Machine Learning for Automated Generation of Multiple-Choice Test Items

Sowmya Ramachandran, Jeremy Ludwig, Bridge Eimon, Stephanie Ortiz-Sevilla

Stottler Henke Associates Inc
San Mateo. CA
sowmya@stottlerhenke.com, ludwig@stottlerhenke.com, beimon@stottlerhenke.com, ssevilla@stottlerhenke.com

ABSTRACT

Effective assessments are essential for training and personnel development. Developing quality assessments is labor-intensive. This could lead to a shortage of assessments that can significantly impact the Army's ability to develop their personnel and a Soldier's ability to plan their career growth. There is a unique opportunity to harness the powers of data science and artificial intelligence to address this problem. Recent breakthroughs in neural network technologies have led to stunning successes in natural language processing techniques (NLP). Neural network architectures like the Transformer, and the availability of vast amounts of digital natural language data have led to the creation of highly effective, pre-trained task-independent language models. Once trained, these models can be applied to any number of natural language processing tasks. All this raises an interesting question: are these approaches sufficiently rich to automate the task of generating assessments? The language models seem particularly useful for automating the generation of multiple-choice items to test recall and comprehension as a starting point of this investigation. This paper will describe an investigation into this question. Specifically, we use pre-trained language models like BERT (Bidirectional Encoder Representation from Transformers) that have been fine-tuned to tasks like question generation, question answering, and summarization. We combine this with traditional approaches like rule-based pattern extraction to generate a range of question types. Success of this approach will offer benefits such as tool that will enable assessment creators to draw from a high-quality, automatically generated question-bank. In this paper, we will examine the feasibility and effectiveness of this approach and to identify gaps and challenges. We will describe the models and algorithms that we have implemented and analyze their effectiveness and describe results of formative evaluations of the quality of generated content.

ABOUT THE AUTHORS

Dr. Sowmya Ramachandran is a research engineer at Stottler Henke Associates where her research focuses on the application of artificial intelligence (AI), machine learning, and natural language processing to improve human performance in a broad range of domains such as intelligence analysis and tactical decision-making. Dr. Ramachandran has led the development of Intelligent Tutoring Systems for a diverse set of domains, from training paramedics to training Information Systems Technicians in the U.S. Navy.

Dr. Jeremy Ludwig is a principal engineer at Stottler Henke Associates, Inc. He directs teams of computer scientists and conducts research in artificial intelligence, applying reasoning, knowledge representation, and machine learning to create solutions for complex, real-world, problems.

Bridge Eimon is an AI software engineer at Stottler Henke Associates, Inc. He leads the computer vision group of Stottler Henke, specializing in machine learning and neural networks for solving a wide variety of real world classification and recognition projects, as well as automating human decision making processes.

Stephanie Oritz-Sevilla is an AI Developer at Stottler Henke Associates. She received B.A. degrees in Computer Science and Data Science with a domain emphasis in robotics from the University of California, Berkeley. Ortiz Sevilla has worked on the development of a wide range of AI projects, primarily with a focus on machine learning.

Machine Learning for Automated Generation of Multiple-Choice Test Items

Sowmya Ramachandran, Jeremy Ludwig, Bridge Eimon, Stephanie Ortiz-Sevilla

Stottler Henke Associates Inc
San Mateo, CA
sowmya@stottlerhenke.com, ludwig@stottlerhenke.com, beimon@stottlerhenke.com, ssevilla@stottlerhenke.com

PROBLEM STATEMENT

Effective assessments are the cornerstone of training and personnel development initiatives. Creating effective assessments is labor-intensive. Multiple-choice assessments require labor for generating high-quality questions and answer choices. Essay-based assessments require someone to create good questions (along with rubrics for assessing responses) as well as score them. Automated or even semi-automation tools for this task will contribute significantly to reduced costs in time and labor. This in turn will make it easier to generate multiple assessments and to update assessments to keep up with curriculum changes. Can artificial intelligence be used to make such a tool?

Recent breakthroughs in neural network technologies have led to stunning successes in natural language processing techniques (NLP). Neural network architectures like the Transformer (Vaswani et al., 2017), and the availability of vast amounts of digital natural language data have led to the creation of highly effective, pre-trained task-independent language models. Once trained, these models can be applied to any number of natural language processing tasks. All this raises an interesting question: are these approaches sufficiently rich to automate the task of generating assessments? The language models seem particularly useful for automating the generation of multiple-choice items to test recall and comprehension. This paper describes an effort to investigate this question. We will describe initial results from research to investigate the effectiveness of using state-of-the-art neural network approaches to automatically generate assessments for the Army. The goal is to demonstrate feasibility and utility by developing a tool called QGen using this approach and validating its effectiveness at generating multiple-choice test items.

We use open-source, Transformers-based, pre-trained language models, combined with heuristic-based processes for data pre-processing and question filtering. We are conducting several cycles of formative evaluations to assess the quality of generated content. This paper will describe the models and algorithms used and results of our initial formative evaluation. We will first provide a high-level overview of the state of the art in using neural networks for NLP. We will then present the details of the methods and models used to generate test items. Following this, we will then discuss the formative evaluation. We will conclude the paper with a summary of what we have learned so far as well as the plans for future work.

