Unsupervised Generation of Long-form Technical Questions from Textbook Metadata using Structured Templates

Indrajit Bhattacharya, Subhasish Ghosh, Arpita Kundu,
Pratik Saini and **Tapas Nayak**
TCS Research, India
{b.indrajit, g.subhasish, arpita.kundu1, pratik.saini, nayak.tapas}@tcs.com

Abstract

We explore the task of generating long-form technical questions from textbooks. Semi-structured metadata of a textbook — the table of contents and the index — provide rich cues for technical question generation. Existing literature for long-form question generation focuses mostly on reading comprehension assessment, and does not use semi-structured metadata for question generation. We design unsupervised template based algorithms for generating questions based on structural and contextual patterns in the index and ToC. We evaluate our approach on textbooks on diverse subjects and show that our approach generates high quality questions of diverse types. We show that, in comparison, zero-shot question generation using pre-trained LLMs on the same meta-data has much poorer quality.

1 Introduction

We address automated generation of long-form technical questions from textbooks. Such questions can then be used for technical assessments, such as in interviews and examinations. Existing work on long-form question generation mostly focuses on questions for reading comprehension assessment (Dhole and Manning, 2020; Bang et al., 2019; Xiao et al., 2020; Zhao et al., 2018; Back et al., 2021; Cui et al., 2021; Huang et al., 2021).

We observe that textbook metadata — specifically, the index and the table of contents (ToC) — provide rich cues for question generation. Fig.1 and Fig.2 show fragments from the index and the ToC of a textbook on Python. The index structure is hierarchical and often parsed into an subject and a context (e.g. *functions and classes, managing*). The ToC is similarly hierarchical. The main challenge in question generation from unrestricted natural language content is identifying the relevant entity (or entities) and their context. The mentions of some of the relevant entities is often far away from the context, and the context may also use complex linguistic constructs. In contrast, the grammar for the metadata is significantly restricted, thereby simplifying the detection of the entities and contexts. Additionally, their hierarchical structure compactly and completely captures relevant context. In this paper, we focus on automatically generating technical long-form questions using the index and ToC of textbooks. While structured or semi-structured data has been used in NLP tasks such as knowledge graph construction (Suchanek et al., 2007), table-to-text generation (Wang et al., 2020b) and factoid question generation has been explored from knowledge-graphs (Wang et al., 2020a; Han et al., 2022), to the best of our knowledge there is no existing work that uses semi-structured content to generate questions either for technical assessment or for reading comprehension.

We use structural patterns in the index and ToC to design question templates of varying types and complexity, and then use unsupervised regular expression-based algorithms to generate questions by instantiating these templates using index and ToC entries. Template based approaches have been used in question generation from free text, typically with higher precision than deep learning based counterparts, while generating fewer questions (Fabbri et al., 2020; Puzikov and Gurevych, 2018; Yu and Jiang, 2021). However, we are not aware of use of templates over semi-structured content for question generation.

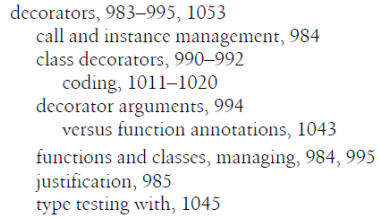

Figure 1: Fragment of a Python text-book Index

We apply our approach to generate questions

Proceedings of the 29th International Conference on Computational Linguistic, pages 21–28
October 12–17, 2022.

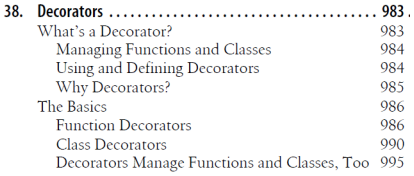

Figure 2: Fragment of a Python text-book ToC

from multiple textbooks on diverse subjects such as Machine Learning, Java and PL/SQL to demonstrate its generality. Using automated and manual evaluation, we show that the generated questions are complex and diverse while having very high quality. We compare our approach with zero-shot question generation using LLMs, GPT-3 (Brown et al., 2020) and BART (Lewis et al., 2020). We show that these perform poorly in terms of both validity and diversity of the generated questions.

