A Practical Toolkit for Multilingual Question and Answer Generation

Asahi Ushio and **Fernando Alva-Manchego** and **Jose Camacho-Collados**
Cardiff NLP, School of Computer Science and Informatics, Cardiff University, UK
{UshioA,AlvaManchegoF,CamachoColladosJ}@cardiff.ac.uk

Abstract

Generating questions along with associated answers from a text has applications in several domains, such as creating reading comprehension tests for students, or improving document search by providing auxiliary questions and answers based on the query. Training models for question and answer generation (QAG) is not straightforward due to the expected structured output (i.e. a list of question and answer pairs), as it requires more than generating a single sentence. This results in a small number of publicly accessible QAG models. In this paper, we introduce AutoQG, an online service for multilingual QAG, along with `lmqg`, an all-in-one Python package for model fine-tuning, generation, and evaluation. We also release QAG models in eight languages fine-tuned on a few variants of pre-trained encoder-decoder language models, which can be used online via AutoQG or locally via `lmqg`. With these resources, practitioners of any level can benefit from a toolkit that includes a web interface for end users, and easy-to-use code for developers who require custom models or fine-grained controls for generation.

1 Introduction

Question and answer generation (QAG) is a text generation task seeking to output a list of question-answer pairs based on a given paragraph or sentence (i.e. the context). It has been used in many NLP applications, including unsupervised question answering modeling (Lewis et al., 2019; Zhang and Bansal, 2019; Puri et al., 2020), fact-checking (Ousidhoum et al., 2022), semantic role labeling (Pyatkin et al., 2021), and as an educational tool (Heilman and Smith, 2010; Lindberg et al., 2013). The most analysed setting in the literature, however, has been question generation (QG) with pre-defined answers, as this simplifies the task and makes the evaluation more straightforward.

Despite its versatility, QAG remains a challenging task due to the difficulty of generating compositional outputs containing a list of question and answer pairs as shown in Figure 1, with recent works mainly relying on extended pipelines that include several ad-hoc models (Lewis et al., 2021; Bartolo et al., 2021). These works integrate QAG into their in-house software, preventing models to be publicly released, and their complex pipelines make them hard to reproduce and use by practitioners.

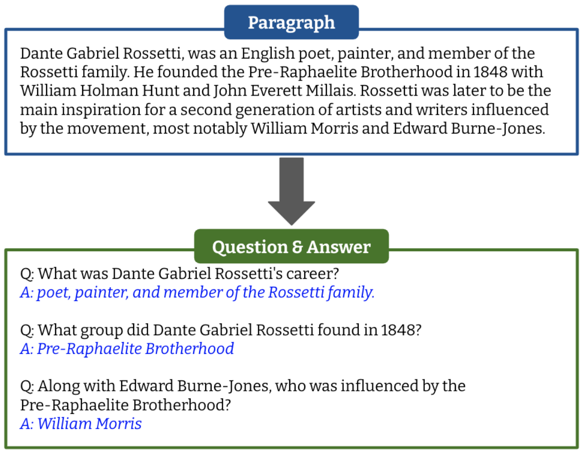

Figure 1: An example of question and answer generation given a paragraph as context.

In this paper, we introduce an open set of software tools and resources to assist on the development and employment of QAG models for different types of users. We publicly release the following resources:[1]

- `lmqg`,[2] a Python package for QAG model fine-tuning and inference on encoder-decoder language models (LMs), as well as evaluation scripts, and a deployment API hosting QAG models for developers;

[1] All the resources except for the datasets are released under an open MIT license, while the datasets follow the license of their original release.

[2] https://github.com/asahi417/lm-question-generation

Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics
Volume 3: System Demonstrations, pages 86–94
July 10-12, 2023 ©2023 Association for Computational Linguistics

- 16 models for English, and three diverse models for each of the seven languages integrated into our library, all fine-tuned on QG-Bench (Ushio et al., 2022) and available on the HuggingFace hub (Wolf et al., 2020);[3]
- *AutoQG* (`https://autoqg.net`), a website where developers and end users can interact with our multilingual QAG models.

2 Resources: Models and Datasets

Our QAG toolkit makes use of pre-existing models and datasets, fully compatible with the HuggingFace hub. This makes our library easily extendable in the future as newer datasets and better models emerge. In this section, we describe the datasets (§ 2.1) and models (§ 2.2) currently available through `lmqg` and AutoQG.

