Simple or Complex? Complexity-Controllable Question Generation with Soft Templates and Deep Mixture of Experts Model

Sheng Bi[1], Xiya Cheng[1], Yuan-Fang Li[2], Lizhen Qu[2], Shirong Shen[1]
Guilin Qi[1]*, Lu Pan[3], Yinlin Jiang[1]
[1]School of Computer Science and Engineering, Southeast University, China
[2]Faculty of Information Technology, Monash University, Melbourne, Australia
[3]Baidu Inc., China
{bisheng,chengxiya}@seu.edu.cn,{yuanfang.li,lizhen.qu}@monash.edu
{ssr,gqi}@seu.edu.cn,panlu01@baidu.com,yljiang@seu.edu.cn

Abstract

The ability to generate natural-language questions with controlled complexity levels is highly desirable as it further expands the applicability of question generation. In this paper, we propose an end-to-end neural complexity-controllable question generation model, which incorporates a mixture of experts (MoE) as the selector of soft templates to improve the accuracy of complexity control and the quality of generated questions. The soft templates capture question similarity while avoiding the expensive construction of actual templates. Our method introduces a novel, cross-domain complexity estimator to assess the complexity of a question, taking into account the passage, the question, the answer and their interactions. The experimental results on two benchmark QA datasets demonstrate that our QG model is superior to state-of-the-art methods in both automatic and manual evaluation. Moreover, our complexity estimator is significantly more accurate than the baselines in both in-domain and out-domain settings.

1 Introduction

The task of Question Generation (QG) aims at generating natural-language questions from different data sources, including passages of text, knowledge bases, images and videos. For a variety of applications, it is highly desirable to be able to *control* the complexity of generated questions. For instance, in the field of education, a well-balanced test needs questions of varying complexity levels in suitable proportions for students of different levels (Alsubait et al., 2014). That is to say, the teacher can tailor the questions to the competence of the learner. In addition, it has recently been shown (Sultan et al., 2020) that Question Answering (QA) models can benefit from training datasets enriched by applying QG models. However, despite the growing interests of answering complex questions (Cao et al., 2019) as well as questions with varying complexity levels (Seyler et al., 2017), most existing work focus on generating simple questions (Zhou et al., 2017). Although Pan et al. (2020) explored the generation of complex questions, they do not consider controlling the complexity of generated questions. Complexity-controllable question generation (CCQG) faces a number of challenges.

High diversity. Compared to simple questions, complex questions contain significantly more information and exhibit more complex syntactic structures. The complexity of questions is caused by compositional complexity because complex questions can be decomposed to a sequence of simple questions (Perez et al., 2020). Generation of both simple and complex questions imposes even higher challenges because simple and complex questions demonstrate different semantic and syntactic patterns. To this end, the resulted distributions are expected to be multimodal, i.e., with different modes for different patterns of questions.

Existing works (Gao et al., 2019; Kumar et al., 2019) fail to capture the diverse nature of CCQG. They model complexity as discrete labels, such as *easy* and *hard*, and introduce a learnable embedding as the representation of the complexity labels in the initial hidden state at the decoding stage. However, the information contained in such an embedding plays a limited role in modelling multiple modes of the underlying distribution. Similarly, it is observed that latent variables are ignored such that the posterior is always equal to the prior in variational autoencoders (Bowman et al., 2016).

Limited training data. The training of CCQG models requires questions annotated with complexity levels. However, although there are a large number of QA datasets in various domains, few of them is annotated with complexity levels. Therefore, in-domain training of high quality CCQG models becomes infeasible in most domains.

In this paper, we propose a novel question generation model, CCQG, capable of controlling ques-

Findings of the Association for Computational Linguistics: EMNLP 2021, pages 4645–4654
November 7–11, 2021. ©2021 Association for Computational Linguistics

tion complexity. We incorporate soft templates and deep *mixture of experts* (MoE) (Shen et al., 2019) to address the high diversity problem. Inspired by a recent work (Cao et al., 2018), we posit that similar questions have similar templates, and that different modes of the underlying distributions should capture different question templates. Instead of manually constructing templates, which is labor-intensive and time-consuming, we employ **soft templates**, each of which is a sequence of latent embeddings. Inspired by Cho et al (2019), we apply MoE to select templates, whereby we introduce a discrete latent variable to indicate the choice of an expert. Taking as input a complexity level, a passage and an answer, our model selects an expert, which chooses a template of that complexity level to guide the question generation process.

To address the challenge of limited training data, we design a simple and effective cross-domain complexity estimator based on five domain-independent features to classify questions w.r.t. their complexity levels. The predicted labels are incorporated into the training of CCQG. The main contributions of this work are three-folds:

- An end-to-end neural complexity-controllable QG model, which incorporates mixture of experts (MoE) and soft templates to model highly diverse questions of different complexity levels.
- A simple and effective cross-domain complexity estimator to assess the complexity of a question.
- We evaluate our CCQG model and complexity estimator on two benchmark QA datasets, SQuAD (Rajpurkar et al., 2018) and HotpotQA (Yang et al., 2018). The experimental results demonstrate that our QG model is superior to baselines in both automatic and human evaluation. The complexity estimator significantly outperforms the strong baselines with pretrained language models in both in-domain and out-domain settings. The source code will be released to encourage reproducibility.

2 Related Work

Our work is mainly relevant to question complexity estimation and question generation.

2.1 Question Complexity Estimation

Several methods have been proposed to determine the complexity of questions. (Alsubait et al., 2014) presented a similarity-based theory to control the complexity of multiple-choice questions and showed its consistency and efficiency with educational theories. (Seyler et al., 2017; Kumar et al., 2019) estimated the complexity of questions with similar manner. In general, they made statistical analysis on some features of the entities in the question, such as popularity, selectivity, and coherence, so as to evaluate the complexity. These estimation methods only focus on the questions themselves while ignoring the effect of the associated input context. Intuitively, a question has distinct complexities with different contexts.

Gao et al. (2019) evaluated the difficulty levels of questions in datasets based on whether reading comprehension systems can answer or not. This method relies heavily on the quality of QA systems and is not accurate enough. For a learner (human or machine), there are typically three iterative steps involved in answering a question, reading the passage, understanding the question, and finding the answer, which means that the complexity of a question should consider these three parts.

2.2 Question Generation

The existing work of question generation (QG) can be roughly divided into two directions, rule-based and neural-based. The former (Heilman, 2011) usually relies on manually designing lexical rules to generate questions, which is labor-intensive and has poor scalability. With the success of deep learning, many sequence-to-sequence (Seq2seq) models have been proposed for QG tasks. (Zhou et al., 2017) used enriched semantic and lexical features in QG with attention and copy mechanism (See et al., 2017). (Bi et al., 2020) designed a new reward with grammatical similarity to improve the syntactic correctness of generated question through reinforcement learning.