BACKGROUND

This project applies state-of-the art natural language processing techniques based on multilayered neural networks. A multilayer feedforward neural network consists of several layers of neurons, with each neuron being a computational unit with several inputs and outputs. A neuron typically computes a weighted sum of its inputs and performs a non-linear transformation of this sum to result in an output, which it then passes on as an input to the neurons in the next layer. There are several ways to build and connect layers of neurons, but a basic architecture is a fully connected network with every neuron in one layer connected to every other neuron in the next layer (Figure 1). The input layer processes the input signals (e.g., a sequence of text) and the output layer produces a prediction, or an output predicated on the input. Note that the number of layers in a network and the number of nodes in each layer are dependent on the data being modeled and therein lies the craft of using neural networks, namely selecting an architecture that models the data with high accuracy. Each layer has a weight matrix:

$$W_l \in \mathbb{R}^{n \times m} \ \forall \ 1 \ \leq l \leq k \qquad (1)$$

A nonlinear transformation is applied at each node. E.g. ReLU

$$ReLU(x) = \begin{cases} 0, if x < 0 \\ x, if x \geq 0 \end{cases} \qquad (2)$$

During training, a loss function (e.g., mean-squared error) is minimized using gradient descent.

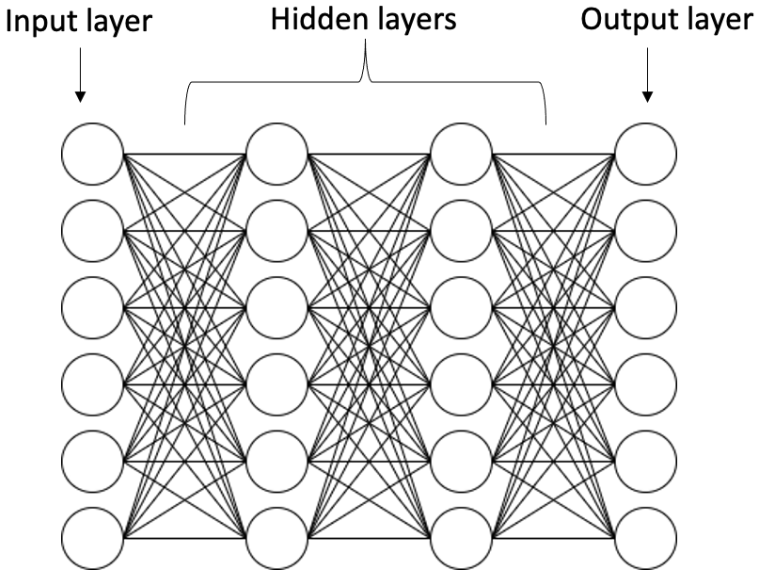

Figure 1. Basic architecture of a multilayer neural network

Neural Networks for NLP: State of the Art

NLP problems typically involve sequence to sequence transformation. For example, when translating from one language to another, a common NLP task, requires processing sequences of words. NLP tasks also have variable length inputs, which requires NLP models to have a strategy for dealing with variable length sequences. Thus, the recurrent neural network (RNN) and the Long-Short Term Memory (LSTM) networks were for a long time the architectures of choice for NLP tasks because of their ability to maintain state information. When processing an input from a sequence, they are able remember the previous items in the sequence and use that state information as context. The transformer architecture was introduced that furthered the state-of-the-art across many different NLP benchmarks and shifted the field away from RNNs and LSTMs towards non-state based models (Vaswani et al., 2017). Instead of using memory to process variable length inputs the transformer truncates inputs to meet its fixed sized input. The transformer architecture uses basic feed forward neural network layers, long distance connections throughout the network and attention mechanisms. The attention mechanisms are designed to codify contextual information to interpret the components of a sequence.

Most machine learning tasks are supervised (i.e. learning from labeled data) or unsupervised (i.e. learning from unlabeled data). Labeled data greatly helps generate accurate models but generating such data is costly. A next major advancement of the state-of-the-art came in 2018 when OpenAI introduced the idea of learning from unlabeled data with a self-supervised task before finetuning the model for the ultimate objective with labeled data (Radford et al., 2018). This approach combines the benefits of supervised and unsupervised learning. To learn from unlabeled data OpenAI designed the Masked Language Model (MLM) task which requires no labels, where they mask words from the input and the network must try to fill in the masked words. This task was incredibly successful at teaching the network a general language understanding which it could leverage to perform better on downstream classification or generation tasks.

