ONE2SET + Large Language Model: Best Partners for Keyphrase Generation

Liangying Shao[1,2*] **Liang Zhang**[1*] **Minlong Peng**[3] **Guoqi Ma**[1]
Hao Yue[1] **Mingming Sun**[4] **Jinsong Su**[1,2†]
[1]School of Informatics, Xiamen University, China
[2]Key Laboratory of Digital Protection and Intelligent Processing
of Intangible Cultural Heritage of Fujian and Taiwan (Xiamen University),
Ministry of Culture and Tourism, China
[3]Cognitive Computing Lab, Baidu Research, China
[4]Beijing Institute of Mathematical Sciences and Applications, China
{liangyingshao,lzhang}@stu.xmu.edu.cn, jssu@xmu.edu.cn

Abstract

Keyphrase generation (KPG) aims to automatically generate a collection of phrases representing the core concepts of a given document. The dominant paradigms in KPG include ONE2SEQ and ONE2SET. Recently, there has been increasing interest in applying large language models (LLMs) to KPG. Our preliminary experiments reveal that it is challenging for a single model to excel in both recall and precision. Further analysis shows that: 1) the ONE2SET paradigm owns the advantage of high recall, but suffers from improper assignments of supervision signals during training; 2) LLMs are powerful in keyphrase selection, but existing selection methods often make redundant selections. Given these observations, we introduce a *generate-then-select* framework decomposing KPG into two steps, where we adopt a ONE2SET-based model as *generator* to produce candidates and then use an LLM as *selector* to select keyphrases from these candidates. Particularly, we make two important improvements on our generator and selector: 1) we design an Optimal Transport-based assignment strategy to address the above improper assignments; 2) we model the keyphrase selection as a sequence labeling task to alleviate redundant selections. Experimental results on multiple benchmark datasets show that our framework significantly surpasses state-of-the-art models, especially in absent keyphrase prediction. We release our code at https://github.com/DeepLearnXMU/KPG-SetLLM.

1 Introduction

The keyphrase generation (KPG) task involves creating a set of phrases to encapsulate the core concepts of given document. High-quality keyphrases enhance various downstream tasks, such as information retrieval (Kim et al., 2013; Tang et al., 2017; Boudin et al., 2020), text summarization (Wang and Cardie, 2013; Pasunuru and Bansal, 2018). In general, keyphrases are categorized into two types: 1) *present keyphrases* that occur continuously in the given document, and 2) *absent keyphrases* that do not match any continuous subsequence. The quality evaluation of keyphrases includes two aspects: *precision*, which requires the generated keyphrases to be pertinent to the document, and *recall*, which demands the generated keyphrases cover the core ideas of the document.

Dominant paradigms for KPG include ONE2SEQ (Yuan et al., 2020) and ONE2SET (Ye et al., 2021). The former treats KPG as a sequence generation task, while the latter treats it as a set generation by introducing multiple control codes for parallel keyphrase generation and dynamically assigning keyphrase ground-truths to control codes as supervision based on bipartite matching (Kuhn, 2010). Recently, pre-trained language models (PLMs) have been widely incorporated into KPG via ONE2SEQ paradigm (Chowdhury et al., 2022; Zhao et al., 2022; Wu et al., 2023a; Dong et al., 2023). Particularly, with the emergence of LLMs, researchers have also begun to introduce LLMs into KPG via in-context learning (Song et al., 2023a; Martínez-Cruz et al., 2023). However, it is difficult for a single model to achieve high performance in precision and recall simultaneously. As verified by our preliminary study (See Section 2), models that excel in recall tend to have lower precision, while models with high precision fall short in recall.

In this paper, to deal with the above issue, we introduce a *generate-then-select* framework that de-

*Equal contribution.
†Corresponding author.
This work is done when Liangying Shao was interning at Cognitive Computing Lab, Baidu Research, China.

Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, pages 11140–11153
November 12-16, 2024

composes the KPG task into two steps, each handled by a separate sub-model. This framework includes a generator that aims to recall correct keyphrases and a selector that eliminates incorrect candidates.

To identify the most suitable models for the generator and selector, we conduct further experiments in the preliminary study to investigate the potential of conventional KPG models and LLMs for these roles. Our experimental results lead to two conclusions: 1) SETTRANS, a ONE2SET-based KPG model, has a significant advantage in recall and thus is well-suited as the generator, and 2) LLMs with their superior semantic understanding, are more effective than small language models (SLMs) for keyphrase selection and are suitable as the selector.

Furthermore, we improve the generator and selector of our framework in two aspects. The generator assigns each ground-truth to only one control code. However, the number of control codes generally exceeds that of ground-truths, leading to insufficient training for many control codes. To address the above improper assignments, we propose an OT-based assignment strategy for ONE2SET. This strategy converts the matching of candidates and ground truth into an OT problem, allowing a ground-truth to be assigned to multiple candidates.

As for the selector, existing studies (Kong et al., 2023; Choi et al., 2023; Sun et al., 2023) employ reranking methods to individually score candidates and then select those with high scores, which, however, results in many semantically similar candidates being selected. To address this issue, we convert keyphrase reranking into an LLM-based sequence labeling task. Leveraging the long sequence modeling capability of LLMs, we feed all candidates into the selector and have it autoregressively generate decision labels indicating whether to keep or discard the corresponding candidate. In this way, we can not only reduce the decoding search space of LLMs, but also alleviate semantic repetition by enabling the selector to fully consider the correlation between the current candidate and previous selections. Particularly, to ensure robustness to the order of candidates, we feed them into the selector in random order during instruction tuning. This encourages the selector to develop a deeper understanding of the candidates' semantics. During inference, candidates are sorted by quality for the selector to prioritize candidates more likely to be correct.

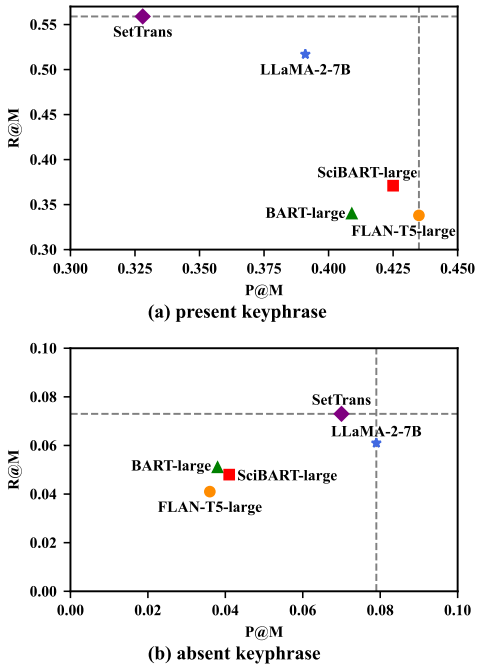

Figure 1: Performance of various models on the KP20k test set. LLaMAGen, a fine-tuned version of LLaMA-2-7B, is optimized for KPG using instruction tuning.

Overall, the major contributions of our work can be summarized as follows:

- Our in-depth analysis reveals that achieving high precision and recall simultaneously for a single model is challenging. Moreover, we find that SETTRANS is advantageous as a generator, while LLM excels as a selector.
- We design an OT-based assignment strategy to refine the training of ONE2SET and enhance our selector by converting keyphrase reranking into a sequence labeling task.
- Experimental results and in-depth analysis of several commonly-used datasets demonstrate the effectiveness of our framework, especially in absent keyphrase prediction.

