149.4	11.5
PersonaChat	122,499	14,602	14,056	120.8	11.8

For evaluation on CNN/Daily and SAMSum, we use the python rouge score package: `https://pypi.org/project/rouge-score/`. For evaluation on SQuAD 1.1, we follow the evaluation scripts open-sourced by Liu et al. (2020) at `https://github.com/microsoft/ProphetNet/tree/master/GLGE_baselines/script/script/evaluate/qg`. For evaluation on PersonaChat, we follow the evaluation scripts open-sourced by Liu et al. (2020) at `https://github.com/microsoft/ProphetNet/tree/master/GLGE_baselines/script/script/evaluate/personachat`.

Table 14: The hyper-parameters of JGR on each benchmark.

	CNN/DailyMain	SAMSum	SQuAD 1.1	PersonaChat
		Warming-up G^0		
# Epochs	5	5	20	5
Learning rate	5e-5	5e-5	5e-5	5e-5
Batch size	96	128	96	96
Max source length	1024	1024	600	700
Max target length	100	100	65	70
		First Ranker training iteration		
# Epochs	3	20	3	3
Learning rate	1e-5	1e-5	1e-5	1e-5
Warm-up ratio/steps	0.2	500 steps	0.2	0.3
Batch size	64	64	64	32
Max source length	512	512	500	500
# Candidates sampled for G^0		16		
# Negative candidates		2		
$\Delta(\hat{\mathbf{y}}, \mathbf{y})$	0.02(R-1)+0.05(R-2)+0.025(R-L)		0.02(R-L)+0.04(B-4)+0.04(MTR)	0.02(B-1)+0.025(B-2)
		JGR training		
# Epochs	3	10	3	3
# JGR-R steps per iteration	500	231 steps (1 epoch)	250	500
# JGR-G steps per iteration	500	231 steps (1 epoch)	250	500
JGR-G learning rate	5e-5	1e-5	5e-5	5e-5
JGR-R learning rate	1e-5	5e-6	1e-5	1e-5
Batch size	64	64	32	64
# Candidates sampled for JGR-R		16		
# Negative candidates for JGR-R		2		
# Candidates sampled for JGR-G		8		
Beam size when inference		16		
$\Delta(\hat{\mathbf{y}}, \mathbf{y})$	0.02(R-1)+0.05(R-2)+0.025(R-L)		0.02(R-L)+0.04(B-4)+0.04(MTR)	0.02(B-1)+0.025(B-2)

F Hyper-parameters of Fine-tuning on Benchmarks.

The hyper-parameters for our JGR on each benchmark are shown in Table 14.