Due to the demand for different complexity-level questions in real scenarios, researchers began to explore generating complexity-controllable questions. (Kumar et al., 2019) used named entity popularity to estimate difficulty and generated difficulty-controllable questions. Besides, (Gao et al., 2019) evaluated the difficulty levels of questions based on QA systems and generated questions under the control of specified difficulty labels. These two models are similar in that they encode the complexity labels and use the encoded vectors as the complexity-controllable constraint. Due to the lack of parallel corpus in real scenes, which means there is only one question with “simple” or “complex” level for a pair of passage and answer. Only relying on one vector as a condition for controlling complexity, it is difficult to make the generated question conform to the given complexity constraint.

Therefore, in this paper, we propose an adaptive, generalizable complexity evaluator that considers both the question and the context while evaluating the question complexity independent of any QA system. In addition, we propose a novel model of CCQG. Compared to traditional methods that encoding complexity with only single vector as complexity constraint, we introduce mixture of experts (Cho et al., 2019) to ensure the diversity of questions with different complexity levels. We also introduce the soft template to improve the fluency of the generated questions.

3 Methodology

Given a passage, an expected answer, and a complexity level, the task of CCQG is to generate questions with the specified complexity. According to (Kunichika et al., 2002), the complexity of a question depends on two factors: i) individual capability of answering a question, and ii) the common process required to answer a question (e.g. understanding content of a question and background knowledge, steps of reasoning to infer an answer). The former varies between individual learners so that it is infeasible to find a generally applicable criterion. Despite that, we can determine the shared factors involved in the answering process and use them to quantify complexity of a question. The resulted score is then used to categorize complexity of a question. More details can be found in Sec. 4.

Formally, given a passage denoted as a word sequence $X = (x_1, \cdots, x_{n_X})$ with x_i in a vocabulary $\mathcal{V}$, a complexity level $d \in \{\text{simple}, \text{complex}\}$, an answer $A = (x_1, \cdots, x_m)$, our goal is to generate the most probable question $\hat{Y} = (y_1, \cdots, y_{n_Y})$ with $y_i \in \mathcal{V}$, which has A as its answer and the complexity level d. The estimation of complexity level d will be described in Sec. 4.

$$\hat{Y} = \arg\max_{Y} p(Y|X, A, d). \qquad (1)$$

Given the same passage, there are different ways of asking questions, which can be summarized into different *question templates* that model their semantic similarities (Cao et al., 2018) and complexity similarities. Templates provide a reference point as guidance for more nuanced question generation. Cao et al. (2018) also suggested that questions generated from templates tend to be fluent and natural.

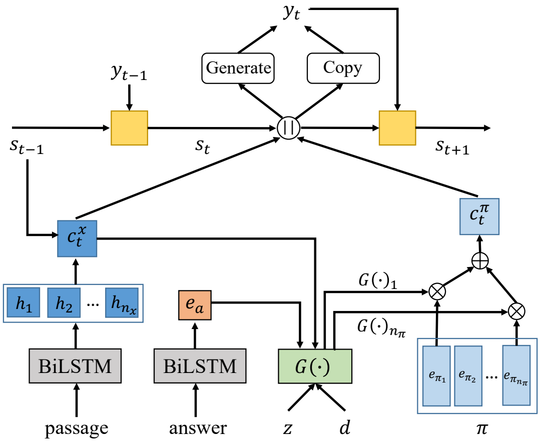


Figure 1: The overall framework of our **CCQG** model. CCQG consists of four main modules: (1) BiLSTM-based encoders of passage and answer (gray); (2) MoE-based template element selector for inputting experts and complexity and outputting probability distributions for different templates (green); (3) template element representation blocks initialized by the centroids of the question clusters at the corresponding complexity (light blue); (4) conditioned question generator (yellow).

Therefore, we argue that question generation would be more effective if the model chooses the appropriate templates at each decoding step.

Despite the usefulness of templates for question generation, template construction is labor-intensive and requires substantial domain knowledge. Therefore, template-based QG approaches typically suffer from low coverage. To alleviate this problem, we employ **soft templates** and avoid explicitly designing string-based templates. A soft template is modeled as a sequence of elements, each of which provides a reference point at a decoding step t. This modeling allows sharing of elements across templates at the same complexity level. The selection of soft templates is conducted through a mixture of experts. Each expert distinguishes from each other in terms of its preference of templates. For a given input, an expert from them is chosen to determine a probable template. Both soft templates and experts are latent. As a template is a sequence of template elements, we introduce a latent variable $\pi_t^d \in \{1, \cdots, n_\pi\}$ for the selection of template elements at the complexity level d at time t. The value of π_t^d indicates the choice of a particular template element. In the same manner, we introduce another latent variable $z \in \{1, \cdots, n_z\}$ to represent the choice of an expert for a given input. Each expert has its own dense vector representation $\mathbf{e}_z$. At each decoding step t, we obtain the probability of estimating y_t by marginalizing over all template elements. We also marginalize over all latent experts for the same input.

$$p(Y|X, A, d) \\ = \sum_{z=1}^{n_z} \prod_{t=1}^{n_Y} \sum_{\pi_t^d=1}^{n_\pi} [p(y_t|y_1, \ldots, y_{t-1}, X, A, \pi_t^d) \\ p(\pi_t^d|X, A, z)]p(z|X, A, d), \quad (2)$$

where $p(z|X, A, d) = 1/n_z$ is the uniform prior probability of the experts, because it has been observed (Shen et al., 2019) that the uniform prior encourages the model to make use of all the components for each input context. The control of complexity is achieved by choosing the set of possible template elements through d and an expert z is chosen to select a probable soft template based on a given input.

3.1 Model Details

As shown in Figure 1, our model consists of a passage encoder, an answer encoder and a question decoder. Each encoder employs a Bidirectional LSTM (BiLSTM) (Hochreiter and Schmidhuber, 1997) with different parameterizations, respectively. The question decoder is modeled by using a single layer LSTM with soft attention (Bahdanau et al., 2015) and a softmax layer.

The LSTM decoder utilizes soft templates and mixture of experts to generate complexity-controllable questions. As input, it takes the previous generated word y_{t-1}, the current context vector $\mathbf{c}_t^x$, the aggregated representation of the soft template elements $\mathbf{c}^{\pi_t^d}$, the embedding of an expert $\mathbf{e}_z \in \mathbb{R}^{d_z}$, and the previous hidden state s_{t-1}.

$$\mathbf{s}_t = LSTM(fc([\mathbf{y}_{t-1}, \mathbf{c}_t^x, \mathbf{c}_t^\pi, \mathbf{e}_z]), \mathbf{s}_{t-1}), \quad (3)$$

where fc denotes a full connected layer. The current context vector $\mathbf{c}_t^x$ is created by attending over the hidden representations of the passage encoder, following (Bahdanau et al., 2015). We initialize the first hidden state s_0 as $fc([h_{n_x}, \mathbf{e}_a, d, \mathbf{e}_z])$, where $\mathbf{e}_a$ denotes the embedding of the input answer a.