Various augmentations and refinements to the two initial papers resulted in improved model performance. The Bidirectional Encoding Representations from Transformers (BERT) paper introduced new pretraining tasks and augmented the Transformer architecture with bidirectional conditioning upon the input (Devlin et al., 2019). The subsequent Robustly Optimized BERT Pretraining Approach (RoBERTa) further improved the state-of-the-art, refining the network structure and pretraining parameters (Liu et al., 2019). XLNet further improves upon BERT, replacing the MLM pretraining task with the Permutation Language Model (PLM) pretraining task, which shuffles

the words instead of randomly hiding words (Yang et al., 2020). The Text-To-Text Transfer Transformer (T5) paper was an effort to integrate these developments by empirically evaluating the many variations of Transformer architectures and pretraining tasks to find the best model. They evaluated the use of various pretraining datasets, pretraining tasks, architectures, and attention masks, pretraining and finetuning versus multitasking, and scaling methods on a series of downstream tasks such as question answering and summarization. The T5 model is empirically the best combinations of the above-mentioned categories according to their results (Raffel et al., 2020).

APPROACH

Figure 2 shows the high-level conceptualization of QGen components and the information flow between them. A basic assumption is that QGen is provided a source document such as a field manual or a doctrinal manual from which to generate test items. Multiple source documents on a topic may be used if necessary. Given a document, QGen creates a question bank of test items and made available to users for assessment creation. Users will be able to edit generates items and augment the answer choices and distractors as necessary. We expect that QGen items will make up a part of the assessment and users will need to add items of their own to make a complete assessment.

The source document is fed into QGen and the text is extracted from the document. The text then goes through preprocessing, which includes cleaning, and segmentation where the text is divided into contiguous clusters called contexts. The contexts are used to generate questions that are subsequently made available to users. The questions and contexts are also passed to the answer and distracter generators. The generated questions, answers, and distracters go to postprocessing, where a variety of rules and heuristics are used to identify and filter out bad test items. The set of test items so generated is saved in a database and made available as a question bank via the QGen web application. The application enables users to select questions, answers, and distracters for use in their test.

Data Ingestion

QGen is currently focused on source texts presented as PDFs. We found that ingesting PDFs is more challenging than expected. After a comparison of readily available, open-source off-the-shelf PDF readers, we selected PyMuPDF (McKie & Liu, 2021). The challenges include removing extraneous text (e.g., headers) while retaining everything of relevance, and segmenting text in coherent chunks (since the PDF reader loses paragraph-based organization of text). The data ingestion step also includes keyword-based filters to remove unnecessary artifacts. An example of such an artifact is a text block containing "<image: [DeviceRGB], width: [42], height: [20], bpc: [8]>,". QGen eliminates such artifacts from the text using regular expressions that look for the following keywords with patterns of strings and integers separating them: "image", "width", "height", and "bpc".

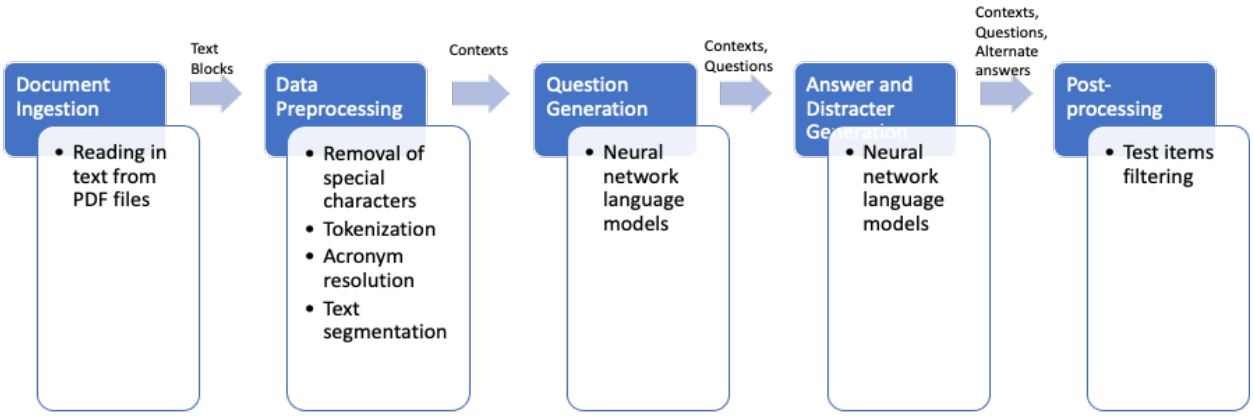

Figure 2. Process Flow for Content Generation

Text Preprocessing and Segmentation

Due to the particularities of the PDF reading package, QGen extracts blocks instead of extracting paragraphs. QGen splits each extracted block into sentences using the sentence tokenizer from the Python library NLTK (Bird et al.,

2009). QGen cleans the master list of sentences by removing any characters or symbols which cannot be represented in standard utf-8 encoding; extracting text offset with brackets, parentheses, or “greater than” / “less than” symbols; and consolidating excessive whitespace.

