# Generate: A NLG system for educational content creation

Saad M. Khan
FineTune Learning
saad@finetunelearning.com

Jesse Hamer
FineTune Learning
jesse@finetunelearning.com

Tiago Almeida
FineTune Learning
tiago@finetunelearning.com

## ABSTRACT

We present Generate, a AI-human hybrid system to help education content creators interactively generate assessment content in an efficient and scalable manner. Our system integrates advanced natural language generation (NLG) approaches with subject matter expertise of assessment developers to efficiently generate a large number of highly customized and valid assessment items. We utilize the powerful Transformer architecture which is capable of leveraging substantive pretraining on several generic text corpora in order to produce sophisticated, context-dependent text as the basis for item creation. We present early results from experimental studies demonstrating the efficiency of our approach.

## Keywords

NLP, Transformer Networks, Domain Knowledge Modeling

## 1. INTRODUCTION

The COVID-19 pandemic has accelerated the push towards remote delivery of formative and summative assessments and with it have arisen heightened security concerns of item pool exposure. Moreover, there is growing adoption of highly personalized and adaptive learning and assessment experiences [25] that require regularly replenished assessment item pools. These twin factors among others are placing ever growing demands on traditional processes of creating assessment content that are based in large part on manual labor, highly dependent on subject matter expertise and challenging to scale up. Furthermore, the manual generation of content and assessment items heightens the risk of incomplete, duplicate and/or redundant content. We believe advances in AI, particularly natural language generation (NLG) can help mitigate this bottleneck and open new possibilities for personalized learning experiences.

Classical natural language processing (NLP) work in this area dates back to John Wolfe's seminal work [17] that demonstrated the feasibility of automatically generating natural language questions. In recent years there has been a revival in interest, spurred in part by advances in dialogue systems such as Amazon Alexa. While traditional approaches to NLP-based assessment item generation involve a pipeline of modules such as content selection, template design and item realization [18], these have been criticized for being rigid and too reliant on arbitrary heuristic rules and having limited novelty and psychometric variability [19]. There is growing interest in developing end-to-end deep neural network based approaches that do not require customized, hand crafted rules and are better equipped to generalize across content areas [20]. A key element of such approaches is leveraging large text content databases and well annotated datasets such as BookCorpus [21], SQuAD [22] and Wikipedia. For further details on related work, readers are directed to the survey of state of the art by Kurdi et al. [26].

In this paper we present Generate, a NLG system that efficiently and in real-time creates lexically and semantically appropriate item content, dramatically speeding up assessment item authoring, freeing item writers and subject matter experts (SMEs) from unnecessary work, and can enable personalized learning and assessment experiences. At the core of Generate's content generation capabilities is the Transformer architecture [4] that leverages substantive pretraining on several generic text corpora to produce sophisticated, context-dependent text as the basis for item creation. From a small number of representative items as training samples to learn lexical and semantic structure, Generate is able to produce a wide variety of draft item content. Users utilize an intuitive graphical interface that allows selection of item stems, keys (correct answers) and distractors from a number of generated options.

In the following sections we provide technical details of our system starting with a brief review of Transformers, system implementation and architecture. Following that we present analysis from experimental studies and share thoughts on future directions.

## 2. TECHNICAL APPROACH AND SYSTEM DETAILS

### 2.1 Transformers and NLG

In order to capture the subtlety and breadth of lexical patterns necessary to faithfully generate novel assessment content, we opted to base our NLG engine on the Transformer architecture, which is capable of leveraging substantive pretraining on several generic text corpora in order to produce highly sophisticated, context-dependent token embeddings for a variety of NLP tasks. First proposed in 2017 by Vaswani et al. [4], the Transformer architecture has since revolutionized NLP research, with state-of-the-art performance on benchmarks like the broad GLUE suite of NLP tasks [5] being set by Transformer-based models such as Google's BERT [6] and OpenAI's GPT series [7, 8, 9].