2 Preliminary Study

To verify the necessity of decomposing KPG, we first explore the performance of dominant models in terms of recall and precision. Then, through more experiments, we analyze which models are best suited as the generator and selector.

Trade-off in Keyphrase Generation. We measure the performance of dominant models on the testset of KP20k, including SETTRANS, fine-tuned BART-large, SciBART-large, Flan-T5-large, and LLaMA-2-7B and report the results in Figure

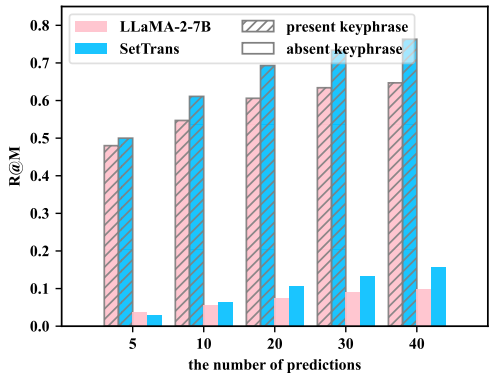

Figure 2: R@M of LLaMA-2-7B and SETTRANS when generating the same number of keyphrases.

1. As shown in Figure 1, achieving high accuracy and recall simultaneously is challenging for a single model. Whether it is a conventional KPG model or an LLM, as the number of predicted keyphrases increases, the recall of the model inevitably increases while its accuracy decreases, and vice versa. This result proves the necessity of decomposing KPG into two steps.

Evaluating the Recall Performance of LLaMA-2-7B and SETTRANS. As revealed above, SETTRANS and LLaMA-2-7B exhibit excellent recall in commonly-used setting. In Figure 2, we further investigate their recall under various settings for a full comparison. We can clearly observe that as the number of generated candidates increases, SETTRANS consistently exhibits better recall performance than LLaMA-2-7B. Given its stronger recall performance and lower computational consumption, we choose SETTRANS as the generator. Additionally, SETTRANS tends to recall more correct keyphrases along with more incorrect candidates (see Appendix B.2), highlighting the necessity of a selector with strong filtering capabilities to improve accuracy.

Evaluating SLM and LLM for Keyphrase Selection. To identify a suitable selector, we first use SETTRANS as the generator to output candidates, and then compare multiple representative keyphrase reranking methods. The methods we consider include 1) SLM-Scorer, the reranker from (Choi et al., 2023), which is an SLM-based one and achieves SOTA performance in keyphrase reranking, and 2) LLM-Scorer, a LLaMA-2-7B reranker, which is fined-tuned as detailed in Appendix D.2. As shown in Table 1, LLM-Scorer achieves higher accuracy and better F1@M scores, indicating that the powerful semantic understanding capability of LLMs is helpful for keyphrase selection. Consequently, we adopt LLaMA-2-7B as the selector.

Model	Acc		F1@M	
	Pre	Abs	Pre	Abs
SETTRANS + SLM-Scorer	0.813	0.806	0.429	0.082
SETTRANS + LLM-Scorer	**0.823**	**0.812**	**0.441**	**0.089**

Table 1: Keyphrase selection performance of SLM and LLM on the KP20k test set. **Acc** represents the accuracy of selection, while **Pre** and **Abs** stand for present and absent keyphrases, respectively.

3 Our Framework

As described above, our framework involves an improved ONE2SET-based generator and an LLM-based selector. Unlike the conventional ONE2SET paradigm, our generator improves the supervision signal assignment during training by modeling it as an Optimal Transport (OT) problem. Distinct from previous studies on keyphrase reranking (Kong et al., 2023; Choi et al., 2023) and LLM-based reranking (Qin et al., 2023; Zhuang et al., 2023), our selector autoregressively generates decision labels for keeping or discarding each candidate. This approach not only reduces the decoding search space but also fully considers the correlation between selections, thus effectively minimizing semantically repetitive selections. Moreover, we design an R-tuning S-infer strategy to help the selector comprehend the semantics of candidates.

3.1 The ONE2SET-based Generator

As an extension of SETTRANS, our generator also uses Transformer (Vaswani et al., 2017) as the backbone, of which the decoder is equipped with N control codes to individually generate candidate keyphrases. During the model training, ground-truth keyphrases $\{y_i\}_{i=1}^{M-1}$ or $\varnothing$ (y_M) are dynamically assigned to the control codes as supervision signals. Concretely, the model first predicts K tokens as the prediction $\hat{y}_j$ for the j-th control code and then calculates a matching score μ_{ij} between the ground-truth y_i and the prediction $\hat{y}_j$ via a pairwise matching function $\mathcal{C}_{match}(*)$[1]:

$$\mu_{ij} = \frac{\mathcal{C}_{match}(y_i, \hat{y}_j)^{\frac{1}{\tau}}}{\sum_{j=1}^{N} \mathcal{C}_{match}(y_i, \hat{y}_j)^{\frac{1}{\tau}}}, \quad (1)$$

where τ is a normalized hyper-parameter.

Then, instead of using bipartite matching, we consider the assignments between ground-truths

[1]The detail of $\mathcal{C}_{match}(*)$ is described in Appendix C.1.

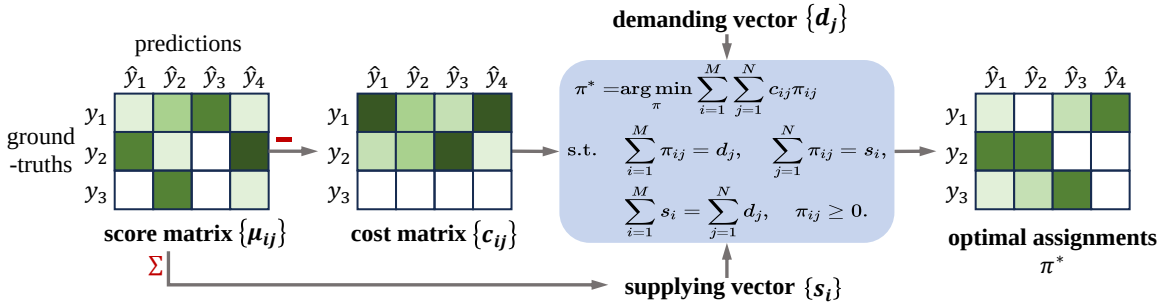

Figure 3: The OT-based supervision signal assignment for keyphrase generation. $\sum$ represents summing as Equation 2, while $-$ stands for taking the negative following Equation 3.

and control codes as an OT problem[2] and search the optimal assignments with Sinkhorn-Knopp Iteration (Cuturi, 2013). Concretely, we consider the following crucial definitions in OT algorithm: 1) control codes are regarded as *demanders* with a demanding vector $\{d_j\}_{j=1}^{N}$, where d_j represents the number of ground-truths assigned to the j-th control code; 2) ground-truths are regarded as *suppliers* with a supplying vector $\{s_i\}_{i=1}^{M}$, where s_i represents the number of control codes that y_i can be assigned to; 3) the cost matrix $\{c_{ij}\}_{i=1,j=1}^{M,N}$, where c_{ij} represents the cost of assigning y_i to the j-th control code. More specifically, we heuristically define them as follows:

- Since assigning multiple ground-truths to one control code at the same time may interfere with each other, we directly limit d_j to 1.
- Intuitively, if y_i is highly matched with more control codes, it should be assigned to more control codes. To this end, we define s_i as a dynamic number positively correlated with $\{\mu_{ij}\}_{j=1}^{N}$:

$$s_i = \begin{cases} \lceil \sum \mathrm{topK}(\{\mu_{\mathrm{ij}}\}_{\mathrm{j=1}}^{\mathrm{N}}, \mathrm{k}) \rceil, & \text{if } y_i \neq \varnothing. \\ N - \sum\limits_{y_{i'} \neq \varnothing} s_{i'}, & \text{otherwise.} \end{cases} \tag{2}$$

  where $\lceil \cdot \rceil$ indicates rounding up to an integer and k is a predefined hyper-parameter.
- To model the intuition that the higher the matching score between y_i and $\hat{y}_j$, the lower the cost for assigning y_i to the j-th control code, we define c_{ij} as

$$c_{ij} = \begin{cases} -\mu_{ij}, & \text{if } y_i \neq \varnothing. \\ 0, & \text{otherwise.} \end{cases} \tag{3}$$

Having obtained the above vectors and matrix, we seek the optimal assignments π^* according to the following objective function:

$$\begin{aligned} \pi^* &= \arg\min_{\pi} \sum_{i=1}^{M} \sum_{j=1}^{N} c_{ij} \pi_{ij}, \quad \pi \in \mathbb{R}^{M \times N} \\ \text{s.t.} \quad & \sum_{i=1}^{M} \pi_{ij} = d_j, \quad \sum_{j=1}^{N} \pi_{ij} = s_i, \\ & \sum_{i=1}^{M} s_i = \sum_{j=1}^{N} d_j, \quad \pi_{ij} \geq 0. \end{aligned} \tag{4}$$

Finally, each control code is assigned with the ground-truth or $\varnothing$ that has the maximal assignment value as shown in the right of Figure 3. Note that we seek the optimal assignment plans π_p^* for present keyphrases and π_a^* for absent keyphrases, respectively, and subsequently calculate their cross-entropy losses accordingly.

3.2 The LLM-based Selector

After using the above generator to obtain candidate keyphrases, it is natural to focus on how to select high-quality keyphrases from them. However, through in-depth analysis, we find that both traditional keyphrase reranking methods and LLM reranking methods tend to output keyphrases with serious semantic repetition. To solve this problem, we propose to utilize LLMs to model keyphrase selection as a sequence labeling task. Furthermore, we design a random-tuning sorted-inference strategy that enables the selector to improve performance while retaining robustness to input order.

Semantic Repetition As shown in Figure 4(a), existing reranking studies contain the following two types: 1) one first individually score each candidate and then keep the candidates with high scores (Choi et al., 2023; Zhuang et al., 2023), 2) the other directly ask LLMs to generate a sorted list of candidates without specific scores and save highest ranked candidates (Sun et al., 2023; Qin et al., 2023). However, when applying these meth-

[2]A detailed description of the OT problem can be found in Appendix C.2.

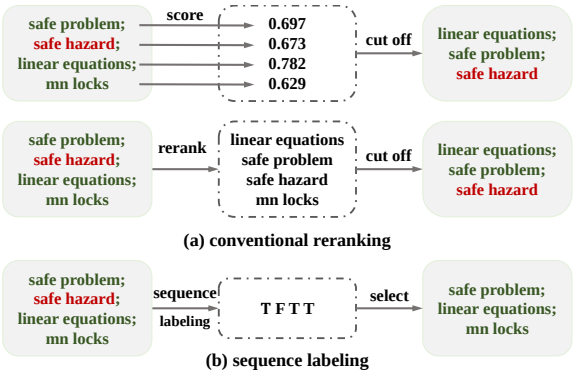

Figure 4: The comparison between the reranking methods and our sequence labeling method. A green candidate indicates a correct keyphrase, while a red candidate is an incorrect keyphrase.

ods to select keyphrases, they tend to assign similar ranks to candidates with similar semantics but different surface representations (i.e. *"safe problem"* and *"safe hazard"*).

LLM-based Sequence Labeling To address the above-mentioned issue, we fine-tune an LLM to model keyphrase reranking as a sequence labeling task. Concretely, we input all candidates into the selector and then ask it to autoregressively output decision labels, each indicating whether to keep or discard the corresponding candidate. As shown in Figure 4(b), each kept candidate is mapped to the label *"T"* while each discarded one to *"F"*. The instruction template we use is shown as follows:

Task Definition:
You are required to perform a sequence labeling task to select multiple keyphrases from the numbered candidates according to the given document. Use the label *"T"* to indicate the selection of a candidate and the label *"F"* to indicate its rejection. For instance, a label sequence *"T F F"* denotes selecting candidate [1] and rejecting candidates [2] and [3].
Input:
Document: $\{document\}$
Candidates:
[1] $\{candidate_1\}$
.....
[n] $\{candidate_n\}$
Response:
Label sequence: $\{label\_sequence\}$

During autoregressive generation, the selector considers previous selections when deciding whether to keep or discard the current candidate. This approach not only reduces the decoding search space but also alleviates semantic repetition. In the example in Figure 4(b), the selector is able to discard *"safe hazard"* after keeping *"safe problem"*.

R-tuning S-infer Strategy Intuitively, sorting candidates in a fixed order is beneficial for humans to select candidates. However, such sorting may cause the selector to select candidates based on their input order rather than truly understanding their semantics. To address this issue, we propose a *R-tuning S-infer* strategy to handle the candidates differently during the selector training and inference. Specifically, during instruction tuning, candidates are input into the selector in a random order, encouraging the selector to learn the semantics of candidates instead of order. By contrast, during inference, candidates are sorted by their quality measured with the average log probability of the generator. In this way, the selector can prioritize candidates more likely to be correct.

3.3 Two-stage Training

We adopt a two-stage training strategy to train our framework, where the generator is first trained with the keyphrase generation data, and the selector is then trained with the instruction data.

Generator Training With the optimal assignment plans π_p^* and π_a^* (See Section 3.1), we compute the cross-entropy losses for absent and present keyphrases, respectively, and combine these two losses with weighted summation to get the final loss. Please refer to Appendix C.3 for details.

Selector Training When training the selector, we adopt the next-token-prediction task that has been widely used in LLMs. Particularly, due to the imbalanced numbers of positive and negative candidates, we design the following loss:

$$\mathcal{L}(\phi) = \frac{1}{N_T}\sum_{t=1}^{|Y|}\mathbb{I}_{\{Y_t=T\}}\log p_\phi(Y_t|X,Y_{<t}) + \frac{1}{N_F}\sum_{t=1}^{|Y|}\mathbb{I}_{\{Y_t=F\}}\log p_\phi(Y_t|X,Y_{<t}), \quad (5)$$

where ϕ represents the parameters of the selector, N_T and N_F are the numbers of *"T"* and *"F"*, respectively, Y is the label sequence consisting of

N_T *"T"* and N_F *"F"*, and X is the input excluding label sequence. The negative effect of label imbalance in tuning can be mitigated by averaging loss items with identical labels.