Segmentation is the next step in the pre-processing pipeline where the cleaned text is further prepared by grouping semantically related sentences together. Text segmentation is important to provide an adequate context for question generation that is richer than a single sentence. The contexts should be sufficiently enough to generate interesting, non-trivial questions but not so long as to substantially decrease the number of generated questions. We implemented three different techniques:

- Segment based on similarity as measured by a word embedding model
- Segment based on similarity as measured by the Term Frequency Inverse Document Frequency (TF-IDF) metric (Manning, Raghavan, and Schütze 2008)
- Rolling window approach

We have found a simple rolling-window based approach to be sufficient. There are also other methods found in literature for grouping adjacent blocks of text by computing coherence, but their utility is uncertain, and we have not investigated their use (Tran et al. 2018). Finally, we implemented coreference resolution as a part of preprocessing the data (Clark & Manning, 2016), but none of the readily available tools for this are sufficiently accurate for our purposes. We found that, in many instances, using coreference resolution (as implemented by these COTS libraries) fundamentally changes the meaning of the text and therefore leads to inaccurate content. For this reason, we have dropped coreference resolution from the pipeline.

Question Generation

The process used by QGen for multiple-choice item generation is to first generate questions for each of the contexts or segments generated by the pre-processing step. For this research, we are using open-source deep language models for this step. Leveraging such models saves significant amount of time and resources that would have otherwise been spent on training new networks. While fine-tuning the pretrained models on domain-specific data may yield better results, obtaining sufficient domain-specific training data would also be resource intensive and increase the barrier to adoption of the tool. Therefore, we wished to study the effectiveness of using off-the-shelf pretrained models for generation of multiple-choice test items. We also have a team of subject matter experts (SMEs) performing formative evaluations of the generated content (see the EVALUATION section). Our formative evaluations indicate that this approach is effective at generating a good number of question and correct answer pairs of acceptable quality. A more extensive validation of the utility of QGen will be conducted at the end of the current phase of research.

We used pretrained models from a large repository of open-source models provided by huggingface.co (www.huggingface.co). Of the several such models in the huggingface.co repository for generating questions and answers, we filtered out those that did not have adequate user documentation. We selected from amongst the remaining models informally. We tested different models against a common span of text from our sample source documents. We selected the model that produced the best results based on an informal examination of the results[1]. This model uses the T5 architecture (Raffel et al., 2020) and was trained on the SQuAd data set (Rajpurkar et al., 2018).

Using a model from the repository is straightforward and requires only a few lines of programming, as shown in Figure 3. Once the models have been loaded and the function, all that is required is to call the function *get_question* with the target context.

[1] Specifically, we used the mrm8488/t5-base-finetuned-question-generation-ap

```
In [2]:
import torch
from transformers import AutoModelWithLMHead, AutoTokenizer, AutoModelForQuestionAnswering

generator_tokenizer = AutoTokenizer.from_pretrained("mrm8488/t5-base-finetuned-question-generation-ap")
generator_model = AutoModelWithLMHead.from_pretrained("mrm8488/t5-base-finetuned-question-generation-ap")



def get_question(context, max_length=64):
    input_text = "context: %s </s>" % (context)
    features = generator_tokenizer([input_text], return_tensors='pt')

    output = generator_model.generate(input_ids=features['input_ids'],
                attention_mask=features['attention_mask'],
                max_length=max_length)

    return generator_tokenizer.decode(output[0])

```

Figure 3. Code for Using the Question Generation Language Model

Figure 4 shows examples of the questions generated from a chapter in the field manual titled "The Commander's Handbook on The Law of Land Warfare" (FM 6-27)[2]. This set of questions are among those rated acceptable by the SMEs. Figure 5 shows the questions that received a bad rating from the SMEs. Note that the last question shown in the figure can be easily made acceptable by deleting the word "other". This highlights an import an important point: we expect that some or many of the questions generated by this approach will require minor editing to render them useful. Given our experience, we assess that the state-of-the-art in this technology is good enough for fully autonomous operations.

What are the two main reasons for the preservation and enforcement of the laws of war?

What is the ICRC's role in detainee operations?

What is a military objective?

What is the purpose of distinction?

Figure 4. Examples of "Good" Questions Generated

What does LOAC try to protect civilians by requiring combatants to apply the principles of distinction and proportionality?

What is the purpose of paragraphs 2-39 through 2-57?

What other protections may apply to persons held by opposing forces?

Figure 5. Examples of "Bad" Questions Generated

Answer Generation

We are using pretrained language models for answer generation as well. Using the same criteria as for question generation, we selected a model from the Huggingface repository of pretrained models[3]. Given a context and a question, the answer generation model returns an answer to the question found in the context. If it does not find an

[2] For this research, we only used manuals that have been cleared for public distribution.

[3] Specifically, we used the tuner007/t5_abs_qa model for both answer and distracter generation

answer the string "No answer available" is returned. The answer generation model is also based on the T5 architecture (Raffel et al., 2020). As with question generation, it only takes a few lines of programming to download the pre-trained models and to generate an answer by passing in a question and a context.

By and large, our evaluations have found that the question-answering model generates good answers for several questions, but it also produces many unsuitable or incorrect answers. By "unsuitable," we mean that answers were nonsensical, incomplete, or unrelated to the questions. This effect was sometimes mitigated by pairing the question with other contexts to generate answers. Sometimes a different context than the one that generated the question yielded a better answer. This is discussed further below.