The central idea of the Transformer architecture is to do away with sequential processing of text altogether, as was done traditionally with deep-learning architectures like LSTMs [10] and GRUs [11], and instead process the tokens (words, subwords, and punctuation) of text simultaneously using an operation called *attention*. The variant of attention used in the original formulation of the Transformer architecture, *scaled dot-product attention*, is defined as follows:

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d}})\ V$$

The matrices $Q$ and $K$ are called the *queries* and *keys*, respectively, and each have column-dimension $d$, while the matrix $V$, called the *values*, has column-dimension $d'$. We consider the case when $Q$, $K$, and $V$ are all the same matrix $X$, and the resulting operation is known as *self-attention*. Each row of $X$ corresponds to a context-independent dense embedding of a single token with a small *positional encoding vector* added so that the model can take into account the position of the token in the input text. Thus, self-attention recomputes every token as a linear combination of every other token, where the weights in the linear combination depend on a scaled dot-product similarity (the $\frac{QK^T}{\sqrt{d}}$ term). In order to allow the Transformer to learn several different patterns of lexical interaction, several matrices of weights are used to compute *multi-head self-attention*:

$$\text{MultiHead}(X, X, X) = \text{Concat}(\text{head}_1, \text{head}_2, \ldots, \text{head}_h)W^O,$$

where

$$\text{head}_i = \text{Attention}(XW_i^Q, XW_i^K, XW_i^V),$$

and all of the $W$ matrices consist of learnable weights. After multihead attention is computed, the results are aggregated and resized using a simple single-hidden-layer feedforward neural network, which has its own learnable weights. This combination of multihead attention followed by a feedforward neural network constitutes the fundamental building block of the Transformer architecture: the *Transformer block*. A *Transformer model*, then, is built by chaining together several Transformer blocks, each potentially with its own set of weights.

For its NLG engine Generate utilizes a Transformer model pre-trained for the task of next-token prediction. The Transformer architecture processes the conditioning input text in order to produce a probability distribution over all tokens in the vocabulary. We sample from the vocabulary according to this distribution, and then proceed *auto-regressively*: we process the newly sampled token and use it to produce a new probability distribution and sample a new token. We continue in this way until a maximum token limit is met, or until a stop sequence is produced (such as a newline character '\n').

## 2.2 How Generate Works

As illustrated in figure 1 Generate has five main system architectural components. The first is a React Javascript-based graphical user interface. Through the interface, users can select pre-uploaded AI models, generate an item, visualize and edit the item and visualize the metrics generated by the AI, allowing the user to create a complete item generation flow, from creation to validation. The user interface is linked to the second component which is the Auth0 authentication platform, a third party service specialized in secure authentication and authorization workflows. Once the user is authenticated, the GUI will connect with the third component, Hasura [2]. Hasura serves mainly as a GraphQL API to connect the GUI with the database and the serverless services. The fourth component is the item generation services (SQS Queue and Lambda Worker), which are responsible for interfacing with the NLG engine API with all the advantages of a serverless architecture [3]. The NLG engine API forms the last core component and is responsible for generating content based on the model provided.

Users begin their interaction with Generate by providing specifications of desired content including: a content map of item types and topics to be generated; user-specific writing guidelines

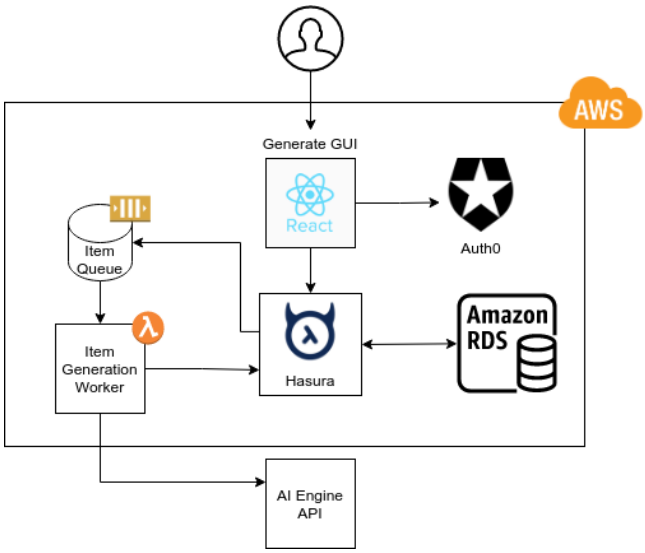


**Figure 1: Generate system architecture is designed to be modular with distributed services hosted on AWS.**

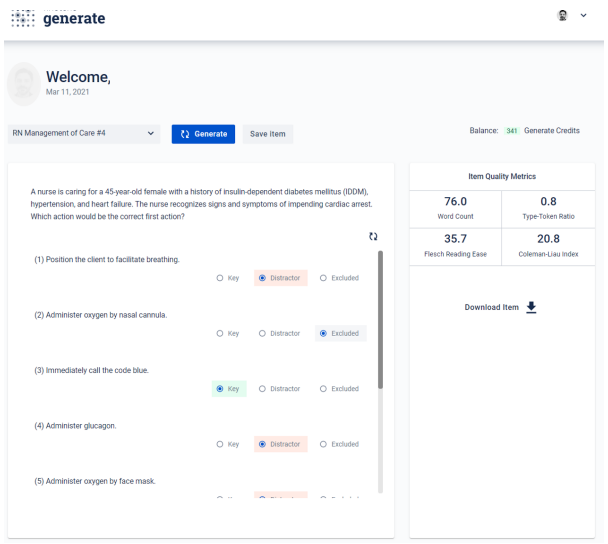


**Figure 2: Generate item authoring interface. Users can select from a number of item generation models and create items on-the-fly with the click of a button.**