2.1 Multilingual Datasets

Our toolkit integrates all QG datasets available in QG-Bench (Ushio et al., 2022). QG-Bench is a multilingual QG benchmark consisting of a suite of unified QG datasets in different languages. In particular, we integrate the following datasets: SQuAD (English), SQuADShifts (Miller et al., 2020) (English), SubjQA (Bjerva et al., 2020) (English), JAQuAD (So et al., 2022) (Japanese), GerQuAD (Möller et al., 2021) (German), SberQuAd (Efimov et al., 2020) (Russian), KorQuAD (Lim et al., 2019) (Korean), FQuAD (d'Hoffschmidt et al., 2020) (French), Spanish SQuAD (Casimiro Pio et al., 2019) (Spanish), and Italian SQuAD (Croce et al., 2018) (Italian). QG-Bench is available through our official `lmqg` HuggingFace project page and GitHub[4].

2.2 Models

Aiming to make QAG models publicly accessible in several languages, we used `lmqg` to fine-tune LMs using QG-Bench (§ 2.1). First, we defined a pipeline QAG model architecture consisting of two independent models: one for answer extraction (AE) and one for question generation (QG). During training, the AE model learns to find an answer in each sentence of a given paragraph, while the QG model learns to generate a question given an answer from a paragraph. To generate question-answer pairs at generation time, the AE model first extracts answers from all the sentences in a given paragraph, and then these are used by the QG model to generate a question for each answer. While not directly evaluated in this paper, we also integrated other types of QAG methods such as multitask and end2end QAG (Ushio et al., 2023), all available via the `lmqg` library (§ 3) as well as AutoQG (§ 5).

As pre-trained LMs, we integrated T5 (Raffel et al., 2020), Flan-T5 (Chung et al., 2022), and BART (Lewis et al., 2020) for English; and mT5 (Xue et al., 2021) and mBART (Liu et al., 2020) for non-English QAG models. The pre-trained weights were taken from checkpoints available in the HuggingFace Hub as below:

- `t5-{small,base,large}`
- `google/flan-t5-{small,base,large}`
- `facebook/bart-{base,large}`
- `google/mt5-{small,base}`
- `facebook/mbart-large-cc25`

All the fine-tuned QAG models are publicly available in our official HuggingFace Hub. While we initially integrated these models, users can easily fine-tune others using `lmqg`, as we show in § 3.

3 `lmqg`: An All-in-one QAG Toolkit

In this section, we introduce `lmqg` (**L**anguage **M**odel for **Q**uestion **G**eneration), a Python library for fine-tuning LMs on QAG (§ 3.1), generating question-answer pairs (§ 3.2), and evaluating QAG models (§ 3.3). Additionally, with `lmqg`, we build a REST API to host QAG models to generate question and answer interactively (§ 5). `lmqg` is interoperable with the HuggingFace ecosystem, as it can directly make use of the datasets and models already shared on the HuggingFace Hub.

3.1 QAG Model Fine-tuning

Fine-tuning is performed via `GridSearcher`, a class to run encoder-deocoder LM fine-tuning with hyper-parameter optimization (see Appendix A for more details). For example, the following code shows how we can fine-tune T5 (Raffel et al., 2020) on SQuAD (Rajpurkar et al., 2016), with the QAG model explained in § 2.2. Since we decomposed QAG into AE and QG, two models need to be fine-tuned independently.

```
from lmqg import GridSearcher
```

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

[4] https://github.com/asahi417/lm-question-generation/blob/master/QG_BENCH.md

```
# instantiate AE trainer
trainer_ae = GridSearcher(
  dataset_path="lmqg/qg_squad",
  input_types="paragraph_sentence",
  output_types="answer",
  model="t5-large")

# train AE model
trainer_ae.train()

# instantiate QG trainer
trainer_qg = GridSearcher(
  dataset_path="lmqg/qg_squad",
  input_types="paragraph_answer",
  output_types="question",
  model="t5-large")

# train QG model
trainer_qg.train()
```

The corresponding dataset, `lmqg/qg_squad`,[5] has as columns: `paragraph_answer` (answer-highlighted paragraph), `paragraph_sentence` (sentence-highlighted paragraph), `question` (target question), and `answer` (target answer). The input and the output to the QG model are `paragraph_answer` and `question`, while those to the AE model are `paragraph_sentence` and `answer`. The inputs and the outputs can be specified by passing the name of each column in the dataset to the arguments, `input_types` and `output_types` when instantiating `GridSearcher`.