The soft template representation $\mathbf{c}_t^\pi$ at time t is aggregated over all template elements at the complexity level d, which is calculated as

$$\mathbf{c}_t^\pi = \sum_{i=1}^{n_{\pi^d}} p(\pi_i^d|\mathbf{c}_t^x, X, A, d, z)\mathbf{e}_{\pi_i^d}, \quad (4)$$

where $\mathbf{e}_{\pi_i^d}$ denotes the trainable embedding of an element π_i^d. The module $p(\pi_i|\mathbf{c}_t^x, X, A, d, z)$ estimates the relevance of a template element at time t. We consider soft attention over hard attention because it allows more than one elements to be relevant to the current context and the input. We take $p(\pi|X, A, d, z)$ as a learned prior distribution and model it with a gating network $G(\cdot)$. Moreover, to encourage sparse selection of elements, we model $G(\cdot)$ with choosing only the top-k most relevant ones by applying the noisy TopK gating network (Shazeer et al., 2017). This network also helps load balancing by introducing a noise term. More details can be found in (Shazeer et al., 2017). As a result, we obtain $\mathbf{c}_t^\pi$ by:

$$\mathbf{c}_t^\pi = \sum_{i=1}^{K} \text{softmax}(\text{TopK}([\mathbf{c}_t^x, \mathbf{e}_a, d, \mathbf{e}_z]))\mathbf{e}_{\pi_i^d}.$$

The parameters of each expert embedding $\mathbf{e}_z$ are initialized randomly and fine-tuned during training. During decoding, we iterate through all experts to generate n_z question candidates. Among them, the question with the highest $p(Y|X, A, d, z)$ is chosen as the final prediction.

Each state $\mathbf{s}_t$ in Eq.(3) is fed to the pointer-generator network (See et al., 2017) for generation of each word. This module is chosen to overcome out-of-vocabulary (OOV) words by coping them from input passages on demand.

3.2 Training

During training, we initialize the template elements by using questions at the respective complexity level and train the whole model with hard EM.

Template Element Initialization We initialize the embeddings of template elements by using the centroids of the question clusters at the respective complexity level. Compared to random initialization, it encourages embeddings to capture the intrinsic properties of distinct question templates. More specifically, we encode each question in the train set by using BERT (Devlin et al., 2019). Then we cluster the outcomes at each complexity level by using the improved k-means algorithm (Shi et al., 2010). The resulted cluster centroids are taken as the initial embeddings.

Training with Hard-EM We train the model with hard-EM (Lee et al., 2016) by taking the following two steps iteratively until convergence, because hard-EM can learn more diverse experts than soft-EM in NLG tasks (Shen et al., 2019).

E-step (hard). We calculate the loss for each expert and choose the expert with the minimal loss as the best one z^*.

$$z^* = \arg\min_z - \log p(Y|X, A, d, z).$$

M-step. We optimize the model parameters θ with the best expert z^*.

$$\min_\theta - \log p(Y|X, A, d, z^*; \theta).$$

4 Cross-Domain Complexity Estimator

It is desirable to build a cross-domain estimator to predict the complexity levels of questions because few domains have questions annotated with complexity levels for training CCQG models. As measuring complexity should be independent of domain-specific content, we use a simple classification rule without any training, which relies on the following five domain-independent features d_{f_i}.

Number of clauses in a question (d_{f_1}): The number of events/facts is a strong indicator of question complexity. We observe that the number of clauses are often proportional to the number of events/facts mentioned in a question. We use NLTK[1] to seek the question's syntactic tree to count the number of clauses.

Number of certain dependency relations in a question (d_{f_2}): Certain dependency relations across words influences the understanding of the content of a question (Kunichika et al., 2002). The more of them, the more difficult it is to understand. Thus, we count the number of *advmod, amod, nounmod, npmod*, and possessive modifiers after running the Spacy dependency parser [2] on questions.

Topic coherence of sentences in a passage (d_{f_3}): Kunichika et al (2002) observed that a passage is easy to understand if the topic coherence of its sentences is high. In light of this, we measure the topic coherence between sentences by calculating the Jensen–Shannon Divergence $\mathcal{JS}$ (Menéndez et al., 1997) between their topic distributions. $d_{\mathcal{JS}} = \frac{1}{n(n-1)} \sum_{i \neq j} \mathcal{JS}(\mathbf{t}_i, \mathbf{t}_j)$, where n is the number of sentences in a passage, and $\mathbf{t}_i$ and $\mathbf{t}_j$ denote the topic distributions of the i-th and j-th sentences in a passage respectively. As we expect the feature value is high if a question is complex, we let this feature $d_{f_3} = 1/d_{\mathcal{JS}}$.

Frequency of question entities in a passage (d_{f_4}): We observe that a question asking about an entity frequently appearing in a passage is often easier to answer than the one about an infrequent entity. Thus, we recognize entities in questions and passages, compute the average frequency of entities mentioned both in a question and a passage by $\text{avg}(Q) = \frac{1}{|E^Q|} \sum_{E^Q} \frac{n_{e_i}}{\sum_{E^P} n_{e_j}}$, where E^Q denotes the entity set in the question, E^P denotes the entity set in the passage and n_{e_i} is the number of mentions of e_i in the passage. Then the feature is the inverse of the averaged frequency $d(f_4) = 1/\text{avg}(Q)$.

Distance between entities in a question and an answer span in a passage (d_{f_5}): The answer to a question is often easy to find, if an entity mentioned in the question is located close to the answer in the same passage. Therefore, d_{f_5} is such a distance by taking the average number of tokens between the entities in a question and an answer span in a passage.

Classification rule The scoring function based on the above features is the average of all feature values after normalization $cpx(Q) = \frac{1}{5} \sum_{i=1}^{5} \text{Norm}(d_{f_i}(Q))$, where $\text{Norm}(d_{f_i}(Q)) = \frac{d_{f_i}(Q) - \min(d_{f_i}(Q))}{\max(d_{f_i}(Q)) - \min(d_{f_i}(Q))}$. We consider a question Q as *complex*, if $cpx(Q)$ is above a threshold λ, otherwise the question is classified as *simple*. The threshold can be easily tuned on a small sample of data annotated with complexity levels.

5 Experiments

In this section, we evaluate the effectiveness of the CCQG model and the complexity estimator.