so that domain semantics and formatting can be tailored to best practices; and specification of admissible lexical metric ranges, such as type-token ratio, Flesch Reading Ease, and the Coleman-Liau Index. These specifications constitute what is called a *project*. Along with these specifications, users must also supply a set of representative items. The number of such items is usually between 100-200 items total, although it will depend on the complexity of the content map specified in the project and the number of different types of items to be generated. At a baseline, all that is needed is the raw item, though users may supply their own item metadata to help improve the performance of our AI engine. Features such as item topic categorization, key and distractor labels, item cognitive type (recall, application, etc.), and difficulty metrics like $p$-value and point biserial [12] may all be used to further hone our AI models' performance.

After supplying content specifications and representative items, the *k*-means clustering algorithm [13] is applied to produce several groups of 8-10 representative items each. The clustering algorithm is predicated on a combination of user-supplied metadata, as well as numeric item features produced by the Transformer-based Universal Sentence Encoder (USE) model [14]. The goal of this clustering procedure is to produce groups of items which are semantically and stylistically homogeneous, which in turn improves the reliability of our AI engine to produce items which are coherent, semantically and factually relevant to the content domain, and stylistically appropriate according to the user's writing guidelines. Each group of representative items corresponds to a different string of conditioning input text for our AI engine, which we refer to as a *model*. Each model produces a different "flavor" of content. By building several models, we ensure that a user's content specification demands are met, and a wide diversity of items is produced while doing so.

Given a model, raw content is generated by our NLG engine and then undergoes several automated quality checks before being presented to the user. First, the raw content must pass a parse check to ensure that desired item formatting has been captured. Next, we perform an overlap check to ensure that no part of the generated content overlaps too heavily with the representative items, or with any other part of the generated content (e.g. to prevent duplicate options in a multiple choice item). For multiple choice content, users are able to specify a range of options to generate and so we also check that a sufficient number of unique options are produced. Finally, previously specified lexical metrics are computed, and we check that the generated item lies in the user-specified admissible ranges for these metrics.

As shown in figure 2, Generate offers a graphical user interface where item writers interact with our NLG engine directly to produce content. With this interface, item writers can select one of several item generation models and generate items on-the-fly with the click of a button. The item writer can then refine and annotate generated items before saving a finalized version. Generate's content generation interface allows item writers the ability to save an intermediate version of a promising item and regenerate unwanted parts. For example, with multiple choice content, one can select a key and one distractor from the list of available options, and then regenerate the remaining options to produce a fresh list to choose from. In this way, item writers can use Generate to help them rapidly ideate additional options, leading to significant speedups to the item-writing process.

Users can review generated content at any time using Generate's content dashboard. This dashboard allows users to review project-level information such as lexical metric distributions, as well as review individual items and their SME annotations. Once a selection of items has been made, users can download their content either as raw text or in QTI format.

### 2.3 SME Usability Experiment Results

For the content domain of nursing professional licensure, we performed two experiments with a subject matter expert (SME)/item writer in the domain. In the first experiment, the SME was asked to perform a quality review of a set of 40 items purely created by Generate NLG spanning four topic areas including biotechnology, medical assisting, nursing assisting, and practical nursing (see figure 3 for an item from this set). For a baseline of comparison, we mixed in a "calibration set" of 40 representative items produced by a separate human item writer spanning these same four topic areas. The SME was not told which items were