In our experiments, we adopt QLoRA (Dettmers et al., 2023) to perform quantization on model parameters for efficient training. As such, the trainable parameters in our model are about 0.06% of the original size.

4 Experiments

4.1 Setup

Datasets. Following previous studies (Meng et al., 2017; Ye et al., 2021; Choi et al., 2023), we use the training set of KP20k to train all models and then evaluate them on five benchmarks[3]: Inspec (Hulth, 2003), Krapivin (Krapivin et al., 2009), NUS (Nguyen and Kan, 2007), SemEval (Kim et al., 2010), KP20k (Meng et al., 2017).

Baselines. We compare our framework with three kinds of baselines: 1) *Generative Models*: these models predict both present and absent keyphrases through generation. We consider following representative models, **catSeq** (Yuan et al., 2020) under ONE2SEQ along with its variant **ExHiRD-h** (Chen et al., 2020), and **SetTrans** (Ye et al., 2021) under ONE2SET along with its variant **WR-one2set** (Xie et al., 2022). Besides, since PLM has been widely applied in KPG, we also consider two competitive models, **CorrKG** (Zhao et al., 2022) and **SciBART-large + TAPT + DESEL** (Wu et al., 2023a). 2) *Unified Models*: these models integrate extractive and generative methods to predict keyphrases. We report the performance of the representative models including **SEG-Net** (Ahmad et al., 2021), **UniKeyphrase** (Wu et al., 2021), **PromptKP** (Wu et al., 2022) and **SimCKP** (Choi et al., 2023). 3) *Composite Models*: We additionally select several representative models combined like our framework.

Evaluation Metrics. As implemented in previous studies (Chan et al., 2019; Zhao et al., 2022; Choi et al., 2023), We evaluate all models using macro-average F1@M, and further provide the F1@5 results in Appendix A. Both predictions and ground-truths are stemmed with the Porter Stemmer (Porter, 2006), and then the duplicates are removed before scoring.

[3]https://huggingface.co/memray

Implementation Details. We separately use Transformer-base (Vaswani et al., 2017) and LLaMA-2-7B (Touvron et al., 2023) to construct the generator and selector, both are optimized with Adam optimizer (Loshchilov and Hutter, 2019).

When constructing the generator, we select the top 50,002 frequent tokens to build the vocabulary. To ensure the consistency with (Ye et al., 2021; Xie et al., 2022), the number of control codes N is 20, K is 2, learning rate is 0.0001, and batch size is 12. Through grid search in Appendix B.3, we set the following hyper-parameters in OT-based assignment: $\tau = 10$ in Equation 1 and Top-3 in Equation 2. During inference, we employ beam search with beam size = 10 and save all candidates for the subsequent selection.[4]

As for the selector, we adopt QLoRA with r = 8, α = 32, and dropout of 0.05. Note that, due to the significant performance gap between present keyphrases and absent keyphrases, we use the same instruction template to tune a LoRA module for each type of keyphrase. The LoRA is optimized with a learning rate of 3e-4 for absent keyphrase, a learning rate of 1e-4 for present keyphrase, per-gpu batch size of 24, and the maximum epoch of 5. Validations are performed every 1,000 iterations for present keyphrase and 400 iterations for absent keyphrase, respectively. Early stopping is triggered if the validation performance does not improve in 5 consecutive rounds. We save the model with the best F1@M score on validation set for testing. Particularly, we perform experiments with three random seeds and report the average results.

4.2 Main Results

The comparison results on the five testsets are shown in Table 2. As for the present keyphrase prediction, our framework significantly outperforms all baselines on all datasets, except for Inspec. In contrast, on the absent keyphrase prediction, our framework always performs best among all models. Note that as an extension of SciBART-large, +TAPT+DESEL is additionally trained in OAGKX (Çano and Bojar, 2020), which leads to its huge improvement on Inspec. Compared to other baselines, our framework still holds comparable performance on this dataset. Our genera-

[4]The experiments on the impact of different beam sizes during inference are presented in Appendix B.4

Model	Inspec		Krapivin		NUS		SemEval		KP20k	
	Pre	Abs	Pre	Abs	Pre	Abs	Pre	Abs	Pre	Abs
Unified Models										
SEG-Net (Ahmad et al., 2021)	0.265	0.015	0.366	0.036	0.397	0.036	0.283	0.030	0.367	0.036
UniKeyphrase (Wu et al., 2021)	0.288	0.036	—	—	0.443	0.056	0.322	0.052	0.352	0.068
PromptKP (Wu et al., 2022)	0.294	0.022	—	—	0.439	0.042	0.356	0.032	0.355	0.042
SimCKP (Choi et al., 2023)	0.358	0.035	0.405	0.089	0.498	0.088	0.386	0.047	0.427	0.080
Generation Models										
catSeq (Yuan et al., 2020)	0.262	0.008	0.354	0.036	0.397	0.028	0.283	0.028	0.367	0.032
ExHiRD-h (Chen et al., 2020)	0.291	0.022	0.347	0.043	—	—	0.335	0.025	0.374	0.032
SetTrans (Ye et al., 2021)	0.324	0.034	0.364	0.073	0.450	0.060	0.357	0.034	0.392	0.058
WR-ONE2SET (Xie et al., 2022)	0.351	0.034	0.362	0.074	0.452	0.071	0.370	0.043	0.378	0.064
CorrKG (Zhao et al., 2022)	0.365	0.045	—	—	0.449	0.079	0.359	0.044	0.404	0.071
SciBART-large (Wu et al., 2023a)	0.328	0.026	0.329	0.056	0.421	0.050	0.304	0.033	0.396	0.057
+ TAPT + DESEL	**0.402**	0.036	0.352	0.086	0.449	0.068	0.341	0.040	0.431	0.076
LLaMA-2-7B	0.344	0.038	0.434	0.087	0.481	0.062	0.354	0.042	0.449	0.069
Composite Models										
SciBART-large + Our selector	0.352	0.053	0.430	0.098	0.521	0.084	0.392	0.046	0.445	0.076
WR-ONE2SET + Our selector	0.350	0.059	0.430	0.123	0.523	0.118	0.398	0.057	0.448	0.108
Our generator + SLM-scorer	0.350	0.033	0.410	0.094	0.510	0.093	0.390	0.050	0.429	0.084
Our generator + Our selector	0.357_2	$\mathbf{0.064}_1$‡	‡$\mathbf{0.435}_3$‡	$\mathbf{0.126}_3$‡	$\mathbf{0.528}_2$‡	$\mathbf{0.122}_3$‡	$\mathbf{0.405}_5$‡	$\mathbf{0.058}_4$	$\mathbf{0.453}_1$‡	$\mathbf{0.112}_1$‡

Table 2: Testing results on all datasets. The best performance is boldfaced, while the second best is underlined. The subscript denotes the corresponding standard deviation (e.g., 0.112_1 indicates 0.112 ± 0.001). ‡ indicates significant at $p < 0.01$ over SimCKP with 1, 000 booststrap tests (Tibshirani and Efron, 1993).