5.1 Datasets and Complexity Annotation

We conduct experiments on two benchmark datasets SQuAD (Rajpurkar et al., 2016) and HotpotQA (Yang et al., 2018). We remove the questions that are unanswerable or whose answers are not contiguous fragments in the passage. For each dataset, we randomly select 80% of samples for training, 10% for validation, and 10% for testing.

We use only predicted complexity levels for training CCQG models on both datasets. In particular, we apply the cross-domain estimator to label each question with complexity levels. We calibrate the threshold λ on the questions labeled by *easy* and *hard* in the train set of HotpotQA, because only the questions in HotpotQA contain manually annotated complexity levels. The resulted $\lambda = 0.682$ is used in both HotpotQA and SQuAD. Table 1 summarizes the data statistics.

[1] http://www.nltk.org/

[2] https://spacy.io/

Table 1: The statistics of HotpotQA and SQuAD.

	HotpotQA			SQuAD		
	train	dev	test	train	dev	test
simple	45,585	5,698	5,426	41,604	5,201	5,235
complex	26,772	3,346	3,617	27,852	3,386	3,446

Table 2: Results of automatic evaluations on SQuAD and HotpotQA for varying complexity levels, the best performance is in bold.

Datasets	SQuAD-simple			SQuAD-complex		
Metrics	B-4	R-L	F1	B-4	R-L	F1
NQG++	12.19	45.39	69.26	11.16	43.70	65.39
DLPH	12.65	46.01	70.15	10.91	45.43	67.01
DeepQG	15.50	54.05	70.22	14.25	52.13	71.53
MoE	12.55	46.52	71.85	12.38	45.58	69.11
CCQG	**17.14**	**54.28**	**78.60**	**16.01**	**53.19**	**74.81**
w/o z	16.02	52.13	75.25	14.95	51.08	72.40
w/o π	13.05	46.21	72.81	12.57	44.19	69.37
Datasets	HotpotQA-simple			HotpotQA-complex		
Metrics	B-4	R-L	F1	B-4	R-L	F1
NQG++	12.35	44.51	63.37	10.76	41.26	64.05
DLPH	12.01	43.28	68.98	11.50	43.58	66.71
DeepQG	14.25	50.18	67.25	13.66	49.17	68.86
MoE	12.95	44.31	72.83	11.68	43.20	68.19
CCQG	**17.85**	**55.36**	**80.57**	**15.41**	**53.73**	**76.19**
w/o z	16.73	53.07	77.12	14.26	51.85	74.70
w/o π	13.87	46.98	73.91	13.01	46.50	70.05

Table 3: Results of human evaluations on SQuAD and HotpotQA with different complexity levels, the best performance is in bold.

Datasets	SQuAD-simple		SQuAD-complex		All	HotpotQA-simple		HotpotQA-complex		All
Metrics	Nat.	Cpx.	Nat.	Cpx.	Div.	Nat.	Cpx.	Nat.	Cpx.	Div.
NQG++	2.6	2.6	2.5	2.4	2.1	2.6	2.9	2.3	2.4	1.9
DLPH	2.7	2.5	2.7	2.7	2.7	2.6	2.7	2.5	2.7	2.4
DeepQG	3.1	2.2	3.0	2.9	2.3	2.9	2.5	3.0	2.9	2.1
MoE	2.9	1.9	2.9	2.9	2.9	3.1	2.3	2.8	2.7	2.8
CCQG	**3.6**	**1.5**	**3.5**	**3.3**	**3.6**	**3.7**	**1.3**	**3.6**	**3.4**	**3.6**
w/o z	3.5	1.7	3.3	3.1	3.0	3.6	1.6	3.3	3.2	3.4
w/o π	3.0	1.9	2.8	2.9	2.9	3.0	2.2	2.8	2.6	2.9

5.2 Settings for CCQG

Baselines. We compare our models with the following baselines on the two datasets.
NQG++: an encoder-decoder model with attention and copy mechanisms for QG tasks. It introduces lexical features and the answer position to enhance semantic representation (Zhou et al., 2017).
DLPH: an end-to-end difficulty-controllable QG model, which estimates the complexity level of a question based on whether the QA systems can answer it correctly or not (Gao et al., 2019).
DeepQG: an attention-based gated graph neural network that fuses the semantic representations of document-level and graph-level to select content and generate complex questions (Pan et al., 2020).
MoE: a method for diverse generation that uses a mixture of experts to identify diverse contents for generating multiple target text (Cho et al., 2019).
w/o z: our model without using mixture of experts.
w/o π: our model without using soft templates.

Implementation Details We set the number of experts n_z to 3 and the number of soft templates n_π to 12, for more values of n_z and n_π. The embedding dimensions for the complexity level d, the expert d_z and soft template π, are set to 30, 50 and 50 respectively. We set hidden vector sizes to 256. Models are optimized with the Adam (Kingma and Ba, 2015) and we initially set the learning rate to 0.001. Other standard parameters follow the default settings of the Pytorch[1]. We stop the training iterations until the performance difference between two consecutive iterations is smaller than 1e-6. For QG models that cannot be complexity-controllable, we concatenate the complexity vector with the hidden state from the encoder to initialize the decoder.

Metrics Automatic and human evaluation metrics are used to analyze the model's performance. **Automatic Metrics:** Following prior works (Zhou et al., 2017; Pan et al., 2020), we use the metrics BLEU-4 (B-4) and ROUGE-L (R-L) (Çelikyilmaz et al., 2020) to evaluate the quality of generated questions against references. The generated questions might have different complexity levels than the input ones. Thus we also report F1-score (F1) based on the discrepancy between the complexity levels of generated questions labeled by our complexity estimator and the input ones. **Human Metrics:** We randomly select 200 pairs of passage and answer from the test datasets in HotpotQA and SQuAD respectively (400 cases in all), and manually evaluate the questions generated by all methods. Three annotators are asked to judge each question independently according to the following four criteria on the Likert scale of 1–5, with 1 being the worst and 5 being the best. **Naturalness (Nat.)** rates the fluency and comprehensibility of the generated question. **Complexity (Cpx.)** is used to measure the complexity of correctly answering a generated question in a given passage. The higher the complexity, the more difficult it is to find the answer. Given the same passage and answer, we expect questions generated by two different com-

[1] https://pytorch.org

plexity levels to be distinct. Therefore, We employ **Diversity (Div.)** to measure the differences between the two questions with different complexity levels based on the same passage and answer.

5.3 Results and Analysis for CCQG

Automatic Evaluation. Table 2 indicates the results of automatic evaluation, we can observe that:

1. For overall performance, our model achieves the best performance across all metrics. Specifically, our model improves the BLEU-4 and ROUGE-L by at least 1.16 and 1.31, respectively, over the best baseline DeepQG, which is specifically designed for generating complex questions.