3.2 QAG Model Generation

In order to generate question-answer pairs from a fine-tuned QAG model, `lmqg` provides the `TransformersQG` class. It takes as input a path to a local model checkpoint or a model name on the HuggingFace Hub in order to generate predictions in a single line of code. The following code snippet shows how to generate a list of question and answer pairs with the fine-tuned QAG model presented in § 2.2. `TransformersQG` decides which model to use for each of AE and QG based on the arguments `model_ae` and `model`.

```
from lmqg import TransformersQG

# instantiate model
model = TransformersQG(
  model="lmqg/t5-base-squad-qg",
  model_ae="lmqg/t5-base-squad-ae"
)

# input paragraph
x = """William Turner was an English
painter who specialised in watercolour
landscapes. One of his best known
pictures is a view of the city of
Oxford from Hinksey Hill."""

# generation
model.generate_qa(x)
[
 (
  "Who was an English painter
   specialised in watercolour
   landscapes?",
  "William Turner"
 ),
 (
  "Where is William Turner's
   view of Oxford?",
  "Hinksey Hill."
 )
]
```

[5]https://huggingface.co/datasets/lmqg/qg_squad

3.3 QAG Model Evaluation

Similar to other text-to-text generation tasks, we implement an evaluation mechanism that compares the set of generated question-answer pairs $\tilde{\mathcal{Q}}_p = \{(\tilde{q}^1, \tilde{a}^1), (\tilde{q}^2, \tilde{a}^2), \dots\}$ to a reference set of gold question-answer pairs $\mathcal{Q}_p = \{(q^1, a^1), (q^2, a^2), \dots\}$ given an input paragraph p. Let us define a function to evaluate a single question-answer pair to its reference pair as

$$d_{q,a,\tilde{q},\tilde{a}} = s\big(t(q,a), t(\tilde{q},\tilde{a})\big) \quad (1)$$

$$t(q,a) = \text{“}\texttt{question:}\{q\}\texttt{, answer:}\{a\}\text{''} \quad (2)$$

where s is a reference-based metric, and we compute the F_1 score as the final metric as below:

$$F_1 = 2\frac{R \cdot P}{R + P} \quad (3)$$

$$R = \text{mean}\Big(\Big[\max_{(q,a)\in\mathcal{Q}_c}\big(d_{q,a,\tilde{q},\tilde{a}}\big)\Big]_{(\tilde{q},\tilde{a})\in\tilde{\mathcal{Q}}_c}\Big) \quad (4)$$

$$P = \text{mean}\Big(\Big[\max_{(\tilde{q},\tilde{a})\in\tilde{\mathcal{Q}}_c}\big(d_{q,a,\tilde{q},\tilde{a}}\big)\Big]_{(q,a)\in\mathcal{Q}_c}\Big) \quad (5)$$

Conceptually, the recall (4) and precision (5) computations attempt to "align" each generated question-answer pair to its "most relevant" reference pair. As with traditional precision and recall metrics, precision is aimed at evaluating whether the predicted question-answer pairs are *correct* (or in this case, aligned with the reference question-answer pairs), and recall tests whether there are enough high-quality question-answer pairs. Thus, we refer to the score in (3) as the **QAAligned F1 score**. The quality of the alignment directly depends on the underlying metric s. Furthermore, the complexity of QAAligned is no more than the complexity of the underlying metric, and invariant to the order of generated pairs because of the alignment at computing recall and precision.

Out-of-the-box, `lmqg` implements two variants based on the choice of `base_metric` s (used for evaluation in § 4): QAAligned BS using BERTScore (Zhang et al., 2019) and QAAligned MS using MoverScore (Zhao et al., 2019). We selected these two metrics as they correlate well with human judgements in QG (Ushio et al., 2022). Nevertheless, the choice of `base_metric` is flexible and users can employ other natural language generation (NLG) evaluation metrics such as BLEU4 (Papineni et al., 2002), METEOR (Denkowski and Lavie, 2014), or ROUGE_L (Lin, 2004).