Model	In-domain		Out-domain	
	Pre	Abs	Pre	Abs
Ours	**0.453**	**0.112**	**0.431**	**0.093**
⇒bipartite matching	0.446	0.105	0.421	0.087
⇒GenKP	0.441	0.089	0.382	0.062
⇒R-tuning R-inference	0.442	0.102	0.416	0.083
⇒S-tuning R-inference	0.264	0.060	0.243	0.048
⇒CE_loss	0.435	0.096	0.412	0.062

Table 3: Ablation study. ⇒* means replacing the corresponding component of our framework with *. We use three random orders and report the average performance in the R-inference setting.

tor combined with our selector outperforms other composite models, making it the best combination. When comparing the results of "Our generator + SLM-scorer" with "Our generator + Our selector", it becomes evident that the LLM-based selector (our selector) demonstrates powerful filtering capabilities, underscoring the semantic understanding of LLMs in keyphrase filtering. In the comparison of results for "* + Our selector", the generators based on the One2Set paradigm excel at handling absent keyphrases, with our generator achieving the best performance, indicating that the OT-based assignment strategy enhances its effectiveness.

4.3 Ablation Study

In Table 3, we investigate the effect of each component on our framework to verify their validity. Following previous studies (Xie et al., 2022; Choi et al., 2023), we conduct experiments on two kinds of test sets: 1) **in-domain**, which is KP20k, and 2) **out-of-domain**, which is the combination of Inspec, Krapivin, NUS, and SemEval.

(1) ⇒*bipartite matching*. In this variant, we replace the OT-based assignment with bipartite matching and observe the performance degradation in both present and absent keyphrases. Furthermore, we compare the recall scores of our generator, SETTRANS, and WR-ONE2SET at the same precision level. From Figure 5, the recall of our generator consistently exceeds those of (Ye et al., 2021) and (Xie et al., 2022) with close accuracy. Both experiments demonstrate the effectiveness of our OT-based assignment.

(2) ⇒*GenKP*. In this variant, we tune the selector to generate the list of kept candidates directly. Compared to our selector, its prediction performance significantly drops, especially on out-of-domain datasets. We argue that the sequence labeling task adopted by our selector reduces the decoding space, thus effectively reducing the task difficulty and improving generalization.

(3) ⇒*R-tuning R-inference*. Unlike our selector, this variant inputs candidates into the selector in a random order during both training and inference, resulting in a slight performance drop. This indi-

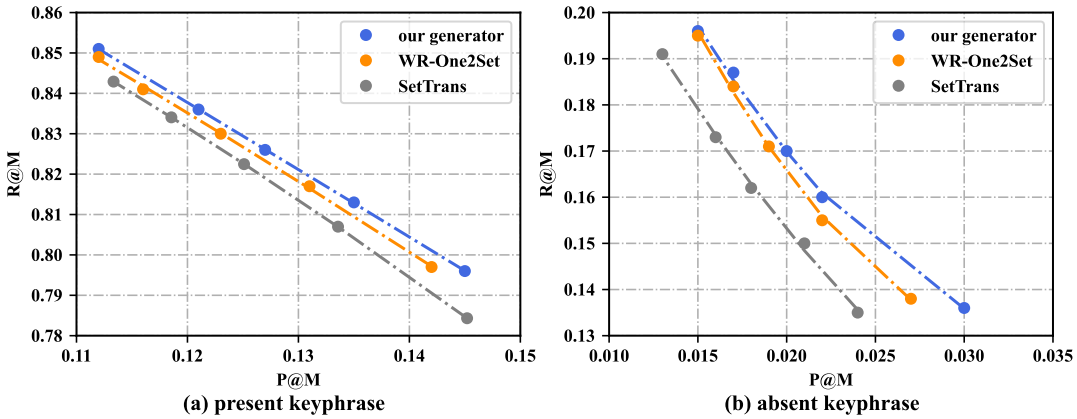

Figure 5: The recall and precision of our generator and other baselines.

cates that our sorted-inference strategy helps the selector make better selections.

(4) ⇒*S-tuning R-inference*. Different from the above variant, this variant inputs candidates into the selector in a fixed order during training while in a random order during inference. Compared to R-tuning R-inference, we observe a more significant performance degradation, suggesting that the random-tuning strategy enhances the robustness of the selector to the input order.

(5) ⇒*CE_loss*. In this variant, we tune the selector with vanilla cross-entropy loss without loss item averaging. The removal of loss item averaging notably diminishes the performance of the selector, demonstrating the effectiveness of this operation. The fact that there are more incorrect candidates than correct ones leads to the overfitting of incorrect candidates when training with vanilla cross-entropy loss.

4.4 Diversity of Predicted Keyphrases

Following Wu et al. (2023b), we take emb_sim and dup_token_ratio as the diversity metrics. As shown in Table 4, the semantic repetition in the original candidate set is severe but significantly reduced by selection models. Among these methods, our selector obtains the lowest emb_sim and dup_token_ratio, demonstrating its effectiveness in reducing semantic repetition.

Please see **Appendix B.1** for more experiments.

Model	Dup_token_ratio ↓	emb_sim ↓
Our generator	0.406	0.198
+ SLM-Scorer	0.330	0.155
+ LLM-Scorer	0.247	0.149
+ Our Selector	**0.222**	**0.147**
ground-truth	0.072	0.132

Table 4: The diversity of all keyphrases produced by various models.

5 Related Work

The related works to ours mainly include keyphrase generation and keyphrase selection.

Keyphrase Generation. Generally, KPG models are constructed under the following paradigms: 1) ONE2ONE (Meng et al., 2017), where keyphrases of each document are split and each keyphrase along with the document forms a training instance. During inference, top-K candidates are picked under beam search. 2) ONE2SEQ (Yuan et al., 2020), which treats KPG as a sequence generation task, concatenating keyphrases into a sequence according to a predefined order. 3) ONE2SET (Ye et al., 2021), which generates keyphrases as an unordered set conditioned on learnable control codes. Among these paradigms, ONE2SET excels in recall. Ye et al. (2021) utilize the bipartite matching to assign ground-truths or ∅ to control codes as the supervision signal. Furthermore, Xie et al. (2022) propose a re-assignment mechanism to refine the assignment results of the bipartite matching, which allows a proportion of control codes matched with ∅ to learn ground-truths.

Keyphrase Selection. Currently, researchers (Song et al., 2021; Zhang et al., 2022; Kong et al., 2023) select keyphrases from candidates using reranking methods. The common practice is to perform phrase mining on n-grams within document to extract candidates. Recently, Choi et al. (2023)

obtain present keyphrase candidates through data mining and absent keyphrase candidates from the KPG model. All the above methods individually score each candidate with PLMs and select those with high scores. However, this independent scoring leads to semantic repetition issue. With the rapid development of LLMs, researchers try to insert the candidates into prompt and instruct the LLMs to generate an ordered list (Sachan et al., 2022; Sun et al., 2023; Zhuang et al., 2023; Qin et al., 2023), which demonstrates impressive effectiveness in document reranking tasks.

Overall, our work differs from previous studies for two main reasons. First, unlike (Xie et al., 2022), we treat the matching of control codes and ground-truths as an OT problem and propose an OT-based assignment strategy to refine the target assignment in the ONE2SET paradigm. Second, in contrast to current selection methods, we consider keyphrase selection as an LLM-based sequence labeling task, where the correlation between the current candidate and previous selections can be fully exploited.