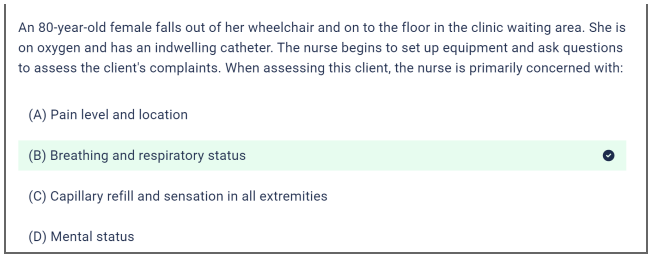


**Figure 3: Sample item created by Generate. A user/SME is able to select the key/correct answer and make any edits required.**

from the calibration set, and which were created by Generate. To perform the quality review, the SME was asked to check factual accuracy and topic relevance, make any necessary edits, estimate the difficulty of the item on a subjective easy/medium/hard scale, give any general comments and feedback, and assign a subjective overall quality rating on a 1-7 Likert scale (with 1 being poor and 7 being excellent). The median quality rating for Generate items was 5.5, compared to a median quality of 6 on the calibration set, with 70% of Generate items rated 5 or higher. There was also considerable overlap in the difficulty distributions, as shown in table 1. It took the SME an average of 3.75 minutes/item to perform this quality review. Compared to the SME's estimated 20-30 minutes/item to write an item manually, Generate demonstrates clear improvements to SME item writing throughput.

In the second experiment, the same SME was asked to interact directly with the Generate content generation interface to produce 50 more items in the domain of nursing professional licensure. We gave the SME five models ranging over a single nursing topic and requested that they produce ten items for each model. We captured data on generation time as well as item survival rate. For each item, the SME used Generate's content generation interface to first generate a multiple choice item with eight possible options. The SME was then asked to select the best combination of key and three distractors from these available eight options, and then perform necessary fact checking and editing. Using the Generate system, it took the SME an average of 2.7 minutes/item, including latency necessary for the system to generate the raw item. According to SME testimony, a similar exercise with a conventional item writing approach would have taken roughly 30 minutes/item, not including slowdowns due to SME fatigue and burnout. We are currently working on a number of follow-on experiments with item writers in a variety of domains including K-12 education, higher-ed and professional licensure.

**Table 1: Comparison of difficulty distributions**

| | Easy | Medium | Hard |
|---|---|---|---|
| Generate | 42.5% | 40% | 17.5% |
| Calibration | 37.5% | 45% | 17.5% |

## 3. DISCUSSION AND FUTURE DIRECTIONS

Our early investigations with item writers indicate a significant increase in assessment authoring throughput, which if borne out in

future and ongoing studies would mitigate bottlenecks in producing easily accessible high quality assessment items. We believe this would help enable innovations in formative assessment, personalized learning, and building customized and efficacious classroom activities.

In addition to assessment content generation, we plan to implement the following functionality to Generate over time.

### 3.1 Item Difficulty Estimation

Content creators must ensure that assessments adhere to a desired difficulty distribution, where we take difficulty to be measured by *p*-value (the proportion of examinees that answered a question correctly). Current methods for estimating *p*-values involve manual field-testing of provisional items, which is both time-intensive and risks item exposure, reducing the lifespan of the item. While previous work in automated difficulty estimation has employed techniques of first-order-logic [15] as well as a machine learning-based word embedding approach [16], we are exploring a blended approach which leverages structured item metadata with Transformer-based processing of unstructured item text. In this way, users can quickly recycle items which do not adhere to required difficulty specifications, thereby increasing the survival rate of items produced by Generate.

### 3.2 Automated Content Tagging

Tagging educational content with the most relevant learning and assessment standards such as CCSS [23], NGSS [24], etc. is one of the most critical elements in creating highly efficacious content. This enables the tracking of student skill gaps, recommendation of remediative learning resources and mastery of discipline topics, skills and cross cutting capabilities. We are currently developing a text content classification approach that can be used to delineate skills, learning objectives and core disciplinary ideas in the generated assessment items.

## 4. CONCLUSION

In this paper we have introduced Generate, a system that utilizes an NLG approach to significantly increase productivity of assessment content creators. Generate is built on a language modeling architecture that understands the deep semantic and lexical structure of assessment content that allow us to handle a variety of assessment domains and item types. Our system's content dashboard integrates elegantly with existing item writer workflows for item review, editing and approval. To the best of our knowledge Generate is the first NLG content authoring system designed for use in education and we believe can enable innovations in personalized learning, formative assessment and efficacious classroom activities.

## 5. REFERENCES


[1] Bowman, M., Debray, S. K., and Peterson, L. L. 1993. Reasoning about naming systems. *ACM Trans. Program. Lang. Syst.* 15, 5 (Nov. 1993), 795-825. DOI= http://doi.acm.org/10.1145/161468.16147.

[2] Hasura's open-source engine gives you instant GraphQL & REST APIs that unify your data and power modern applications - https://hasura.io/.

[3] Servereless architecture advantages - https://blog.newrelic.com/engineering/what-is-serverless-architecture/

[4] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł. and Polosukhin, I. 2017. Attention is all you need. *Advances in Neural Information Processing Systems*, 5998-6008.