With `lmqg`, QAAligned score can be computed with the `QAAlignedF1Score` class as shown in the code snippet below:

```
from lmqg import QAAlignedF1Score

# gold reference and generation
ref = [
"question: What makes X?, answer: Y",
"question: Who made X?, answer: Y"]
pred = [
"question: What makes X?, answer: Y",
"question: Who build X?, answer: Y",
"question: When X occurs?, answer: Y"]

# compute QAAligned BS
scorer = QAAlignedF1Score(
  base_metric="bertscore")
scorer.get_score(pred, ref)

# compute QAAligned MS
scorer = QAAlignedF1Score(
  base_metric="moverscore"
)
scorer.get_score(pred, ref)
```

Model	QAAligned BS	QAAligned MS
BART_BASE	92.8 / 93.0 / 92.8	64.2 / 64.1 / 64.5
BART_LARGE	**93.2** / **93.4** / **93.1**	**64.8** / 64.6 / **65.0**
T5_SMALL	92.3 / 92.5 / 92.1	63.8 / 63.8 / 63.9
T5_BASE	92.8 / 92.9 / 92.6	64.4 / 64.4 / 64.5
T5_LARGE	93.0 / 93.1 / 92.8	64.7 / **64.7** / 64.9
Flan-T5_SMALL	92.3 / 92.1 / 92.5	63.8 / 63.8 / 63.8
Flan-T5_BASE	92.6 / 92.5 / 92.8	64.3 / 64.4 / 64.3
Flan-T5_LARGE	92.7 / 92.6 / 92.9	64.6 / **64.7** / 64.5

Table 1: QAAligned scores ($F_1/P/R$) on the test set of SQuAD dataset by different QAG models, where the best score in each metric is shown in boldface.

	Language	QAAligned BS	QAAligned MS
mT5_SMALL	German	**81.2 / 80.0 / 82.5**	**54.3 / 54.0 / 54.6**
	Spanish	79.9 / 77.5 / 82.6	54.8 / 53.3 / 56.5
	French	**79.7 / 77.6 / 82.1**	**53.9 / 52.7 / 55.3**
	Italian	81.6 / 81.0 / **82.3**	**55.9** / 55.6 / **56.1**
	Japanese	79.8 / 76.8 / 83.1	55.9 / 53.8 / 58.2
	Korean	80.5 / 77.6 / 83.8	**83.0 / 79.4 / 87.0**
	Russian	77.0 / 73.4 / 81.1	55.5 / 53.2 / 58.3
mT5_BASE	German	76.9 / 76.3 / 77.6	53.0 / 52.9 / 53.1
	Spanish	**80.8 / 78.5 / 83.3**	**55.3 / 53.7 / 57.0**
	French	68.6 / 67.6 / 69.7	47.9 / 47.4 / 48.4
	Italian	**81.7 / 81.3** / 82.2	55.8 / **55.7** / 56.0
	Japanese	**80.3 / 77.1 / 83.9**	**56.4 / 54.0 / 59.1**
	Korean	77.3 / 76.4 / 78.3	77.5 / 76.3 / 79.0
	Russian	77.0 / 73.4 / 81.2	55.6 / 53.3 / 58.4
mBART	German	0 / 0 / 0	0 / 0 / 0
	Spanish	79.3 / 76.8 / 82.0	54.7 / 53.2 / 56.4
	French	75.6 / 74.0 / 77.2	51.8 / 51.0 / 52.5
	Italian	40.1 / 40.4 / 39.9	27.8 / 28.1 / 27.5
	Japanese	76.7 / 74.8 / 78.9	53.6 / 52.3 / 55.1
	Korean	**80.6 / 77.7 / 84.0**	82.7 / 79.0 / **87.0**
	Russian	**79.1 / 75.9 / 82.9**	**56.3 / 54.0 / 58.9**

Table 2: QAAligned scores ($F_1/P/R$) on the test set of QG-Bench by different QAG models, where the best score in each language is shown in boldface.

4 Evaluation

We rely on the QAG models and datasets included in the library (see § 2). The individual QG components of each model (i.e. the generation of a question given an answer in a paragraph) were extensively evaluated in Ushio et al. (2022). For this evaluation, therefore, we focus on the quality of the predicted questions and answers given a paragraph (i.e. the specific answer is not pre-defined). For each model, we fine-tune, make predictions and compute their QAAligned scores via `lmqg`.