2. Our model achieves also superior consistency between input and output complexity levels in terms of F1 than all baselines, which use a single vector for each complexity label, attesting to our model's effectiveness in complexity modeling with mixture of experts and soft templates.

3. It is no surprise that generation of complex questions is more challenging. Our model and all baselines perform slightly better in terms of BLEU-4 and ROUGE-L on simple question generation than complex question generation. In contrast, complexity control is not always more difficult for some baselines on generating complex questions.

Human Evaluation. We conduct human evaluation to inspect if our findings of automatic evaluation are consistent with human perception. Apart from using the above mentioned metrics, we also provide sample questions generated by different models with varying complexity levels in Table 4.

1. **Naturalness** measures semantic and linguistic quality of generated questions. From Table 3 we can see that our model is superior in this metric in comparison to the SOTA models. Due to the task complexity, all models perform still better on simple questions than complex ones. As we can see from the sample questions in Table 4, the length of complex questions is relatively longer than that of the simple ones. Our close inspection also shows that our model generated more questions with complex syntactic structures than the baselines.

2. On simple question subsets, our model obtains the lowest **Complexity**, and conversely, on complex question subsets, we obtain the highest, which shows that our model is more capable of generating questions at the target complexity level.

3. **Diversity** is measured between two questions of different complexity levels, given the same passage and answer. The results show that CCQG yields the highest diversity, which leads to the conclusion that MoE and soft templates make the generated questions with varying complexities more distinct from each other. In contrast, a single vector for each complexity level makes the baselines difficult to generate substantially diverse questions.

Ablation Analysis. To further investigate the effectiveness of the MoE and soft templates, we perform the experiments by removing them respectively.

Effect of Expert z. From Tables 2 and 3, we can observe that the model (**w/o z**) performance drops obviously on complexity controlling (**F1** avd **Cpx.**) and diversity (**Div.**). We believe the main reason is that different experts z captures different modes of the underlying distributions, thus effectively play the vital role for selecting template elements at the target complexity level.

Effect of Soft Templates π. Without the soft templates, our model (w/o π) degenerates into the baseline MoE. It is evident from Tables 2 and 3 that the result of the model **w/o** π is very close to that of MoE. Concretely, all the metric values drop significantly, especially those related to the quality of the question, such as **B-4** and **R-L**. This shows that soft templates π play an important role in the full model. We believe that, on the one hand, π guarantees the quality of the generated questions by providing additional constraints (cluster centroids for similar questions). On the other hand, since the constraint information is different with different inputs (different cluster centroids are selected), it guides the model to generate more diverse questions.

5.4 Evaluation of Complexity Estimator

We evaluate the efficiency of the proposed complexity estimator on HotpotQA (in-domain) and SQuAD (out-domain). The threshold is tuned on the training set of HotpotQA. We compare our model with two baselines. The first one is **QA-sys** (Gao et al., 2019), which evaluates the complexity level of a question based on whether QA models can answer it or not. The second one is the BERT-based classifier utilizing unsupervised domain adaptation (**UDA**) (Nishida et al., 2020). **In-Domain Evaluation:** Only HotpotQA has ground truth complexity levels. In-domain evaluation is conducted on the questions labeled as $easy$ or $hard$ in the corresponding train, validation, and test datasets. **Out-Domain Evaluation:** The out-domain evaluation is conducted on SQuAD, whose

Table 4: Examples generated by our model and baselines, given the same passage and answer from HotpotQA.

Passage	The 2013 Liqui Moly Bathurst 12 Hour was an endurance race for a variety of GT and touring car classes, including: GT3 cars, GT4 cars, Group 3E Series Production Cars and Dubai 24 Hour cars. The event, which was staged at the Mount Panorama Circuit, near Bathurst, in New South Wales, Australia on 10 February 2013, was the eleventh running of the Bathurst 12 Hour. Mount Panorama Circuit is a motor racing track located in Bathurst, New South Wales, Australia. The 6.213 km long track is technically a street circuit, and is a public road, with normal speed restrictions, when no racing events are being run, and there are many residences which can only be accessed from the circuit.
NQG++	How long is the track? (simple) How long is the long track? (complex)
DLPH	What is the length of the track? (simple) What is the length of Mount Panorama Circuit? (complex)
DeepQG	What is the length of Mount Panorama Circuit, located in Bathurst, New South Wales? (simple) What is the length of the track, located in Bathurst, New South Wales, Australia? (complex)
MoE	How long is the track? (simple) What is the length of the track? (complex)
CCQG	**How long is the track? (simple) What is the length of the track at which the 2013 Liqui Moly Bathurst 12 Hour was staged? (complex)**
w/o z	What is the length of the track? (simple) What is the length of the track which is located in Bathurst, New South Wales? (complex)
w/o π	How long is the track? (simple) What is the length of the track? (complex)
Gold	**What is the length of the track where the 2013 Liqui Moly Bathurst 12 Hour was staged?** (complex)

questions are not labelled with complexity levels. We randomly sample 200 questions and employ three annotators to give feedback individually on the complexity level of each question on a scale of 1–3, with 1 being *simple*, 2 being *uncertain* and 3 being *complex*. Only when the results of two or more annotators are consistent, the label is regarded as the final complexity level of a question. We exclude the questions annotated with *uncertain* and use the remaining 187 questions for testing. Furthermore, to verify the reliability of annotators, we conduct a *Fleiss' kappa* test for each annotator's result. To this end, the kappa coefficients are 0.796, 0.794 and 0.776, respectively.

Table 5: In-domain and Out-domain evaluations of complexity estimator on SQuAD and HotpotQA, **p.s.**, **p.c.**, **t.s.** and **t.c.** refer to predicted as simple/complex, and true simple/complex, respectively.

Dataset	HotpotQA (In-domain)						SQuAD (Out-domain)					
Method	QA-sys		UDA		Ours		QA-sys		UDA		Ours	
	p.s.	p.c.	p.s.	p.c.	p.s.	p.c.	p.s.	p.c.	p.s.	p.c.	p.s.	p.c.
t.s.	4,369	1,057	4,628	798	5,271	155	87	22	76	34	93	15
t.c	1,219	2,398	961	2,656	210	3,407	24	54	30	47	12	67
F1	0.736		0.795		**0.958**		0.753		0.658		**0.856**	

Results: Table 5 reports F1-score and the confusion matrix for each method on the two datasets. (1) In the in-domain setting, our cross-domain estimator outperforms QA-sys and UDA in terms of F1 scores with a wide margin. QA-sys falls short of UDA by 5%, which shows that it is not reliable to use the correctness of answering questions as a way of assessing complexity levels. (2) In the out-domain setting, QA-sys surprisingly achieves comparable performance in both settings, but is still more than 10% behind our model. We conjecture that the relatively poor performance of both learning-based deep models may attribute to the domain specific spurious features that are irrelevant to complexity levels of questions.