6 Conclusion and Future Work

This paper introduces a generate-then-select framework that integrates a ONE2SET model and an LLM selector together, so as to fully leverage the high recall of the ONE2SET paradigm and powerful semantic understanding of LLM. The ONE2SET model acts as the generator and is optimized by our OT-based assignment to recall more correct candidates. The LLM acts as the selector that models the selection of keyphrase candidates as a sequence labeling task and reduces the semantic repetition through its long sequence modeling capability. Experimental results show that our framework achieves significant performance improvements compared to existing state-of-the-art models.

In the future, we tend to combine the generation and selection tasks into a multi-task learning framework, which further improves the synergy between the two tasks.

Acknowledgments

The project was supported by National Natural Science Foundation of China (No. 62276219), and the Public Technology Service Platform Project of Xiamen (No. 3502Z20231043). We also thank the reviewers for their insightful comments.

Limitations

While this paper introduces a generate-then-select framework for KPG that effectively combines the strengths of the ONE2SET paradigm and LLM, it has several limitations in terms of resource consumption. First, the LLM is inherently resource-intensive due to its large number of parameters, demanding significant computational power and memory. Second, the two-step process of generating and then selecting keyphrases is time-consuming, which can lead to relative inefficiency in practical applications. These factors combined make the proposed framework more resource-consuming and challenging to implement compared to single-model solutions.

References

Wasi Uddin Ahmad, Xiao Bai, Soomin Lee, and Kai-Wei Chang. 2021. Select, extract and generate: Neural keyphrase generation with layer-wise coverage attention. In *ACL*.

Florian Boudin, Ygor Gallina, and Akiko Aizawa. 2020. Keyphrase generation for scientific document retrieval. In *ACL*.

Erion Çano and Ondrej Bojar. 2020. Two huge title and keyword generation corpora of research articles. In *LREC*.

Hou Pong Chan, Wang Chen, Lu Wang, and Irwin King. 2019. Neural keyphrase generation via reinforcement learning with adaptive rewards. In *ACL*.

Wang Chen, Hou Pong Chan, Piji Li, and Irwin King. 2020. Exclusive hierarchical decoding for deep keyphrase generation. In *ACL*.

Minseok Choi, Chaeheon Gwak, Seho Kim, Si Hyeong Kim, and Jaegul Choo. 2023. Simckp: Simple contrastive learning of keyphrase representations. In *Findings of EMNLP*.

Md. Faisal Mahbub Chowdhury, Gaetano Rossiello, Michael R. Glass, Nandana Mihindukulasooriya, and Alfio Gliozzo. 2022. Applying a generic sequence-to-sequence model for simple and effective keyphrase generation. *CoRR*, abs/2201.05302.

Marco Cuturi. 2013. Sinkhorn distances: Lightspeed computation of optimal transport. In *NeurIPS*.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. Qlora: Efficient finetuning of quantized llms. In *NeurIPS*.

Yifan Dong, Suhang Wu, Fandong Meng, Jie Zhou, Xiaoli Wang, Jianxin Lin, and Jinsong Su. 2023. Towards better multi-modal keyphrase generation via visual entity enhancement and multi-granularity image noise filtering. In *ACM MM*.

Anette Hulth. 2003. Improved automatic keyword extraction given more linguistic knowledge. In *EMNLP*.

Su Nam Kim, Olena Medelyan, Min-Yen Kan, and Timothy Baldwin. 2010. Semeval-2010 task 5 : Automatic keyphrase extraction from scientific articles. In *SemEval@ACL*.

Youngsam Kim, Munhyong Kim, Andrew Cattle, Julia Otmakhova, Suzi Park, and Hyopil Shin. 2013. Applying graph-based keyword extraction to document retrieval. In *IJCNLP*.

Aobo Kong, Shiwan Zhao, Hao Chen, Qicheng Li, Yong Qin, Ruiqi Sun, and Xiaoyan Bai. 2023. Promptrank: Unsupervised keyphrase extraction using prompt. In *ACL*.

Mikalai Krapivin, Aliaksandr Autaeu, Maurizio Marchese, et al. 2009. Large dataset for keyphrases extraction.

Harold W. Kuhn. 2010. The hungarian method for the assignment problem. In *50 Years of Integer Programming 1958-2008 - From the Early Years to the State-of-the-Art*.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *ICLR*.

Roberto Martínez-Cruz, Alvaro J. López-López, and José Portela. 2023. Chatgpt vs state-of-the-art models: A benchmarking study in keyphrase generation task. *CoRR*, abs/2304.14177.

Roberto Martínez-Cruz, Alvaro J López-López, and José Portela. 2023. Chatgpt vs state-of-the-art models: a benchmarking study in keyphrase generation task. *arXiv preprint arXiv:2304.14177*.

Rui Meng, Sanqiang Zhao, Shuguang Han, Daqing He, Peter Brusilovsky, and Yu Chi. 2017. Deep keyphrase generation. In *ACL*.

Thuy Dung Nguyen and Min-Yen Kan. 2007. Keyphrase extraction in scientific publications. In *ICADL*.

Ramakanth Pasunuru and Mohit Bansal. 2018. Multi-reward reinforced summarization with saliency and entailment. In *NAACL*.

Martin F. Porter. 2006. An algorithm for suffix stripping. *Program*.

Zhen Qin, Rolf Jagerman, Kai Hui, Honglei Zhuang, Junru Wu, Jiaming Shen, Tianqi Liu, Jialu Liu, Donald Metzler, Xuanhui Wang, and Michael Bendersky. 2023. Large language models are effective text rankers with pairwise ranking prompting. *CoRR*, abs/2306.17563.

Devendra Singh Sachan, Mike Lewis, Mandar Joshi, Armen Aghajanyan, Wen-tau Yih, Joelle Pineau, and Luke Zettlemoyer. 2022. Improving passage retrieval with zero-shot question generation. In *EMNLP*.

Mingyang Song, Haiyun Jiang, Shuming Shi, Songfang Yao, Shilong Lu, Yi Feng, Huafeng Liu, and Liping Jing. 2023a. Is chatgpt A good keyphrase generator? A preliminary study. *CoRR*, abs/2303.13001.

Mingyang Song, Haiyun Jiang, Shuming Shi, Songfang Yao, Shilong Lu, Yi Feng, Huafeng Liu, and Liping Jing. 2023b. Is chatgpt a good keyphrase generator? a preliminary study. *arXiv preprint arXiv:2303.13001*.

Mingyang Song, Liping Jing, and Lin Xiao. 2021. Importance estimation from multiple perspectives for keyphrase extraction. In *EMNLP*.

Weiwei Sun, Lingyong Yan, Xinyu Ma, Shuaiqiang Wang, Pengjie Ren, Zhumin Chen, Dawei Yin, and Zhaochun Ren. 2023. Is chatgpt good at search? investigating large language models as re-ranking agents. In *EMNLP*.

Yixuan Tang, Weilong Huang, Qi Liu, Anthony K. H. Tung, Xiaoli Wang, Jisong Yang, and Beibei Zhang. 2017. Qalink: Enriching text documents with relevant q&a site contents. In *CIKM*.

Robert J Tibshirani and Bradley Efron. 1993. An introduction to the bootstrap. *Monographs on statistics and applied probability*, 57(1):1–436.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton-Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurélien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open foundation and fine-tuned chat models. *CoRR*, abs/2307.09288.