4.1 Results

Monolingual evaluation (English). Table 1 presents the test results on SQuAD for seven English models based on BART, T5 and Flan-T5. The QAG model based on BART_LARGE proves to be the best aligned with gold reference question and answers among most of the metrics. As with other QG experiments and NLP in general, the larger models prove more reliable.

Multilingual evaluation. Table 2 shows the test results of three multilingual models (mBART, mT5_SMALL and mT5_BASE) in seven languages other than English, using their corresponding language-specific SQuAD-like datasets in QG-Bench for fine-tuning and evaluation.[6] In this evaluation, no single LM produces the best results across the board, yet QAG models based on mT5_SMALL and mT5_BASE are generally better than those based on mBART.

[6]The result of mBART in German is zero. Upon further inspection, we found that the fine-tuned answer extraction module did not learn properly, probably due to the limited size of the German dataset. T5 models, however, proved more reliable in this case.

Gold	BART_B	BART_L	T5_S	T5_B	T5_L	Flan-T5_S	Flan-T5_B
4.9	4.1	4.2	4.2	4.3	4.3	4.2	4.3

Table 3: Average number of generated question and answer pairs per paragraph on the test set of SQuAD by different QAG models.

Language	Gold	mT5_SMALL	mT5_BASE	mBART
German	4.6	10.1	8.4	0.0
Spanish	1.3	4.6	4.8	4.7
French	1.3	4.9	3.6	5.4
Italian	3.8	4.7	4.6	2.5
Japanese	1.3	6.6	6.8	3.6
Korean	1.3	6.7	6.3	6.7
Russian	1.3	4.8	4.9	4.7

Table 4: The averaged number of generated question and answer pairs per paragraph on the test set of QG-Bench for each language.

4.2 Number of Generated Questions and Answers

Table 3 and Table 4 show the averaged number of generated question-answer pairs and compare it to the number in the gold dataset. For English, there is a small difference across all QAG models, with all generating fewer pairs than the gold dataset, but with a limited margin. For other languages, however, there are clear differences across QAG models, with the numbers of question-answer pairs generated by the QAG models always being larger than those in the gold dataset. When comparing the number of pairs generated by the QAG models with their QAAligned scores, in languages such as German, Spanish, and Korean, QAG models that generated a larger number question-answer pairs achieved higher scores, not only recall-wise but also generally for F1.

5 AutoQG

Finally, we present AutoQG (https://autoqg.net), an online QAG demo where users can generate question-answer pairs for texts in eight languages (English, German, Spanish, French, Italian, Japanese, Korean, Russian) by simply providing a context document. We deploy the QAG models described in § 2. In addition to the features described above, the online demo shows perplexity computed via lmppl,[7] a Python library to compute perplexity given any LM architecture. This feature helps us provide a ranked list of generation to the user. Although we can compute perplexity for non-English

[7]https://pypi.org/project/lmppl

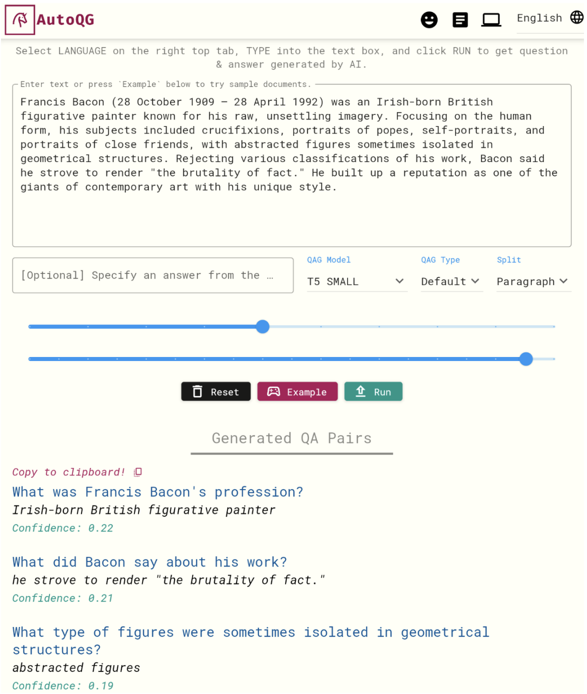

Figure 2: A screenshot of AutoQG with an example of question and answer generation over a paragraph.

generations based on the QAG models in each language, it entails large memory requirements on the the hosting server. As such, we compute a lexical overlap between the question and the document as a computationally-light alternative to the perplexity, which is defined as:

$$1 - \frac{|q \cap p|}{|q|} \tag{6}$$

where $|\cdot|$ is the number of characters in a string, and $q \cap p$ is the longest sub-string of the question q matched to the paragraph p.