The goal of QGen is to generation multiple-choice questions which require, in addition to the question, one or more correct answers and a set of distracters. Distracter generation is a time-consuming activity for assessment authors. Automating this presents a challenge as the distracters must be similar enough to the actual answers both in syntax and semantics to be plausible answers. Otherwise, the distracters seem obviously incorrect, and this has a negative impact on assessment validity. We have found distractor generation to be one of the most challenging parts of generating multiple-choice test item. To generate a good distractor, it is necessary to go beyond the context used to generate the questions. Humans often draw from their own knowledge of adjoining and related concepts and a knowledge of the target audience to write good distractors.

To simulate the process of looking beyond the text that generated the question, QGen uses the approach of pairing the question with contexts other than the one associated with the question to try to find alternate answers that may serve as distractors. This is illustrated in Figure 6. The figure shows context-question pairs and shows how questions are paired with other contexts to generate a list of potential responses. We find that this method leads to the generation of answers that are syntactically and semantically appropriate. Furthermore, we found that this method leads to the identification of alternate correct answers to the question and sometimes also to better answers than that generated by the original context. However, as the EVALUATION section shows, it does not yield a high number of actual distractors as we had hoped. One reason is that distractor generation requires knowledge that is source document even – it requires deep subject matter expertise and a high degree of common-sense reasoning that is difficult to replicate with the QGen approach.

We have not implemented any method to automatically determine if an answer generated this way is a correct answer, a distracter (or just irrelevant). End users will have to make this discrimination. Figure 7 shows examples of generated distracters that received high ratings from the SMEs. We continue to investigate other solutions to the problem of distractor generation.

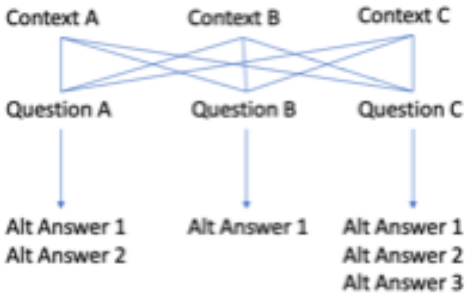

Figure 6. Pairing of Questions and Contexts for Generation of Alternate Answers

Q: What is the first step for a navigator in the field?
Generated correct answer: Orienting the map
Generated distracter: Determine the elevation of a point on the map
Generated distracter: The user must know the contour interval for the map he is using.

Q: What is the name of a groove in the land that is stretched out?
Generated correct answer: Valley
Generated distracter: Ridgeline
Generated distractor: Draw

Figure 7. Examples of Acceptable Distracters Generated

The generation steps described above are computationally intensive due to the use of deep neural network language models for several of the steps. Each pass through the neural networks involves millions of computations and generating test items for a document may involve several hundreds or thousands of such passes, depending on the size of the document. However, this process is entirely automated save for the initial step of specifying the input source document and no human labor is involved even if the question generation process takes a long time.

Test Item Filtering

It is expected that QGen will generate questions and answers of mixed quality. We have developed filters to remove inappropriate questions and poor answers. We find that language models, when presented with inputs that are outside their training distribution, will systematically present various behaviors leading to poor quality questions and answers. A simple case might be when a model cannot find an answer to a question in the provided text. This is a completely reasonable example, because the question-generation models may ask questions about subjects which are not described in the source context. In this case, it is common for answering models to return blank answers or some text of the nature, "No answer available". These types of responses are easily filtered out.

A less interpretable case is when, for some inputs, question answering models generates nonsensical text with repeated phrases or is a tautology (e.g. "A hill is a hill that is a hill that is a hill."). This text is clearly nonsensical—however, the model is confident it has answered the question. Filtering out these redundant nonsensical answers is a bit trickier. We have implemented two post-processing filters to help remove bad questions and/or answers from the results of the models. The first is a simple phrase-triggered elimination rule. Should specific phrases appear in the question, or answer text, we filter out that question or answer. For example, QGen currently filters out questions with the word "Figure" because most questions refer to figures that QGen cannot currently extract. The second is a Vocabulary Diversity measurement defined as follows:

$$VocabDiversity = \frac{number\ of\ unique\ words\ in\ an\ answer}{number\ of\ words\ in\ the\ answer} \quad (3)$$

We require every generated answer or distractor to be above some vocabulary diversity threshold. This filter is good at removing nonsensical redundant words; however, the vocabulary diversity filter is somewhat dependent upon the threshold used. During the Base Period we set this threshold empirically. Figure 8 shows examples of answers that were eliminated by this filter.