[5] Wang, A., Singh, A., Michael, J., Hill, F., Levy, O. and Bowman, S. 2018a. GLUE: A multi-task benchmark and analysis platform for natural language understanding. *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, 353-355.

[6] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. *NAACL-HLT (1)*, 4171-4186. DOI=10.18653/v1/N19-1423.

[7] Radford, A., Narasimhan, K., Salimans, T. and Sutskever, I. 2018. Improving language understanding by generative pre-training. *Technical report. https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf.*

[8] Radford, A., Wu, J., Child, R., Luan, D., Amodei, D. and Sutskever, I. 2019. Language models are unsupervised multitask learners. *Technical report https://openai.com/blog/better-language-models/.*

[9] Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. 2020. Language models are few-shot learners. *Advances in Neural Information Processing Systems.*

[10] Hochreiter, S. and Schmidhuber, J. 1997. Long short-term memory. *Neural Computation* 9, 8, 1735-1780.

[11] Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H. and Bengio, Y. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *Proceedings of the 2014 Conference on EMNLP* (Oct. 2014), 1724-1734. DOI=10.3115/v1/D14-1179

[12] Glass, G. and Hopkins, K. 1995. Statistical Methods in Education and Psychology (3rd ed.). *Allyn & Bacon.* ISBN 0-205-14212-5.

[13] Lloyd, S. 1982. Least squares quantization in PCM. *IEEE Transactions on Information Theory* 28, 2, 129-137. DOI=10.1109/TIT.1982.1056489.

[14] Cer, D., Yang, Y., Kong, S., Hua, N., Limtiaco, N., St. John, R., Constant, N., Guajardo-Céspedes, M., Yuan, S., Tar, C. et al. 2018. Universal sentence encoder. *Proceedings of the 2018 Conference on EMNLP: Demonstrations*, (Nov. 2018), 169-174, DOI=10.18653/v1/D18-2029.

[15] Perikos, I., Grivokostopoulou, F., Kovas, K. and Hatzilygeroudis, I. 2016. Automatic estimation of exercises' difficulty levels in a tutoring system for teaching the conversion of natural language into first-order logic. *Expert Systems* 33, 6 (Dec. 2016), 569-580. DOI=https://doi.org/10.1111/exsy.12182.

[16] Hsu, F.-Y., Lee, H.-M., Chang, T.-H. and Sung, Y.-T. 2018. Automated estimation of item difficulty for multiple-choice tests: An application of embedding techniques. *Information*

*Processing & Management* 54, 6 (Nov. 2018), 969-984. DOI=https://doi.org/10.1016/j.ipm.2018.06.007.

[17] Wolfe, J., 1977. Reading retention as a function of method for generating interspersed questions. *Technical report, DTIC Document*

[18] Gierl, M., Lai, H., Turner, S., 2012. Using automatic item generation to create multiple-choice test items. *Medical Education* 46, 8 (July 2012), 757-65. DOI=10.1111/j.1365-2923.2012.04289.x.

[19] Heilman, M. 2011. Automatic factual question generation from text, *Ph.D. thesis, Carnegie Mellon University.*

[20] Cervone, A., Khatri, C., Goel, R., Hedayatnia, B., Venkatesh, A., Hakkani-Tur, D. and Gabriel, R. 2019. Natural language generation at scale. *arXiv preprint arXiv:1903.08097.*

[21] Zhu. Y., Kiros, R., Zemel, R., Salakhutdinov, R., Torralba, A. and Fidler, S. 2015. Aligning books and movies. *In Proceedings of the IEEE ICCV*, 19-27.

[22] Rajpurkar, P., Zhang, J., Lopyrev, K. and Liang, P. 2016. SQuAD: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*.

[23] National Governors Association Center for Best Practices, Council of Chief State School Officers 2010. Common Core State Standards. *National Governors Association Center for Best Practices, Council of Chief State School*.

[24] NGSS Lead States. 2013. Next Generation Science Standards: For states, by states. *Washington, DC: The National Academic Press*.

[25] Pane, J. F., Steiner, E. D., Baird, M. D., & Hamilton, L. S. (2015). Continued Progress: Promising Evidence on Personalized Learning. Rand Corporation.

[26] Kurdi, G., Leo, J., Parsia, B., Sattler, U., & Al-Emari, S. (2020). A systematic review of automatic question generation for educational purposes. International Journal of Artificial Intelligence in Education, 30(1), 121-204.