Figure 2 and Figure 3 show examples of the interface with English and Japanese QAG, where there is a tab to select QAG models, language, and parameters at generation including the beam size and the value for nucleus sampling (Holtzman et al., 2020). Optionally, users can specify an answer and generate a single question on it with the QG model, as shown in Figure 4. A short introduction video to AutoQG is available at https://youtu.be/T6G-D9JtYyc.

6 Conclusion

In this paper, we introduced lmqg, a Python package to fine-tune, evaluate and deploy QAG models with a few lines of code. The library implements the QAG task as an efficient integration of answer

Figure 3: A screenshot of AutoQG with an example of question and answer generation over a paragraph in Japanese.

Figure 4: A screenshot of AutoQG when an answer is specified by the user.

extraction and question generation, and includes automatic reference-based metrics for model evaluation. Finally, we showcase AutoQG, an online demo where end users can benefit from QAG models without any programming knowledge. AutoQG enables the selection of features going from different models and languages to controlling the diversity of the generation.

Limitations

The focus on this paper was introducing software to make QAG models available to as many practitioners as possible, but there are a couple of limitations in the models and evaluation metrics we proposed.

First, our released QAG models assume a paragraph up to around 500 tokens as an input, and longer documents can not be directly fed into the models. Additionally, the released QAG models were fine-tuned on questions that require one-hop reasoning only, so they are unable to generate multi-hop reasoning.

Second, the QAAligned score is a framework to extend any NLG metric to match the prediction to the reference when they are different in size, where we employed two well-established metrics (BERTScore and MoverScore) as underlying metrics. Since those underlying metrics are already proven to be effective (Zhang et al., 2019; Zhao et al., 2019; Ushio et al., 2022), we have not conducted any human annotation for QAG specifically. Nonetheless, an extended human evaluation could help provide more insights on other limitations of the model not detected by the automatic evaluation.

Ethics Statement

While the QAG models are fine-tuned on pre-trained language models, which are known to contain some toxic contents (Schick et al., 2021), an internal check does not reveal any toxic generation. However, there is a potential risk that the QAG model could generate toxic text due to the underlying LMs.

References

Max Bartolo, Tristan Thrush, Robin Jia, Sebastian Riedel, Pontus Stenetorp, and Douwe Kiela. 2021. Improving question answering model robustness with synthetic adversarial data generation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 8830–8848, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Johannes Bjerva, Nikita Bhutani, Behzad Golshan, Wang-Chiew Tan, and Isabelle Augenstein. 2020. SubjQA: A Dataset for Subjectivity and Review Comprehension. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 5480–5494, Online. Association for Computational Linguistics.

Carrino Casimiro Pio, Costa-jussa Marta R., and Fonollosa Jose A. R. 2019. Automatic Spanish Translation of the SQuAD Dataset for Multilingual Question Answering. *arXiv e-prints*, page arXiv:1912.05200v1.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Eric Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. 2022. Scaling instruction-finetuned language models. *arXiv preprint arXiv:2210.11416*.

Danilo Croce, Alexandra Zelenanska, and Roberto Basili. 2018. Neural learning for question answering in italian. In *AI*IA 2018 – Advances in Artificial Intelligence*, pages 389–402, Cham. Springer International Publishing.

Michael Denkowski and Alon Lavie. 2014. Meteor universal: Language specific translation evaluation for any target language. In *Proceedings of the Ninth Workshop on Statistical Machine Translation*, pages 376–380, Baltimore, Maryland, USA. Association for Computational Linguistics.

Martin d'Hoffschmidt, Wacim Belblidia, Quentin Heinrich, Tom Brendlé, and Maxime Vidal. 2020. FQuAD: French question answering dataset. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1193–1208, Online. Association for Computational Linguistics.

Pavel Efimov, Andrey Chertok, Leonid Boytsov, and Pavel Braslavski. 2020. Sberquad–russian reading comprehension dataset: Description and analysis. In *International Conference of the Cross-Language Evaluation Forum for European Languages*, pages 3–15. Springer.