2 Structural Templates for Q.Generation

In this section, we describe our approach for question generation from text-book index and ToC using structural templates. We focus on generating questions that are *answerable from book context*. These have reference answers in the book, which can be used reliably for assessment. Consider the index fragment from Fig.1. WHAT ARE THE BENEFITS OF DECORATORS? and WHAT IS THE RELATION BETWEEN DECORATOR ARGUMENTS AND CLASS DECORATORS? are meaningful questions, but the index provides no evidence that the book contains the answers. In contrast, the index suggests that WHAT ARE CLASS DECORATORS? and WHY DO WE NEED DECORATORS? are answerable from the book. We want to generate the two latter questions but not the first two.

Index Indexes are typically structured as a forest of trees. Our example fragment shows one such tree. It typically mentions one or more root entities (e.g. *decorators*), sometimes with an additional comma separated context (e.g. *libraries, third-party*). Each child entry is about a specific context of the root entity. These may be one or more related entities (*call and instance management*) or attributes and instances (*class decorators*), sometimes with additional connecting context, which maybe a prefix (e.g. *versus function annotations*) or a comma separated suffix (e.g. *functions and classes, managing*). The context may also be about specific tasks involving the root entity (e.g. *coding, type testing with*). This structure may repeat at the third level. We use NLTK to detect the entities as simple or compound noun phrases, and contexts as involving prepositions (IN) and gerunds or present participles (VBG), making use of separator commas when present.

The templates used for generating questions from index entries are summarized in Tab.4 in the Appendix. First, we discuss question templates based on a single index entry containing an entity phrase e, a context c, or both. (a) WHAT IS/ARE e?, if e is present; (b) WHAT IS/ARE c OF e?, if e and c are both present, and c matches the *examples* regex (e.g. example(s)|instance(s) (of)*); (c) WHAT IS/ARE c OF e?, same as above with c matching the *uses* regex (use(s)|usage|application(s) (of)*; (d) WHAT IS/ARE c OF e?, same as above with c matching the *property* regex (part(s)|component(s)|step(s)...(of|for)*).

More interesting templates are those that consider a parent index entry containing entity e_p, and a child entry containing entity e_c and connecting context c. Some are based on simple patterns: (e) HOW DO YOU COMPARE e_p AND e_c?, if c matches the *comparison* regex (vs|versus|compared to|...); (f) WHAT IS THE RELATION BETWEEN e_p AND e_c?, if c is 'and'. We also generate the *uses*, *examples* and *properties* questions for e_p when e_c is absent and c matches the corresponding regex.

Other connecting contexts c specify action sometimes involving one or two child entities e_{c1} and e_{c2} (e.g. *using decorators in functions*). The specific action patterns for c are VBG, VBG IN, VBG e_{c1}, VBG IN e_{c1} and VBG e_{c1} IN e_{c2}. The corresponding question templates are WHAT CAN YOU SAY ABOUT VBG followed by e_p, IN e_p, e_{c1} FOR e_p, IN e_{c1} FOR e_p, and e_{c1} IN e_{c2} FOR e_p, respectively. Though HOW DO YOU VB is a more natural prefix, lemmatization frequently fails to recover the correct base verb from the VBG form.

A frequent pattern for the child context c is ending with preposition (e.g. *debugging with*, *coding of*, *karma configuration for*). The corresponding question template is WHAT CAN YOU SAY ABOUT c e_p? when the token preceding the preposition is VBG, and WHAT IS/ARE c e_p? otherwise.

Any parent entry e that is unused in the main question template, is used to construct a question prefix 'REGARDING e, ' to completely specify the context, (e.g. REGARDING DECORATORS, WHAT ARE CLASS DECORATORS?).

Table of Contents The restricted structure of the index makes detection of entities and contexts simple. The same restriction, however, prevents the templates from drawing upon context to add natural variety to generated questions. In contrast, Fig.2 illustrates that ToC entries are often complex phrases and even complete sentences and questions. However, the grammar is still considerably restricted compared to that for natural language used inside the book. This forms a nice trade-off between ease of detection and naturalness of the expression. The ToC is however much smaller than the index.