Points on contour lines are points on a contour line that are a line of
The slope diagram is a diagram of the slope of a slope
The contour line is a line of a line that is a line of a
The vertical distance between two points is the distance between the two points.
The contour line is a line of water that is a line of water
The highest and lowest contour lines are the lowest and highest contour lines are the lowest
The direction of the declination is in the direction of the declination

Figure 8. Example of Content Filtered Out by the Vocabulary Diversity Rule

EVALUATION

We have conducted a round of formative evaluation of the questions, answers, and to a limited extent, the distracters generated by QGen. The purpose of this formative evaluation was to gain insights into the quality of the content generated. An important step was to create the rubric the SMEs use to rate the content. We were interested in answering the following questions:

Q1: Does QGen generate a high percentage of acceptable questions?
Q2: For acceptable questions, does QGen generate at least one acceptable answer for most questions?

We had a panel of four subject-matter experts (SMEs) to rate the content generated by QGen using a rubric we created. The SMEs come from an Army training background, each with 20+ years of experience. We used the following methodology. We generated questions from different chapters for each of two topic areas. We provided the SMEs with a representative sample of items generated by QGen to rate, along with a rating rubric. The rubric provided a set of criteria for rating questions and answers as Bad, Ok, or Good. They were also asked to indicate if an answer would be useful as a distractor. We coded and analyzed their ratings as discussed below.

To address the concern that the quality of the content generate may depend on the topic, we included two different content areas in this evaluation. Land Navigation (NAV) emphasizes concrete concepts such as how to read maps and measure distances. Law of Armed Conflict (LOAC) deals with abstract concepts and doctrine, such as when a populated area is considered undefended. Of course, including a larger number and variety of topics will significantly increase the reliability of the results. We plan to include other topics in the future. There were 39 LOAC and 35 NAV questions, where each question could have up to six possible answers / distracters. There were 107 LOAC answers and 126 NAV answers.

Figure 9 visualizes the percent acceptable questions and answers by rater. The left column is the percent of Questions with one OK+ rating (i.e., OK or Good), labelled Question. Following this the w/Answer column is the percent of OK+ questions that also had an OK+ rated answer. The third column, w/Distractor, is the percentage of OK+ questions that had both an OK+ answer and a distractor. The final column on the right, labelled Complete, is the percentage of OK+ questions that also had at least three answers that were marked either OK+ or as a distractor. The average value of the columns is 71%, 54%, 16%, and 13% respectively.

The inter-rater agreement analysis is shown in Table 1, where the Krippendorff's Alpha values do not meet the acceptable threshold of 0.667. Unrated items were treated the same as missing values in the calculations (Krippendorff, 2011a; Krippendorff, 2011b).

The distribution of ratings by reviewer is shown in Figure 10. Multiple answers were generated for each question, so the y-axis scale differs between the two charts. Information on the number of distracter answers identified by each of the reviewers is included in the figure.

Figure 9 provides evidence that QGen does create a high percentage of acceptable questions and answers. On average, SMEs found 71% of the questions to be acceptable. Additionally, we examined the distribution of scores by reviewer in Figure 10 and inter-rater agreement in Table 1. While the reviewers tend to have a low level of agreement on what constitutes an acceptable question or answer, they do find a good proportion of them acceptable. This suggests that a typical user would find the majority of generated questions to be useful in creating a test, even if the definition of an acceptable question varies across users.