6 Discussion on MoE-based Architecture

We provide justifications of the MoE-based architecture from the perspective of high-level cognition. Humans can easily ask questions that are simple and complex questions (Rothe et al., 2017), mainly because we can identify patterns through a certain mechanism and then combine these patterns for generalizing to various scenarios. That is, humans possess the capability for compositional generalization, which is critical for learning in real-world situations (Atzmon et al., 2020). Some studies have shown the importance of modularity for this capability (Sternberg, 2011; Clune et al., 2012). They suggest that modularity is conducive to the specialization of different modules, which are responsible for different functions. In other words, specialization improves generalization. Similarly, a modular neural network enables compositional generalization like human intelligence. MoE-based architecture can be regarded as an implementation of this concept. MoE is a tightly coupled modular structure designed so that similar inputs are mapped to similar expert modules, effectively making each module specialize in a different selection.

7 Conclusion and Future Work

We propose a novel encoder-decoder model incorporating soft templates and MoE to address the problem of complexity-controllable question generation. As most domains do not have training data for CCQG models, we propose a simple and effective cross-domain estimator to predict the missing complexity levels of questions. In the extensive experiments of both CCQG and complexity assessment tasks, our models achieve superior performance over the competitive baselines across all experimental settings. In the future, we will consider anaphora resolution and numerical reasoning in complexity estimator, and explore the performance of our model in different applications, such as examination and assisting QA systems.

Acknowledgments

This work was supported by the National Key Research and Development Program of China under grants [2017YFB1002801, 2018YFC0830200]; the Natural Science Foundation of China grants [U1736204]; the Fundamental Research Funds for the Central Universities [2242021k10011].

References

Tahani Alsubait, Bijan Parsia, and Uli Sattler. 2014. Generating multiple choice questions from ontologies: Lessons learnt. In *ISWC*, pages 73–84.

Yuval Atzmon, Felix Kreuk, Uri Shalit, and Gal Chechik. 2020. A causal view of compositional zero-shot recognition. In *NeurIPS*.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *ICLR*.

Sheng Bi, Xiya Cheng, Yuan-Fang Li, Yongzhen Wang, and Guilin Qi. 2020. Knowledge-enriched, type-constrained and grammar-guided question generation over knowledge bases. In *COLING*, pages 2776–2786.

Samuel R. Bowman, Luke Vilnis, Oriol Vinyals, Andrew M. Dai, Rafal Józefowicz, and Samy Bengio. 2016. Generating sentences from a continuous space. In *SIGNLL*, pages 10–21.

Nicola De Cao, Wilker Aziz, and Ivan Titov. 2019. Question answering by reasoning across documents with graph convolutional networks. In *NAACL*, pages 2306–2317.

Ziqiang Cao, Wenjie Li, Sujian Li, and Furu Wei. 2018. Retrieve, rerank and rewrite: Soft template based neural summarization. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 152–161, Melbourne, Australia. Association for Computational Linguistics.

Asli Çelikyilmaz, Elizabeth Clark, and Jianfeng Gao. 2020. Evaluation of text generation: A survey. *CoRR*, abs/2006.14799.

Jaemin Cho, Min Joon Seo, and Hannaneh Hajishirzi. 2019. Mixture content selection for diverse sequence generation. In *EMNLP*, pages 3119–3129.

Jeff Clune, Jean-Baptiste Mouret, and Hod Lipson. 2012. The evolutionary origins of modularity. *CoRR*, abs/1207.2743.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Yifan Gao, Lidong Bing, Wang Chen, Michael R Lyu, and Irwin King. 2019. Difficulty controllable generation of reading comprehension questions. In *IJCAI*, pages 4968–4974.

Michael Heilman. 2011. *Automatic factual question generation from text*. Ph.D. thesis, Ph. D. thesis, Carnegie Mellon University.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Comput*, 9(8):1735–1780.

Diederik P Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. *ICLR*.

Vishwajeet Kumar, Yuncheng Hua, Ganesh Ramakrishnan, Guilin Qi, Lianli Gao, and Yuan-Fang Li. 2019. Difficulty-controllable multi-hop question generation from knowledge graphs. In *ISWC*, pages 382–398.

Hidenobu Kunichika, Minoru Urushima, Tsukasa Hirashima, and Akira Takeuchi. 2002. A computational method of complexity of questions on contents of english sentences and its evaluation. In *ICCE*, pages 97–101.

Stefan Lee, Senthil Purushwalkam, Michael Cogswell, Viresh Ranjan, David J. Crandall, and Dhruv Batra. 2016. Stochastic multiple choice learning for training diverse deep ensembles. In *NeurIPS*, pages 2119–2127.

M.L. Menéndez, J.A. Pardo, L. Pardo, and M.C. Pardo. 1997. The jensen-shannon divergence. *J Franklin Inst*, 334(2):307 – 318.

Kosuke Nishida, Kyosuke Nishida, Itsumi Saito, Hisako Asano, and Junji Tomita. 2020. Unsupervised domain adaptation of language models for reading comprehension. In *LREC*, pages 5392–5399.

Liangming Pan, Yuxi Xie, Yansong Feng, Tat-Seng Chua, and Min-Yen Kan. 2020. Semantic graphs for generating deep questions. In *ACL*, pages 1463–1475.

Ethan Perez, Patrick S. H. Lewis, Wen-tau Yih, Kyunghyun Cho, and Douwe Kiela. 2020. Unsupervised question decomposition for question answering. In *EMNLP*, pages 8864–8880.

Pranav Rajpurkar, Robin Jia, and Percy Liang. 2018. Know what you don't know: Unanswerable questions for squad. In *ACL*, pages 784–789.

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.

Anselm Rothe, Brenden M. Lake, and Todd M. Gureckis. 2017. Question asking as program generation. In *NeurIPS*, pages 1046–1055.

Abigail See, Peter J. Liu, and Christopher D. Manning. 2017. Get to the point: Summarization with pointer-generator networks. In *ACL*, pages 1073–1083.

Dominic Seyler, Mohamed Yahya, and Klaus Berberich. 2017. Knowledge questions from knowledge graphs. In *SIGIR*, pages 11–18.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc V. Le, Geoffrey E. Hinton, and Jeff Dean. 2017. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *ICLR*.

Tianxiao Shen, Myle Ott, Michael Auli, and Marc'Aurelio Ranzato. 2019. Mixture models for diverse machine translation: Tricks of the trade. In *ICML*, volume 97, pages 5719–5728.

Na Shi, Xumin Liu, and Yong Guan. 2010. Research on k-means clustering algorithm: An improved k-means clustering algorithm. In *IITSI*, pages 63–67.