ToC entries are of different types: *question* (Q) (e.g. What's a Decorator?), *question phrase* (QP) (e.g. Why Decorators?), *sentence* (S) (containing non-gerund verb) (e.g. Decorators Manage Functions and Classes, Too), *VBG phrase* (VP) (e.g. Using and Defining Decorators), *simple noun phrase* (SNP) (e.g. Class Decorators) and *complex noun phrase* (CNP) (with coordinating conjunctions) (e.g. Things to Remember about Decorators). We detect these types using simple regular expressions involving POS tags and small dictionaries. The templates for the above categories are Q, CAN YOU EXPLAIN QP?, DO YOU THINK S?, WHAT CAN YOU SAY ABOUT VP?, WHAT IS/ARE SNP? and WHAT CAN YOU SAY ABOUT CNP?, respectively. As for the index, we recursively construct the question prefix using remaining parents. The parent e can again be of one of the above types. We construct the prefix based on the parent's type: 'REGARDING e, ' for CNP and QP, 'FOR e, ' for VBG, 'SINCE e, ' for S, and 'e ' for Q, respectively. The templates used for generating questions from ToC entries are summarized in Tab.3 in the Appendix.

3 Experimental Evaluation

In this section, we present experimental evaluation of our question generation approach.

Data: We use text books on diverse subjects — Python (Lutz, 2013), PLSQL (Feuerstein and Pribyl, 2014), Java (Schildt, 2007), Machine Learning (Murphy, 2012) and Deep Learning (Goodfellow et al., 2016). The first two have the richest metadata structure, with 3-level indexes and ToCs. In contrast, Java has a 2-level index, while ML and DL have 1-level indexes and DL has 2-level ToCs. We process the textbook PDFs to extract their index and ToC automatically. Details are in the appendix.

Deep Models: We compare with two state-of-the-art LLMs. For GPT-3 (Brown et al., 2020), we use its Interview Questions preset. We take BART (Lewis et al., 2020) pre-trained for (factoid) question generation on SQuAD (Rajpurkar et al., 2016), and post-train it for long-form question generation on the MASH-QA dataset (Zhu et al., 2020). This has been previously used for long-form answer extraction. We use contexts and their corresponding questions for post-training. Details of hyperparameter settings are in the appendix. For both models, we split the index and ToC forests into complete individual trees, and provide one tree as one context. For GPT-3, we construct a prompt by concatenating a context with a new line and "Generate interview questions from this book index" (alternatively "book table of contents").

Evaluation: For each approach, we evaluate different aspects of their quality. A question is **(a) context-relevant** if it includes (non-trivial) terms from the context. It is **(b) context-closed** if its non-trivial terms *only* from the context, and only from a *single hierarchy path*. These checks invalidate the two example unanswerable questions at the start of Sec.2. A question is **(c) context-complete**, if it includes non-trivial terms from each ancestor entry in the hierarchy. In Fig.1, a question that just mentions *coding*, without referring to *decorators* and *class decorators* is incomplete. The **(d) level-span** of a question is the number of hierarchy levels that contribute terms to the question. In Fig.1, a question only about *decorators* has level span 1. It increases to 2 by additionally including *class decorators*, and to 3 on further including *coding*. We obtain a question form / template by masking out copied terms from the context. In addition to varying level spans, **(e) the number of unique question forms** is an indicator of diversity of questions. These measures are computed automatically (details in appendix). In addition, we manually evaluate the **(f) validity** of a question by checking its semantic correctness, completeness, and answerability from book context.

Results: The results for Index questions are shown in Tab.1a. Note that the first evaluation is context-relevance. *All other evaluations are performed only on context-relevant questions*. The main pattern is similar across subjects. GPT-3 generates the largest number of questions, but ranks the lowest in context-relevance. It also scores poorly in context-closedness.Structured templates generate fewer questions but with very high quality. BART