Michael Heilman and Noah A. Smith. 2010. Good question! statistical ranking for question generation. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 609–617, Los Angeles, California. Association for Computational Linguistics.

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. 2020. The curious case of neural text degeneration. In *International Conference on Learning Representations*.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880, Online. Association for Computational Linguistics.

Patrick Lewis, Ludovic Denoyer, and Sebastian Riedel. 2019. Unsupervised question answering by cloze translation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4896–4910, Florence, Italy. Association for Computational Linguistics.

Patrick Lewis, Yuxiang Wu, Linqing Liu, Pasquale Minervini, Heinrich Küttler, Aleksandra Piktus, Pontus Stenetorp, and Sebastian Riedel. 2021. PAQ: 65 million probably-asked questions and what you can do with them. *Transactions of the Association for Computational Linguistics*, 9:1098–1115.

Seungyoung Lim, Myungji Kim, and Jooyoul Lee. 2019. Korquad1. 0: Korean qa dataset for machine reading comprehension. *arXiv preprint arXiv:1909.07005*.

Chin-Yew Lin. 2004. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain. Association for Computational Linguistics.

David Lindberg, Fred Popowich, John Nesbit, and Phil Winne. 2013. Generating natural language questions to support learning on-line. In *Proceedings of the 14th European Workshop on Natural Language Generation*, pages 105–114, Sofia, Bulgaria. Association for Computational Linguistics.

Yinhan Liu, Jiatao Gu, Naman Goyal, Xian Li, Sergey Edunov, Marjan Ghazvininejad, Mike Lewis, and Luke Zettlemoyer. 2020. Multilingual denoising pre-training for neural machine translation. *Transactions of the Association for Computational Linguistics*, 8:726–742.

John Miller, Karl Krauth, Benjamin Recht, and Ludwig Schmidt. 2020. The effect of natural distribution shift on question answering models. In *International Conference on Machine Learning*, pages 6905–6916. PMLR.

Timo Möller, Julian Risch, and Malte Pietsch. 2021. Germanquad and germandpr: Improving non-english question answering and passage retrieval.

Nedjma Ousidhoum, Zhangdie Yuan, and Andreas Vlachos. 2022. Varifocal question generation for fact-checking. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 2532–2544, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.

Raul Puri, Ryan Spring, Mohammad Shoeybi, Mostofa Patwary, and Bryan Catanzaro. 2020. Training question answering models from synthetic data. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 5811–5826, Online. Association for Computational Linguistics.

Valentina Pyatkin, Paul Roit, Julian Michael, Yoav Goldberg, Reut Tsarfaty, and Ido Dagan. 2021. Asking it all: Generating contextualized questions for any semantic role. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 1429–1441, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

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

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas. Association for Computational Linguistics.

Timo Schick, Sahana Udupa, and Hinrich Schütze. 2021. Self-diagnosis and self-debiasing: A proposal for reducing corpus-based bias in NLP. *Transactions of the Association for Computational Linguistics*, 9:1408–1424.

ByungHoon So, Kyuhong Byun, Kyungwon Kang, and Seongjin Cho. 2022. Jaquad: Japanese question answering dataset for machine reading comprehension. *arXiv preprint arXiv:2202.01764*.

Asahi Ushio, Fernando Alva-Manchego, and Jose Camacho-Collados. 2022. Generative language models for paragraph-level question generation. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, Abu Dhabi, U.A.E. Association for Computational Linguistics.

Asahi Ushio, Fernando Alva-Manchego, and Jose Camacho-Collados. 2023. An empirical comparison of lm-based question and answer generation methods. In *Proceedings of the 61th Annual Meeting of the Association for Computational Linguistics*, Toronto, Canada. Association for Computational Linguistics.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi 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 Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Linting Xue, Noah Constant, Adam Roberts, Mihir Kale, Rami Al-Rfou, Aditya Siddhant, Aditya Barua, and Colin Raffel. 2021. mT5: A massively multilingual pre-trained text-to-text transformer. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 483–498, Online. Association for Computational Linguistics.