Lewis Tunstall, Edward Beeching, Nathan Lambert, Nazneen Rajani, Kashif Rasul, Younes Belkada, Shengyi Huang, Leandro von Werra, Clémentine

Fourrier, Nathan Habib, Nathan Sarrazin, Omar Sanseviero, Alexander M. Rush, and Thomas Wolf. 2023. Zephyr: Direct distillation of LM alignment. *CoRR*, abs/2310.16944.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *NeurIPS*.

Lu Wang and Claire Cardie. 2013. Domain-independent abstract generation for focused meeting summarization. In *ACL*.

Di Wu, Wasi Uddin Ahmad, and Kai-Wei Chang. 2023a. Rethinking model selection and decoding for keyphrase generation with pre-trained sequence-to-sequence models. In *EMNLP*.

Di Wu, Da Yin, and Kai-Wei Chang. 2023b. Kpeval: Towards fine-grained semantic-based evaluation of keyphrase extraction and generation systems. *CoRR*, abs/2303.15422.

Huanqin Wu, Wei Liu, Lei Li, Dan Nie, Tao Chen, Feng Zhang, and Di Wang. 2021. Unikeyphrase: A unified extraction and generation framework for keyphrase prediction. In *Findings of ACL*.

Huanqin Wu, Baijiaxin Ma, Wei Liu, Tao Chen, and Dan Nie. 2022. Fast and constrained absent keyphrase generation by prompt-based learning. In *AAAI*.

Binbin Xie, Jia Song, Liangying Shao, Suhang Wu, Xiangpeng Wei, Baosong Yang, Huan Lin, Jun Xie, and Jinsong Su. 2023. From statistical methods to deep learning, automatic keyphrase prediction: A survey. *IPM*.

Binbin Xie, Xiangpeng Wei, Baosong Yang, Huan Lin, Jun Xie, Xiaoli Wang, Min Zhang, and Jinsong Su. 2022. Wr-one2set: Towards well-calibrated keyphrase generation. In *EMNLP*.

Jiacheng Ye, Tao Gui, Yichao Luo, Yige Xu, and Qi Zhang. 2021. One2set: Generating diverse keyphrases as a set. In *ACL*.

Xingdi Yuan, Tong Wang, Rui Meng, Khushboo Thaker, Peter Brusilovsky, Daqing He, and Adam Trischler. 2020. One size does not fit all: Generating and evaluating variable number of keyphrases. In *ACL*.

Linhan Zhang, Qian Chen, Wen Wang, Chong Deng, Shiliang Zhang, Bing Li, Wei Wang, and Xin Cao. 2022. Mderank: A masked document embedding rank approach for unsupervised keyphrase extraction. In *Findings of ACL*.

Guangzhen Zhao, Guoshun Yin, Peng Yang, and Yu Yao. 2022. Keyphrase generation via soft and hard semantic corrections. In *EMNLP*.

Shengyao Zhuang, Honglei Zhuang, Bevan Koopman, and Guido Zuccon. 2023. A setwise approach for effective and highly efficient zero-shot ranking with large language models. *CoRR*, abs/2310.09497.

Model	Inspec		Krapivin		NUS		SemEval		KP20k	
	Pre	Abs	Pre	Abs	Pre	Abs	Pre	Abs	Pre	Abs
SimCKP	**0.358**	0.035	0.405	0.089	0.498	0.088	0.386	0.047	0.427	0.080
Our generator + Our selector	0.357	**0.064**	**0.435**	**0.126**	**0.528**	**0.122**	**0.405**	**0.058**	**0.453**	**0.112**

Table 5: F1@5 results on all datasets.

Model	Inspec		Krapivin		NUS		SemEval		KP20k	
	Pre	Abs	Pre	Abs	Pre	Abs	Pre	Abs	Pre	Abs
SimCKP (Choi et al., 2023)	0.358	0.035	0.405	0.089	0.498	0.088	0.386	0.047	0.427	0.080
SciBART-large (Wu et al., 2023a)	0.328	0.026	0.329	0.056	0.421	0.050	0.304	0.033	0.396	0.057
+ TAPT + DESEL	**0.402**	0.036	0.352	0.086	0.449	0.068	0.341	0.040	0.431	0.076
Zephyr-7B	0.358	0.035	0.428	0.092	0.479	0.058	0.353	0.047	0.443	0.072
Our generator + Our selector (Zephyr-7B)	0.374	**0.058**	**0.430**	**0.133**	**0.518**	**0.114**	**0.401**	**0.067**	**0.448**	**0.114**

Table 6: F1@M results on different LLM-based selectors.

A Experimental Results of F1@5

The F1@5 results of our generator + our selector are shown in Table 5. On this metric, our framework also outperforms the strongest baseline, proving its effectiveness.

B Further Experiments

B.1 Experimental Results on More LLMs

More Results on Open LLMs As described previously, we mainly conduct experiments on LLaMA-2-7b, due to its widespread use in the research community. To verify the validity of our framework on other LLMs, we report the results of experiments using Zephyr-7B (Tunstall et al., 2023). As shown in Table 6, our framework still achieves better performance to other baselines, suggesting that our framework is not sensitive to the choice of LLMs.

More Results on Close LLMs We conduct experiments to report the few-shot performance of GPT-4 as the generator and selector, respectively. The experiment results on the top-100 samples of each dataset are shown Table 7. Consistent with previous studies (Song et al., 2023b; Martínez-Cruz et al., 2023; Xie et al., 2023), GPT-4 also excels in the present keyphrase prediction of Inspec dataset. However, on other datasets, GPT-4 exhibits significantly worse performance than Our generator + Our selector, highlighting the effectiveness of our framework. Furthermore, when replacing our selector with GPT-4, Our generator + GPT-4 selector is still inferior to Our generator + Our selector on most datasets, demonstrating that One2set generator and LLM-based selector are the best combination for keyphrase generation.

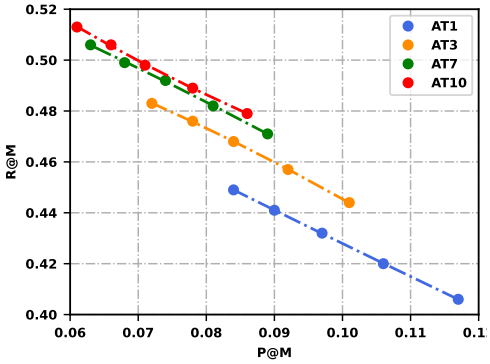

Figure 6: The recall and precision of our generator over different τ on KP20k validation set.

B.2 Recall and Precision of SetTrans with Different Numbers of Prediction

As mentioned in Section 2, SetTrans tends to recall more correct keyphrases along with more incorrect candidates. We report the recall and precision of SetTrans in Table 8. As the number of predicted keyphrases increases, recall greatly improves, but precision significantly drops, indicating that more incorrect candidates are generated. This underscores the urgent need for a strong selector to improve accuracy.

B.3 Effect of τ

We conduct experiments on various τ for smoothing the matching scores among predictions and ground-truth keyphrases and report the precision and recall of the generator under various beam sizes. As shown in Figure 6, the generator achieves the best performance at 10. Therefore, we adopt $\tau = 10$ in all experiments.