Subject	#Entry	Model	#Q.	%Cx-Rel	%Cx-Cmp	%Cx-Cl	%L-Span 1	%L-Span 2	%L-Span 3	#Q.Form	%valid
Python	1822	BART	728	96	93	63	94	6	0	10	76
		GPT-3	**6251**	23	72	24	84	15	1	**19**	48
		STemp	1216	**100**	**99**	**100**	50	44	5	15	**80**
Java	2541	BART	1753	81	96	67	97	3	NA	9	86
		GPT-3	**15522**	13	84	28	86	14	NA	**26**	48
		STemp	2294	**100**	**100**	**100**	73	27	NA	11	**96**
DL	585	BART	589	98	**100**	80	100	NA	NA	3	78
		GPT-3	**3943**	44	**100**	18	100	NA	NA	**5**	34
		STemp	556	**100**	**100**	**100**	100	NA	NA	2	**98**

(a) Results for Index Question Generation

Subject	#Entry	Model	#Q.	%Cx-Rel	%Cx-Cmp	%Cx-Cl	%L-Span 1	%L-Span 2	%L-Span 3	#Q.Form	%valid
Python	906	BART	42	90	24	45	66	31	3	3	42
		GPT-3	313	92	10	53	76	23	1	16	60
		STemp	**324**	**100**	**100**	**100**	7	33	60	**27**	**72**
Java	866	BART	33	**100**	39	70	82	18	0	5	57
		GPT-3	486	99	12	77	82	16	2	17	62
		STemp	**503**	**100**	**100**	**100**	4	45	50	**21**	**72**
DL	184	BART	21	95	60	55	50	50	NA	6	48
		GPT-3	**175**	85	19	74	92	8	NA	10	68
		STemp	122	**100**	**100**	**100**	11	88	NA	**11**	**84**

(b) Results for ToC Question Generation

Table 1: Question generation results for BART, GPT-3 and Structural template (STemp) for different subjects. #Entry: no. of index/ToC entries, #Q: no. of questions, %Cx-Rel: pct. of context-related, %Cx-Cmp: pct. of context-complete, %Cx-Cl: pct. of context-closed, %L-span: 1, 2, 3 pct. of level-span 1, 2, 3, #Q-form: no. of unique question forms, and %valid: pct. of manually verified valid questions. All columns after %Cx-Rel are evaluated only for context-relevant questions. %L-span is NA when data has fewer than that number of levels.

generates the fewest number of questions but with higher quality than GPT-3. However, BART and GPT-3 questions are largely restricted to single entries and do not span 2 or 3 levels. GPT-3 however has more variety in question forms.

The results for ToC questions are in Tab.1b. First, we note that here GPT-3 and structured templates (STemp) generate similar number of questions, while BART generates very few. All three approaches mostly generate context-relevant questions. Context-closedness increases for BART and GPT-3 compared to index questions, but is still significantly lower than templates. Context-completeness on the other hand drops for these two approaches compared to index. In terms of level span, BART and GPT-3 questions are again mostly restricted to single entries while templates make use of multiple levels. BART has very few question forms, while those for GPT-3 and templates is similar and higher.

Additional results for PL/SQL and Machine Learning are included in the appendix.

The measures discussed so far are automated. We also performed human validation for all subjects on 50 randomly sampled questions from each of the three approaches, *after filtering out context-irrelevant questions*. For index questions, structured templates have very high validity. BART performance is acceptable, but majority of GPT-3 questions are invalid. The situation is different for ToC questions. While template questions have the highest validity, it is much lower than for index. The performance of GPT-3 increases significantly while that of BART falls close to or below 50%.

Error Analysis: In validity evaluation, the reduced performance of STemp for ToC is largely due to errors in type classification of entries. Unsurprisingly, POS pattern based classification misfires more often for ToC entries than index entries. For example, the ToC entry *Bayesian inference when* σ^2 *is unknown?* is misclassified as Sentence(S), instead of *Complex Noun Phrase(CNP)*. As a result, the question is generated using the DO YOU THINK template for S instead of the WHAT CAN YOU SAY ABOUT template for CNP.

GPT-3 and BART errors are largely due to introduction of additional irrelevant terms. This happens more for GPT-3 for shorter index entries, as it fails to understand the context. As examples, from the context *wake-sleep algorithm*, GPT-3 generates the question WHAT ARE COMMON SLEEP DISORDERS?, and from the context *elif (else if) clause*,

BART generates the question WHAT IS THE ELIF (ELSE IF) CLAUSE IN A CONTRACT?. Surprisingly, BART makes such errors more frequently for longer ToC entries.