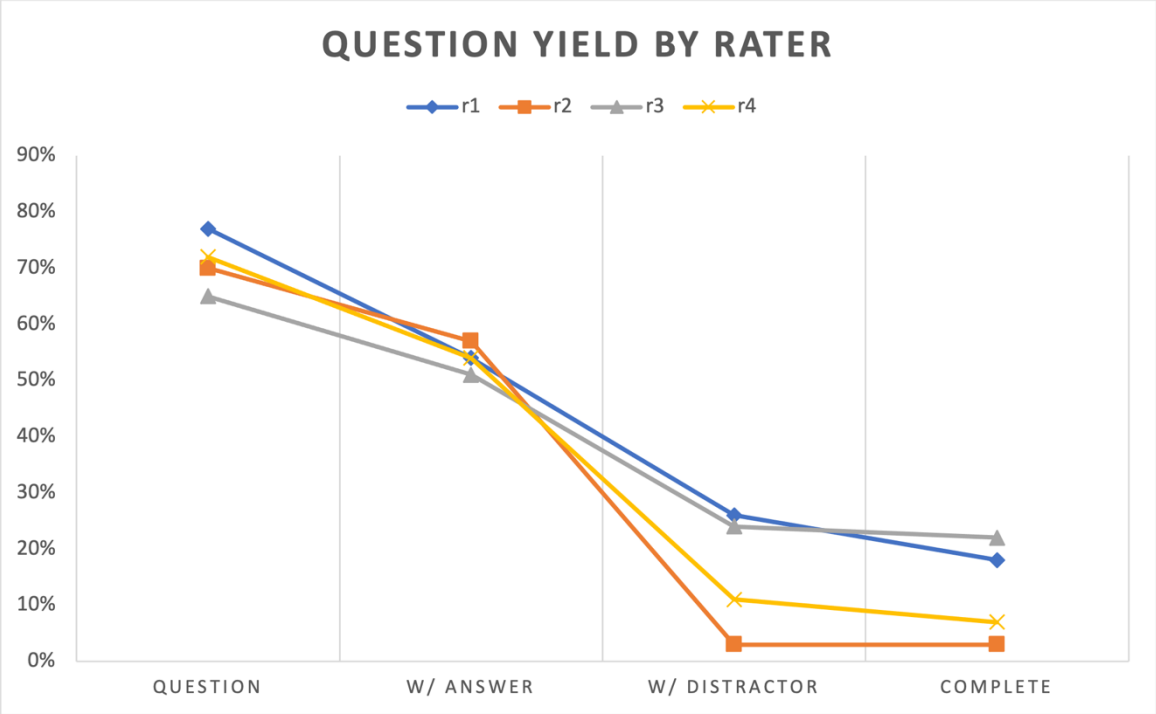

Figure 9. Question Yield by Rater

Table 1. Inter-rater Agreement for the Combined Results

Krippendorff's Alpha	
Questions	0.22
Answers	0.27

Figure 9 also provides evidence that QGen does generate at least one acceptable answer for most questions, as well. On average, SMEs found 54% of the questions to be acceptable and to include at least one correct answer. Additionally, we examined the distribution of scores by reviewer in Figure 10 and inter-rater agreement in Table 1. Similar to questions, the reviewers have a low level of rating agreement for answers but similar rating distributions. This supports greater confidence in the percentage of acceptable questions with an answer as shown in Figure 9. Based on all this information, we expect that if a user selects a question as acceptable to include in a test, then most of the time that question will be accompanied by an acceptable answer.

There are two other interesting aspects of these results. First, there are a lot of bad answer reviews. This is not completely unexpected. QGen generates a matching question-answer pair and then also generates a pool of other answers as possible alternate answers or distractors. For this analysis, the paired answer is always presented first followed by up to five alternate answers. These alternate answers get the bulk of the bad ratings, and we have already worked to improve the filtering of the list to remove obviously bad answers. Second, we asked the raters to identify answers that are incorrect but would make ok or good distracters for acceptable questions. On average, 22% of the answers generated for acceptable questions were identified as potential distractors. However, as shown in the right-most column of Figure 10, one reviewer identified 5 distracters and another 58. That one reviewer found 58 is a positive outcome, suggesting that some valid distracters are being generated. Follow-up discussions with the SMEs about their distractor ratings suggests that that there is divergent opinion even among experts on what makes for a good distractor. This is another reason distractor generation is challenging. We have updated QGen since this study to reduce the number of answers generated. Further studies will examine the effect of this change on the ratings.