Saul Sternberg. 2011. Modular processes in mind and brain. *Cognitive neuropsychology*, 28(3-4):156–208.

Md. Arafat Sultan, Shubham Chandel, Ramón Fernandez Astudillo, and Vittorio Castelli. 2020. On the importance of diversity in question generation for QA. In *ACL*, pages 5651–5656.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *EMNLP*, pages 2369–2380.

Qingyu Zhou, Nan Yang, Furu Wei, Chuanqi Tan, Hangbo Bao, and Ming Zhou. 2017. Neural question generation from text: A preliminary study. In *NLPCC*, volume 10619, pages 662–671.

A Analysis on Complexity Factors

We have proposed a novel complexity computational method, which consists of five factors. In this section, we present questions of different complexity levels of the same passage, as shown in Figure 2, to illustrate the detailed calculation of each factor. Furthermore, we analyze the influence of each factor in detail. We randomly select 1000 samples from each complexity-level questions in HotpotQA whose original label is $easy$, $medium$ or $hard$. As is shown in Figure 3, 4, 5, 6 and 7, we can observe that the values of these proposed factors have apparent relationship with the complexity of the questions, which demonstrates the efficiency of our complexity estimator. Specifically,

$\boldsymbol{d_{f_1}}$**) Number of clauses in a question**: We leverage the off-the-shelf toolkit NLTK[1] to seek the question's syntactic tree to count the number of clauses. The larger d_{f_1} is, the more complex the question is. In Figure 2, both Q_1 and Q_2 have no clauses and Q_3 has one clause, hence $d_{f_1}(Q_1) = d_{f_1}(Q_2) = 0$, and $d_{f_1}(Q_3) = 1$. From Figure 3, we can see that the number of clauses in $simple$ questions is less than 2. Although some questions with $medium$ and $hard$ level contain fewer clauses or even no clauses, most of them have more clauses than simple questions on the whole. This is also in line with our intuition, a question with complex construction is often not simple.

$\boldsymbol{d_{f_2}}$**) Number of certain dependency relations in a question**: We use spaCy[2] to make dependency parsing for a question and count the number of modifiers, which are labeled "mod" in the dependency parsing tree. In Figure 2, Q_1 and and Q_2 have no modifiers and Q_3 has four modifiers, including "American", "black", "comedy", and "thriller", hence $d_{f_2}(Q_1) = d_{f_2}(Q_2) = 0$, and $d_{f_2}(Q_3) = 4$. Figure 4 shows the number of modifiers in questions with different complexity levels. We can see that there is an obvious difference in this factor between $easy$ and $hard$ questions. The effect of the number of modifiers on a question's complexity is essentially similar to the number of clauses.

$\boldsymbol{d_{f_3}}$**) Topic coherence of sentences in a passage**: We train a topic model by Gensim[3] to compute the topic distribution of each sentence in the passage. Then we leverage the Jensen–Shannon Divergence $\mathcal{JS}$ (Menéndez et al., 1997) to measure the similarities between these topic distributions. Figure 5 demonstrates the relevance between sentences in a passage. We use the divergence of topic distribution between sentences to quantify this complexity factor. It is a seemingly unrelated factor, because even if the topic of a passage is scattered, it is random for the questioner to ask complex or simple questions. However, our statistical results show that this factor has a significant correlation with complexity. In addition to being inspired by previous work, we also have a look inside HotpotQA. We find that most $hard$ level questions are multi-hop, and the more sentences involved in answering these questions, the more likely the topic distribution between them is scattered, and the lower the relevance.

$\boldsymbol{d_{f_4}}$**) Frequency of question entities in the passage**: As can be seen from Figure 2, $Passage_2$ has six entities, including "Irma Pamela Hall", "A Family Thing", "Soul Food", "The Ladykillers","Joel" and "Ethan Coen". Q_2 has one entity "The Ladykillers", which appears twice in the passage. And Q_3 has three entities "Irma Pamela Hall", "Joel" and "Ethan Coen", and each appears once in the passage. Therefore, $n_{\text{Irma Pamela Hall}} = 1$, $n_{\text{A Family Thing}} = 1$, $n_{\text{Soul Food}} = 1$, $n_{\text{The Ladykillers}} = 2$, $n_{\text{Joel}} = 1$, $n_{\text{Ethan Coen}} = 1$. As a result, $d_{f_4}(Q_2) = 7/2$, $d_{f_4}(Q_3) = 7$, and Q_3 is more complex than Q_2. Figure 6 shows that the more frequently the entity in question appears in the passage, the more complex the question becomes.

$\boldsymbol{d_{f_5}}$**) Distance between entities in a question and an answer span in a passage**: For $Passage_2$ in Figure 2, for Q_2, there are 10 tokens between "The Ladykillers" and "Joel and Ethan Coen"; and for Q_3, there are 10 tokens between "The Ladykillers" and "Joel", 12 tokens between "The Ladykillers" and "Ethan Coen", and 37 tokens between "The Ladykillers" and "Irma Pamela Hall", so $d_{f_5}(Q_2) = 10$, and $d_{f_5}(Q_3) = 59/3$. Hence, Q_3 is more complex than Q_2. Figure 7 shows, intuitively, that as the complexity of the question increases, the distance between the answer and the entities in question increases.

On the whole, a single factor cannot completely distinguish questions with different complexities, but statistics show that $easy$ and $hard$ can be clearly distinguished.

[1] http://www.nltk.org/

[2] https://spacy.io/

[3] https://radimrehurek.com/gensim/

$Passage_1$: Chavano Rainier Buddy Hield is a Bahamian professional basketball player for the Sacramento Kings of the NBA…

Q_1: Which team does Buddy Hield play for? ***Ans*:** Sacramento Kings.

$Passage_2$: Irma Pamela Hall (born June 3, 1935) is an American actress who has appeared in numerous films and television shows since the 1970s. She is best known for playing matriarchal figures in the films "A Family Thing", "Soul Food", and "The Ladykillers". … The Ladykillers is a 2004 American black comedy thriller film directed by Joel and Ethan Coen.

Q_2: Who directed "The Ladykillers" ? ***Ans*:** Joel and Ethan Coe.

Q_3: Which American black comedy thriller film directed by Joel and Ethan Coen includes Irma Pamela Hall, an American actress who has appeared in numerous films and television shows since the 1970s? ***Ans*:** The Ladykillers.

Figure 2: Question-answer pairs for two sample passages. We use different colors to mark different entities in passages and questions.

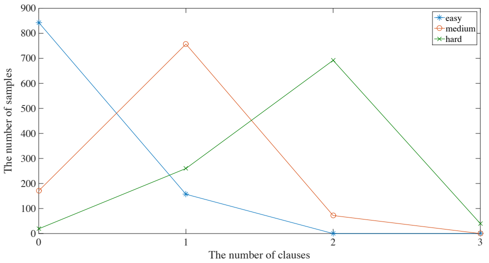


Figure 3: The distribution of the number of clauses among different complexity-level questions.