Also, unlike GPT-3, BART typically generates a single question per context, which is an index or a ToC tree, even though it is post-trained with multiple questions per context.

4 Conclusions

We have motivated the task of automated technical question generation from semi-structured text-book index and ToC. We have proposed an unsupervised approach based on structured templates that make use of their restricted grammar and hierarchical structure. We have shows using extensive evaluation that this approach performs better according to many different aspects of quality on multiple text-books on a variety of subjects compared to zero-shot approaches using large language models.

References

Seohyun Back, Akhil Kedia, Sai Chetan Chinthakindi, Haejun Lee, and Jaegul Choo. 2021. Learning to generate questions by learning to recover answer-containing sentences. In *Findings of the ACL-IJCNLP*.

Liu Bang, Zhao Mingjun, Niu Di, Lai Kunfeng, He Yancheng, Wei Haojie, and Xu Yu. 2019. Learning to generate questions by learning what not to generate. In *WWW*.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, T. J. Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeff Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *NeurIPS*.

Shaobo Cui, Xintong Bao, Xinxing Zu, Yangyang Guo, Zhongzhou Zhao, Ji Zhang, and Haiqing Chen. 2021. Onestop qamaker: Extract question-answer pairs from text in a one-stop approach. In *ArXiv*.

Kaustubh D. Dhole and Christopher D. Manning. 2020. Syn-qg: Syntactic and shallow semantic rules for question generation. In *ACL*.

Alexander Fabbri, Patrick Ng, Zhiguo Wang, Ramesh Nallapati, and Bing Xiang. 2020. Template-based question generation from retrieved sentences for improved unsupervised question answering. In *ACL*.

Steven Feuerstein and Bill Pribyl. 2014. *Oracle PL/SQL Programming*. O'Reilly, CA, USA.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

Kelvin Han, Thiago Castro Ferreira, and Claire Gardent. 2022. Generating questions from wikidata triples. In *LREC*.

Xinting Huang, Jianzhong Qi, Yu Sun, , and Rui Zhang. 2021. Latent reasoning for low-resource question generation. In *ACL*.

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 com- prehension. In *ACL*.

Mark Lutz. 2013. *Learning Python*. O'Reilly, CA, USA.

Kevin Patrick Murphy. 2012. *Machine Learning: a Probabilistic Perspective*. MIT Press, Massachusetts, USA.

Yevgeniy Puzikov and Iryna Gurevych. 2018. E2E NLG challenge: Neural models vs. templates. In *INLG*.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100,000+ questions for machine comprehension of text. In *EMNLP*.

Herbert Schildt. 2007. *Java: The Complete Reference*. McGraw-Hill, NY, USA.

Fabian M. Suchanek, de, and Gerhard Weikum. 2007. Yago: A core of semantic knowledge unifying wordnet and wikipedia. In *WWW*.

Siyuan Wang, Zhongyu Wei, Zhihao Fan, Zengfeng Huang, Weijian Sun, Qi Zhang, and Xuanjing Huang. 2020a. PathQG: Neural question generation from facts. In *EMNLP*.

Zhenyi Wang, Xiaoyang Wang, Bang An, Dong Yu, and Changyou Chen. 2020b. Towards faithful neural table-to-text generation with content-matching constraints. In *ACL*.

Dongling Xiao, Han Zhang, Yukun Li, Yu Sun, Hao Tian, Hua Wu, and Haifeng Wang. 2020. Ernie-gen: An enhanced multi-flow pre-training and fine-tuning framework for natural language generation. In *IJCAI*.

Xiaojing Yu and Anxiao Jiang. 2021. Expanding, retrieving and infilling: Diversifying cross-domain question generation with flexible templates. In *EACL*.

Yao Zhao, Xiaochuan Ni, Yuanyuan Ding, and Qifa Ke. 2018. Paragraph-level neural question generation with maxout pointer and gated self-attention networks. In *EMNLP*.

Ming Zhu, Aman Ahuja, Da-Cheng Juan, Wei Wei, and Chandan K Reddy. 2020. Question answering with long multiple-span answers. In *Findings of the Association for Computational Linguistics: EMNLP*.