Model	Inspec@100		Krapivin@100		NUS@100		SemEval@100		KP20k@100	
	Pre	Abs	Pre	Abs	Pre	Abs	Pre	Abs	Pre	Abs
SimCKP (Choi et al., 2023)	0.342	0.046	0.397	0.101	0.482	0.064	0.387	0.051	0.420	0.086
LLaMA-2-7B	0.307	0.048	0.453	0.073	0.498	0.042	0.372	0.034	0.413	0.056
GPT-4	**0.513**	0.055	0.343	0.010	0.244	0.037	0.356	0.024	0.224	0.015
Our generator + GPT-4 selector	0.435	0.067	0.435	0.077	0.400	0.077	0.354	0.034	0.309	0.038
Our generator + Our selector	0.327	**0.069**	**0.457**	**0.142**	**0.499**	**0.136**	**0.405**	**0.058**	**0.424**	**0.097**

Table 7: F1@M results on GPT-4.

Model	Present		Absent		Num
	$P@M$	$R@M$	$P@M$	$R@M$	
SetTrans	0.340	0.500	0.050	0.030	5
	0.274	0.611	0.066	0.064	10
	0.221	0.693	0.044	0.105	20
	0.195	0.734	0.032	0.133	30
	0.178	0.763	0.026	0.156	40

Table 8: Recall and precision of SETTRANS with different prediction numbers.

Model	$F1@M$	
	Pre	Abs
Our generator ($\#bs$ = 1) + Our selector	0.438	0.050
Our generator ($\#bs$ = 5) + Our selector	**0.457**	0.065
Our generator ($\#bs$ = 10) + Our selector	**0.457**	**0.080**
Our generator ($\#bs$ = 15) + Our selector	**0.457**	**0.080**

Table 9: Performance on KP20k validation set. $\#bs$ denotes beam size.

B.4 Effect of Different Beam Sizes

We investigate the impact of beam size on the KP20k validation set. To this end, we gradually varied beam size from 1 to 15. As shown in Table 9, both present keyphrases and absent keyphrases achieved the best performance with a beam size of 10, and the performance is maintained as the beam size increased further. Therefore, we use a beam size of 10.

C Algorithm Details

C.1 Formulation of $\mathcal{C}_{match}$

Same as (Ye et al., 2021), we generate K tokens conditioned on each control code and collect their predictive probability distributions $\{P_j\}_{j=1,2,...,N}$ and $P_j = \{p_j^t\}_{t=1,...,K}$, where p_j^t is the predictive distribution at time step t for control code j. The matching score between any pair of ground-truth y_i and candidate $\hat{y}_{\pi(i)}$ is calculated as following:

$$\mathcal{C}_{match}(y_i, \hat{y}_{\pi(i)}) = -\sum_{t=1}^{K'} \mathbb{I}_{\{y_i^t \neq \varnothing\}}\, p^t_{\pi(i)}(y_i^t) \quad (6)$$

where $K' = \min(|y_i|, K)$ and $p^t_{\pi(i)}(y_i^t)$ represents the probability of token y_i^t in $p^t_{\pi(i)}$. The scores from matching any prediction with $\varnothing$ are set to 0, which avoids interference in the assignment of valid ground-truths.

C.2 Optimal Transport

Assume a scenario involving m suppliers and n demanders, where the i-th supplier holds s_i units of goods and the j-th demander needs d_j units of goods. Every route between a supplier and demanders has a per-unit transportation cost, denoted by c_{ij}. The objective of Optimal Transport (OT) is to seek the most efficient distribution plan $\pi^* = \{\pi_{ij} | i = 1, 2, \ldots, m, j = 1, 2, \ldots, n\}$, which minimizes the total cost of transporting all goods from suppliers to demanders.

An equivalent mathematical formulation of the OT problem is presented as follows:

$$\begin{aligned} \min_{\pi} \quad & \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}\pi_{ij} \\ \text{s.t.} \quad & \sum_{i=1}^{m} \pi_{ij} = d_j, \quad \sum_{j=1}^{n} \pi_{ij} = s_i, \\ & \sum_{i=1}^{m} s_i = \sum_{j=1}^{n} d_j, \quad \pi_{ij} \geq 0, \\ & i = 1, 2, \ldots, m, \quad j = 1, 2, \ldots, n. \end{aligned}$$

In order to tackle the OT problem efficiently, we employ the Sinkhorn-Knopp Iterative algorithm (Cuturi, 2013), which demonstrates polynomial-time complexity.

C.3 Formulation of The Generator Loss Function

Following (Ye et al., 2021), we organize the loss of our generator as:

$$\mathcal{L}(\theta) = -\left[\sum_{i=1}^{\frac{N}{2}} \mathcal{L}^p(\theta, y_{\pi^*_p(i)}) + \sum_{i=\frac{N}{2}+1}^{N} \mathcal{L}^a(\theta, y_{\pi^*_a(i)})\right] \quad (7)$$

$$\mathcal{L}^p(\theta, y_i) = \begin{cases} \lambda_{pre} \cdot \sum_{t=1}^{|y_i|} \log \hat{p}_i^t(y_i^t), & \text{if } y_i = \varnothing. \\ \sum_{t=1}^{|y_i|} \log \hat{p}_i^t(y_i^t), & \text{otherwise.} \end{cases} \quad (8)$$

where y_i^t is the t-th token of target y_i, $\hat{p}_i^t$ denotes the predictive probability distribution of the i-th candidate at t-th step, λ_{pre} is a hyperparameter used to decrease the impact of excessive $\varnothing$. $\mathcal{L}^a(\theta, y_i)$ is defined in the same form as $\mathcal{L}^p(\theta, y_i)$, except that λ_{pre} is replaced with λ_{abs}. We adopt λ_{pre} as 0.2 and λ_{abs} as 0.1.

D Implementation Details

D.1 LLaMA-2-7B

We use LLaMA-2-7B to perform instruction tuning and use the prompt template as follows:

Instruction: Generate keyphrases for the given document and use ; to space keyphrases. For example, "phrase_A; phrase_B; phrase_C".
Input: Document: $\{document\}$
Response: Keyphrases: $\{keyphrases\}$

We adopt QLoRA and a learning rate of 2e-4. Validation is performed every 1,000 iterations. The rest of the experimental setup is consistent with 4.1.

D.2 LLM-Scorer

We use LLaMA-2-7B to perform instruction tuning and use the prompt template as follows:

Instruction:
Score each candidate according to the given document.
Input:
Document: $\{document\}$
Candidates:
[1] $\{candidate_1\}$

[n] $\{candidate_n\}$
Response:
Score: $\{scores\}$

The LLM-Scorer predicts the scores for all candidates in one step by generating a logit distribution, where $candidate_1$ maps to token "<0x00>", $candidate_2$ maps to token "<0x01>", and so forth. We extract the logits corresponding to these indices and assign these values as the scores for $candidate_1$ to $candidate_n$. The scorer is then tuned using the contrastive loss function proposed by (Choi et al., 2023), which helps to maximize the distinction between the correct and incorrect candidates by adjusting the scores accordingly.

We adopt QLoRA and a learning rate of 3e-4 to tune a LoRA module for both present and absent keyphrases. The rest of the experimental setup is consistent with 4.1.