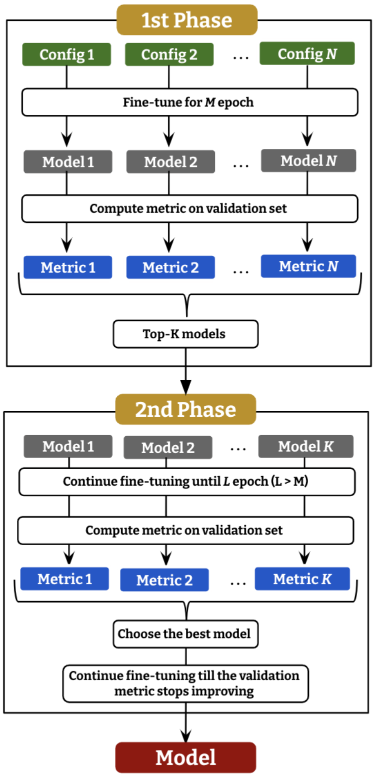

Figure 5: An overview of the hyper-parameter search implemented as `GridSearcher`.

Shiyue Zhang and Mohit Bansal. 2019. Addressing semantic drift in question generation for semi-supervised question answering. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2495–2509, Hong Kong, China. Association for Computational Linguistics.

Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q Weinberger, and Yoav Artzi. 2019. Bertscore: Evaluating text generation with bert. In *International Conference on Learning Representations*.

Wei Zhao, Maxime Peyrard, Fei Liu, Yang Gao, Christian M. Meyer, and Steffen Eger. 2019. MoverScore: Text generation evaluating with contextualized embeddings and earth mover distance. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 563–578, Hong Kong, China. Association for Computational Linguistics.

A Grid Search

To fine-tune LMs on QAG, one can use the `GridSearcher` class of `lmqg`, which performs LM fine-tuning with a two-stage optimization of hyper-parameter, a set of parameters to be used at fine-tuning such as learning rate or batch size, as described in Figure 5. Let us assume that we want to find an optimal combination of the learning rate and random seed from a list of candidates [1e-4,1e-5] and [0,1] for learning rate and random seed respectively on QG as an example. We also assume a training and a validation dataset to train a model on the task and an evaluation score that reflects a performance of a model (eg. BLEU4(Papineni et al., 2002)), and we define a search-space as a set including all the combinations of those candidates, i.e. {(1e-4, 0), (1e-4, 1), (1e-5, 0), (1e-5, 1)}. The goal of the `GridSearcher` is to find the best combination to train a model on the training dataset for the target task over the search-space with respect to the evaluation score computed on the validation dataset.

Brute-force approach such as to train model over every combination in the search-space can be a highly-inefficient, so `GridSearcher` employs a two-stage search method to avoid training for all the combinations, while being able to reach to the optimal combination as possible. To be precise, given an epoch size L (`epoch`), the first stage fine-tunes all the combinations over the search-space, and pauses fine-tuning at epoch M (`epoch_partial`). The top-K combinations (`n_max_config`) are then selected based on the evaluation score computed over the validation dataset, and they are resumed to be fine-tuned until the last epoch. Once the K chosen models are fine-tuned at second stage, the best model is selected based on the evaluation score, which is kept being fine-tuned until the evaluation score decreases.

The dataset for training and validation can be any datasets shared in the HuggingFace Hub, and one can specify the input and the output to the model from the column of the dataset by the arguments `input_types` and `output_types` at instantiating `GridSearcher`. For example, the following code shows how we can fine-tune T5 (Raffel et al., 2020) on question generation, a sub-task of QAG, with SQuAD (Rajpurkar et al., 2016), where the dataset `lmqg/qg_squad` is shared at https://huggingface.co/datasets/lmqg/qg_squad on the HuggingFace Hub, which has columns of `paragraph_answer`, that contains a answer-highlighted paragraph, and `question`, which is a question corresponding to the answer highlighted in the `paragraph_answer`. We choose them as the input and the output to the model respectively by passing the name of each column to the arguments, `input_types` and `output_types`.

```
from lmqg import GridSearcher

# instantiate the trainer
trainer = GridSearcher(
  dataset_path="lmqg/qg_squad",
  input_types="paragraph_answer",
  output_types="question",
  model="t5-large",
  batch_size=128,
  epoch=10,
  epoch_partial=2,
  n_max_config=3,
  lr=[1e-4,1e-5],
  random_seed=[0,1])

# train model
trainer.train()
```