A Appendix

A.1 Textbook PDF Processing

We process the textbook PDFs to extract their index and ToC automatically. The task is to extract the entries along with their levels in the hierarchy from the ToC and index of the book PDFs. ToC entries often have different text font and size to represent chapter, sub-chapter headers and different indentation to represent different levels as shown in Fig.2. Index entries mostly have same text font and size but have different indentation to represent the level of hierarchy as shown in Fig.1. We first use **pdfminer library** to extract texts with the metadata such as text sizes, fonts, and text coordinates from the PDFs. Then we write a wrapper on top of pdfminer that uses this metadata to filter out unnecessary parts (e.g. header, footers) from the text and annotate different aspects(e.g. entry, level, page no) of the index and ToC. Books from different publishers have different text size, fonts and coordinates for different elements. So the filtering parameters need to customized to some extent for each book.

A.2 Additional Results

The results for PL/SQL and Machine Learning are shown in Tab.2. The observations for the three models are very similar to those for the 3 subjects in Sec.3.

A.3 Automated Evaluation

Here, we describe our automated evaluation algorithms.

Context-relevance: We first identify non-trivial terms by tokenizing a question using NLTK tokenizer and removing stopwords, wh-words and prepositions using NLTK POS tagger. Then we check for presence of these terms and entries in ToC or index by using stemming and a synonym dictionary. If at least one question term occurs in an entry, then we mark it as context-relevant.

Context-closedness: A question is context-closed if there exists a *hierarchy path* from root to leaf in *a single tree* of the index or ToC that contains *all non-trivial terms* in the question. We first create a set of composite terms from all entries, where a composite term is the longest contiguous sequence of non-trivial terms in an entry. We identify composite terms in a question and map these to some entry (or none) using Rouge-F1 score. We finally check if all composite terms in the question have been mapped, and also mapped to entries in a single hierarchy path.

Context-completeness: After mapping composite terms in a question to a hierarchy path of entries, we check if for each matched entry in the matched entry list, if its parent is also presents in the list. If so, we mark the as context-complete.

Level-span: The level span of a question is the number of distinct entries mapped to it in a hierarchy path.

Question form: To extract the form of a question, we mask all mapped terms in it. The resultant masked string is the question form. Then we count the number of distinct forms in a set of questions. Note that, here we report number of question form on context-closed questions.

A.4 Hyperparameter Settings

We experiment with BART-base (Lewis et al., 2020) model pre-trained on SQuAD (Rajpurkar et al., 2016) and post-train on MASH-QA data (Zhu et al., 2020). All experiments are done on 10GB A100 GPU with 8 CPU cores and 30GB RAM. We use batch size of 8 and train the model for 10 epochs. We optimize the model parameters using Adam optimizer with a learning rate of 0.0001.

For GPT-3, we use the Interview Question preset. We set the temperature parameter to 0 to eliminate randomness and keep the other parameters as default.

A.5 Templates Summary

Templates used for generating questions from hierarchical index and ToC are summarized in Tab. 4 and Tab. 3 respectively.

Subject	#Entry	Model	#Q.	%Cx-Rel	%Cx-Cmp	%Cx-Cl	%L-Span 1	2	3	#Q.Form	%valid
PL/SQL	748	BART	1329	98	92	70	92	8	0	22	74
		GPT-3	**10659**	36	68	32	81	18	1	**44**	46
		STemp	2330	**100**	**99**	**100**	44	51	6	16	**92**
ML	2418	BART	2418	97	**100**	78	**100**	NA	NA	6	80
		GPT-3	**17855**	32	**100**	15	**100**	NA	NA	**17**	20
		STemp	2254	**100**	**100**	**100**	**100**	NA	NA	2	**94**

(a) Additional Results for Index Question Generation

Subject	#Entry	Model	#Q.	%Cx-Rel	%Cx-Cmp	%Cx-Cl	%L-Span 1	2	3	#Q.Form	%valid
PL/SQL	748	BART	31	**100**	35	68	74	26	0	3	39
		GPT-3	329	98	3	74	81	18	1	11	52
		STemp	**357**	**100**	**100**	**100**	4	28	67	**13**	**90**
ML	777	BART	32	**100**	25	72	72	28	0	4	69
		GPT-3	**366**	**100**	9	80	76	23	1	15	68
		STemp	344	**100**	**100**	**100**	5	28	66	**20**	**74**