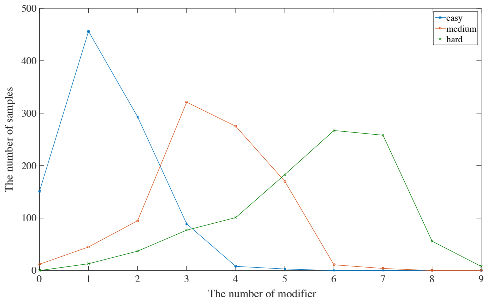


Figure 4: The distribution of the number of modifier among different complexity-level questions.

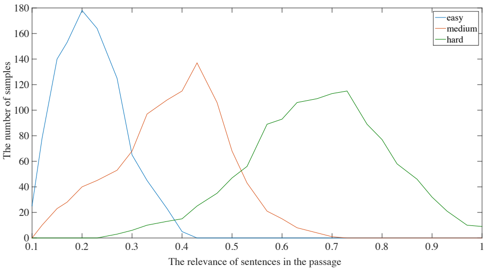


Figure 5: The distribution of the relevance of sentences in the passage among different complexity-level questions.

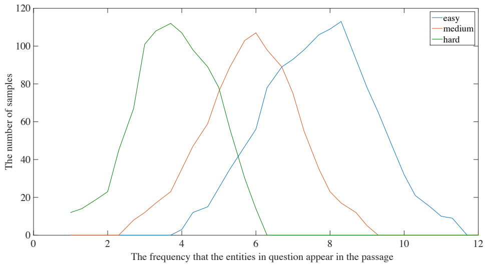


Figure 6: The distribution of the frequency that the entities in question appear in the passage among different complexity-level questions.

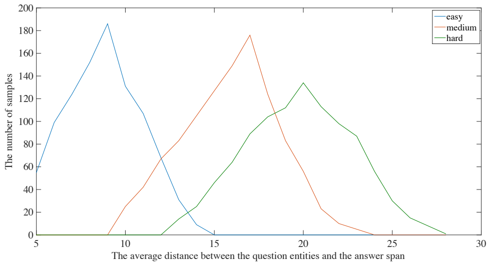


Figure 7: The distribution of the average distance between the question entities and the answer span in the passage among different complexity-level questions.

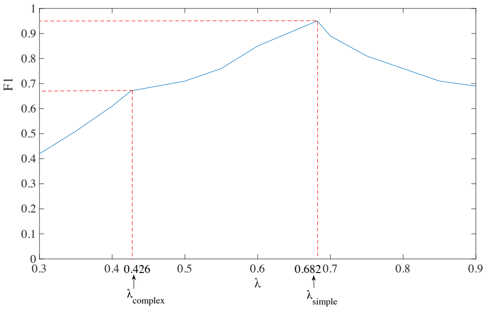


Figure 8: The performance of the estimator with different thresholds on HotpotQA.

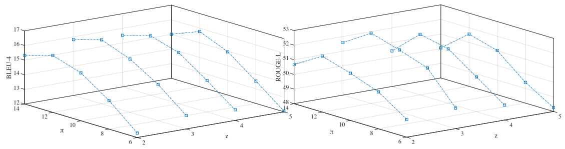
B Parameter Selection

Selection of the Complexity Threshold λ

For HotpotQA, it has three original complexity levels, $easy$, $medium$ and $hard$. For simplicity, we reclassify HotpotQA into two complexity levels for our complexity estimator, $simple$ and $complex$. Concretely, we compute complexity score for each question. Among them, λ_{simple} refers to the maximum complexity of all $easy$-level questions, and $\lambda_{complex}$ refers to the minimum complexity of all $hard$-level questions.

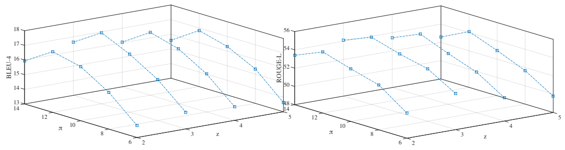
We select different threshold values to label the dataset and evaluate our estimator. As is shown in Figure 8, when the threshold λ=0.682, our estimator has the highest F1 value. Hence, we leverage λ_{simple} as our standard complexity threshold.

In this work, we apply the complexity threshold learned from HotpotQA to SQuAD, that is, if a question's complexity score is higher than λ, the question is labeled as $complex$, otherwise, labeled as $simple$.



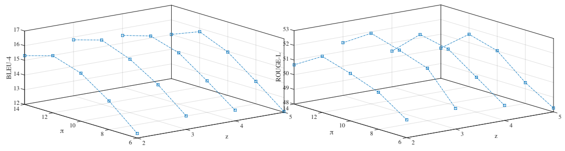
(a) BLEU-4. (b) ROUGE-L.

Figure 9: The performance of our model with different number of z and π on SQuAD.



(a) BLEU-4. (b) ROUGE-L.

Figure 10: The performance of our model with different number of z and π on HotpotQA.



(a) BLEU-4. (b) ROUGE-L.

Figure 11: The performance of our model with different number of z and π on HotpotQA.

Selection of z and π

For better performance, we investigate how the number of expert z and soft template π influence our model performance with GridSearch in scikit-learn [1]. We set the number of z to 2, 3, 4, 5 and the number of π to 6, 8, 10, 12, 14 and make experiments on SQuAD and HotpotQA datasets. We use BLEU-4 and ROUGE-L to evaluate the performance. The results are shown in figure 9 and figure 11, and we find our model has the best performance when z=3 and f=12. Although the distributions of the two datasets are different, the trends in z and π are similar, that is too high or too low z and π will make the results worse. Hence, we set the number of experts n_z to 3 and the number of soft templates n_π to 12 in our experiments.

C Fleiss' kappa

We conduct a *Fleiss' kappa* test for three annotators. Specifically, we sample another 100 examples and ask 3 human annotators to give complexity level of each question on a scale of 1–3, with 1 being *simple*, 2 being *uncertain* and 3 being *complex*. We remove the questions whose complexity is labels as *uncertain* and utilize the remaining 187 questions for *Fleiss' kappa* text. Finally, we calculate the kappa coefficients are 0.796, 0.794 and 0.776, respectively, using a Python module *statsmodels*[2]. The confusion matrix is shown in Table 6.

Table 6: Confusion matrix of different annotation results.

	Annotator 1		Annotator 2		Annotator 3	
	simple	complex	simple	complex	simple	complex
simple	49	4	50	3	47	6
complex	5	33	6	32	4	34

[1] https://scikit-learn.org

[2] https://www.statsmodels.org/