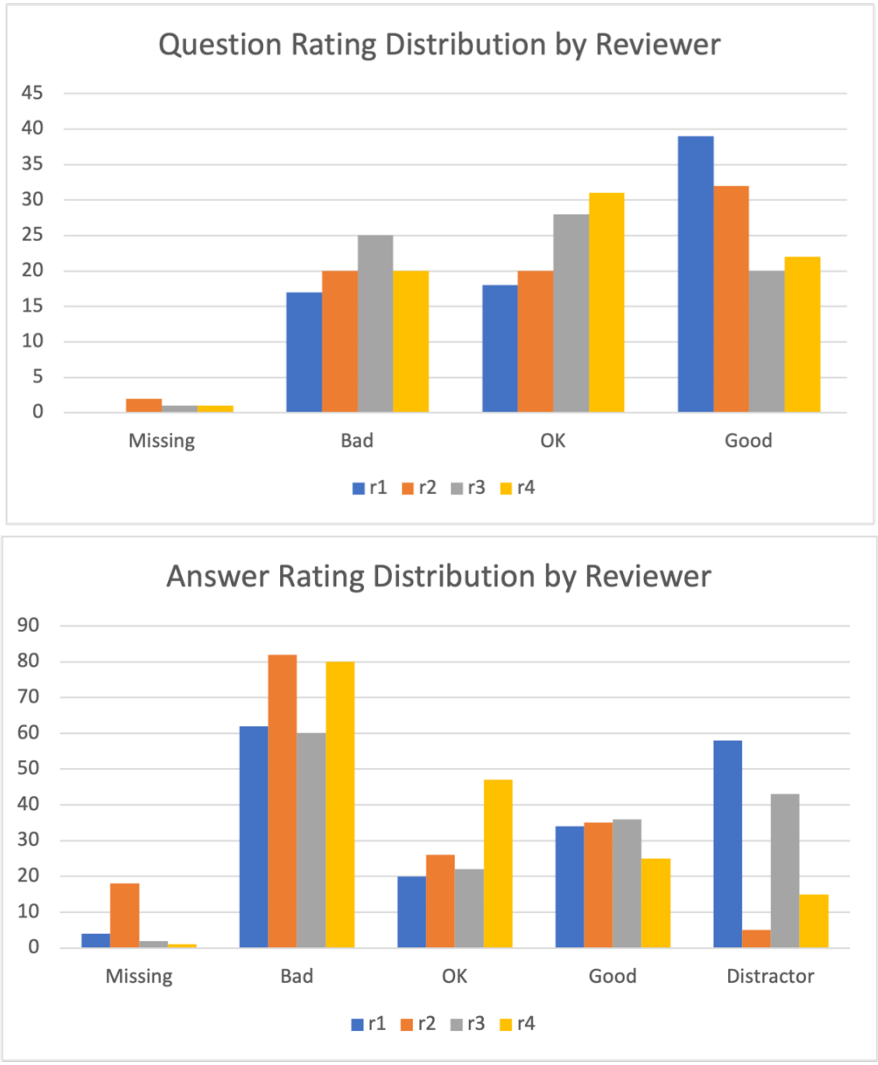

Figure 10. Distribution of Questions and Answers by Reviewer

CONCLUSIONS AND FUTURE WORK

The research described here is ongoing and we continue to update the approach and algorithms to improve the results. A summative validation is planned at the end of the current phase of the project. Our ongoing work is informed by the following observations and findings:

1. The language models, combined with a variety of techniques for pre- and post-processing questions generate a high proportion of good questions and reduced but significant proportion of question-answer pairs. Just this capability could be of benefit in reducing the time for creating certain types of assessments for organizations like the Army and for purposes like certification.
2. The necessity for human input for editing and augmenting the generated test items is going to be unavoidable. However, it is desirable to push the amount of editing required to the low end of the scale.
3. Distractor generation is a challenge that potentially requires intelligent solutions beyond the use of language models.

4. The models cannot explain why they generated certain questions and answers. This makes it difficult to determine how to change the model or the inputs to guide it towards generating better or more useful questions without training a different model altogether.

Moving forward, we plan to continue to improve quality of questions by updating the algorithms and language models used. This includes training language models from scratch and fine-tuning existing models with SME-provided examples of good questions and answers. We are also developing and testing a user-facing web application for QGen. The project will culminate with a set of validation studies to evaluate the usability of QGen as a tool for assessment generation, to study its utility as a time-saving tool, and to test the validity of assessments created using QGen items.

ACKNOWLEDGEMENTS

The research described herein was sponsored by the U.S. Army Research Institute for the Behavioral and Social Sciences, Department of the Army (Contract No. W911NF-20-C-0067). The views expressed in this presentation are those of the author and do not reflect the official policy or position of the Department of the Army, DOD, or the U.S. Government. We also acknowledge the contributions of Mr. Christian Belardi, now a graduate student at Cornell University, to this effort.

REFERENCES

Bird, S., Loper, E., & Klein, E. (2009). *Natural language processing with Python*. O'Reilly Media Inc.

Clark, K., & Manning, C. D. (2016). Deep Reinforcement Learning for Mention-Ranking Coreference Models. *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, 2256–2262. https://doi.org/10.18653/v1/D16-1245

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). BERT: Pre-training of deep bidirectional Transformers for language understanding. *CoRR*, *abs/1810.04805*. Retrieved from http://arxiv.org/abs/1810.04805

FM 6-27, MCTP 11-10C, The Commander's Handbook on the Law of Land Warfare. (2019). Department of the Army, United States Marine Corps.

Krippendorff, K. (2011a). Agreement and Information in the Reliability of Coding. *Communication Methods and Measures*, *5*, 93-112.

Krippendorff, K. (2011b). Computing Krippendorff's alpha-reliability. *Philadelphia: Annenberg School for Communication Departmental Papers*. Retrieved June 28, 2021, from: http://repository.upenn.edu

Manning, Christopher D., Prabhakar Raghavan, and Hinrich Schütze. 2008. *Introduction to Information Retrieval*. USA: Cambridge University Press.

McKie, J., & Liu, R. (2021). *PyMuPDF: Python bindings for the PDF toolkit and renderer MuPDF (1.18.14) [C, Python; MacOS, Microsoft :: Windows, POSIX :: Linux]*. https://github.com/pymupdf/PyMuPDF

Radford, A., Narasimhan, K., Salimans, T., & Sutskever, I. (2018). *Improving Language Understanding by Generative Pre-Training*, 12.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., & Liu, P. J. (2020). Exploring the limits of transfer learning with a unified text-to-text Transformer. *Journal of Machine Learning Research*, *21*(140), 1–67.

Rajpurkar, P., Jia, R., & Liang, P. (2018). Know what you don't know: Unanswerable questions for SQuAD. *ArXiv Preprint ArXiv:1806.03822*.

Tran, Khoi-Nguyen, Jey Han Lau, Danish Contractor, Utkarsh Gupta, Bikram Sengupta, Christopher J. Butler, and Mukesh Mohania. 2018. *Document Chunking and Learning Objective Generation for Instruction Design*. arXiv:1806.01351. arXiv. doi: 10.48550/arXiv.1806.01351.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention is all you need. *Advances in Neural Information Processing Systems*, 5998–6008.

Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R., & Le, Q. V. (2020). XLNet: Generalized Autoregressive Pretraining for Language Understanding. *ArXiv:1906.08237 [Cs]*. http://arxiv.org/abs/1906.08237