(b) Additional Results for ToC Question Generation

Table 2: Question generation results for BART, GPT-3 and Structural template for different subjects. #Entry: no. of index/ToC entries, #Q: no. of questions, %Cx-Rel: pct. of context-related, %Cx-Cmp: pct. of context-complete, %Cx-Cl: pct. of context-closed, %L-span: 1, 2, 3 pct. of level-span 1, 2, 3, #Q-form: no. of unique question forms, and %valid: pct. of manually verified valid questions. All columns after %Cx-Rel are evaluated only for context-relevant questions. %L-span is NA when data has fewer than that number of levels.

Entry Types	**Example Entry**	**Template**	**Example**
Question (Q)	*What's a Decorator?*	Q	What's a Decorator?
Question Phrase (QP)	*Why Decorators?*	CAN YOU EXPLAIN QP?	Can you explain why decorators?
Sentence (S)	*Decorators Manage Functions and Classes, Too*	DO YOU THINK S?	Do you think decorators manage functions and classes?
VBG Phrase (VP)	*Managing Functions and Classes*	WHAT CAN YOU SAY ABOUT VP?	What can you say about managing functions and classes?
Simple Noun Phrase (SNP)	*Class Decorators*	WHAT IS/ARE SNP?	What are class decorators?
Complex Noun Phrase (CNP)	*Things to Remember about Decorators*	WHAT CAN YOU SAY ABOUT CNP?	What can you say about things to remember about decorators?

Table 3: Types of ToC entries and templates used for generating questions from these with examples.

	Entry Pattern	Condition	Example Entry	Template	Example
Single	e		*decorators*	WHAT IS/ARE e?	What are decorators?
	e, c or $c\ e$	c matches 'use', 'example', 'property' regex	*decorators*, *use*	WHAT IS/ARE c OF e?	What are uses of decorators?
Parent Child	Pa: e_p Ch: e_c, c or $c\ e_c$	c matches 'comparison' regex	Pa: *decorators arguments* Ch: *versus function annotations*	HOW DO YOU COMPARE e_p and e_c?	How do you compare decorator arguments and function annotations?
		c matches 'and'	Pa: *decorators arguments* Ch: *and function annotations*	WHAT IS THE RELATION BETWEEN e_p AND e_c?	What is the relation between decorators and class decorators?
	Pa: e_p Ch: c	c matches 'use', 'example', 'property' regex	Pa: *decorators* Ch: *examples*	WHAT IS/ARE c OF e?	What are examples of decorators?
	Pa: e_p Ch: VBG $e_{c_1}\ c_2\ e_{c_2}$	e_{c_1}, c_2, e_{c_2} = Null	Pa: *decorators* Ch: *Coding*	WHAT CAN YOU SAY ABOUT VBG e_p?	What can you say about coding decorators?
		POS(c_2) = IN e_{c_1}, e_{c_2} = Null	Pa: *decorators* Ch: *type testing with*	WHAT CAN YOU SAY ABOUT VBG IN e_p?	What can you say about type testing with decorators?
		c_2, e_{c_2} = Null	Pa: loops Ch: coding techniques	WHAT CAN YOU SAY ABOUT VBG e_{c_1} FOR e_p?	What can you say about coding techniques for loops?
		e_{c_1} = Null	Pa: *decorators* Ch: *using functions*	WHAT CAN YOU SAY ABOUT VBG e_p IN e_{c_2}?	What can you say about using decorators in functions?
		POS(c_2) = IN	Pa: *performing essential tasks* Ch: *hiding source code of stored programs*	WHAT CAN YOU SAY ABOUT VBG e_{c_1} IN e_{c_2} FOR e_p?	What can you say about hiding source code of stored programs for performing essential tasks?

Table 4: Patterns and templates used for generating questions from index entries with examples. Pa and Ch denote parent and child index entries respectively.