Gatekeeper to save COGS and improve efficiency of Text Prediction

Nidhi Tiwari, Sneha Kola, Milos Milunovic, Si-Qing Chen and Marjan Slavkovski
Microsoft Ltd
(nidhitiwari,snehakola,sqchen,mmilunovic,mslavkovski)@microsoft.com

Abstract

The text prediction (TP) workflow in editor calls a Large Language Model (LLM), after a character is typed by the user to get subsequent sequence of characters. The confidence score of the prediction is used for filtering the results to ensure that only correct predictions are shown to user. As LLMs require massive amount of computation and storage, such an approach incurs high execution cost. So, we propose a Model gatekeeper (GK) to stop the LLM calls that will result in incorrect predictions at client application level itself. This way a GK can save cost of model inference and improve user experience by not showing the incorrect predictions. We demonstrate that use of a model gatekeeper saved $\approx 46.6\%$ of COGS (Cost Of Goods Sold) for TP, at the cost of $\approx 4.5\%$ loss in character saving. Use of GK also improved the efficiency (suggestion rate) of TP model by 73%.

1 Introduction

Large Language Models (LLMs), such as Generative Pre-trained Transformers (GPT-2, GPT-3) and Turing Natural Language Generation (T-NLG) models, have billions of parameters. These can be fine-tuned for various Natural Language Processing (NLP) tasks, such as text classification, question answering and text prediction. Our text editor application uses a distilled version of one such large text prediction model to provide text suggestions when user types in editor boxes. This improves users' writing productivity and reduces grammar and spelling errors. This application calls a large text prediction (TP) model after every keystroke (i.e. a character is typed) to show text completion suggestions. The last 256 character(s) typed by a user are sent to this model to get subsequent text prediction with confidence score. The editor application considers only the predictions that have a high confidence score. But, these confidence values are available only after model inference. As these models have a large number of parameters, they require large number of floating point operations (FLOPs) for an inference. We perform another round of quantization aware distillation to reduce the latency and host it on cloud from where it is accessed by millions of users. Such a large number of inferences incur high Cost of Goods Sold (COGS). Furthermore, it has been observed that they are typically overconfident in their predictions on out-of-distribution (OOD) data (Lakshminarayanan et al., 2017; Guo et al., 2017). So, when the LLM provides outputs for input examples that are far from distribution in training set (i.e. OOD), their predictions can be arbitrarily bad. These false positives from model will reduce the reliability of application and result in poor user experience.

To maintain user confidence, save unrewarding COGs and delay, we propose to have a model gatekeeper for LLM. A model gatekeeper filters out the inputs for its large model. A model gatekeeper is small in size so that it can be used at edge to stop calls for model inferences for the inputs that may result in incorrect prediction. This provides average latency, performance and cost advantages for enterprises hosting large models.

Gatekeeper is a binary classification model trained using the large model's evaluation data. For a given input, gatekeeper predicts 0, if large model may return a valid prediction else predicts 1. We developed and evaluated gatekeeper for a large text prediction model using publicly-available data and internal data. We demonstrate that the model gatekeeper is capable of identifying relevant inputs for text prediction model. Use of gatekeeper improved the suggestion rate (i.e. the percentage of times the server model was able to provide predictions with a confidence score higher than set threshold) by $\approx 70\%$ and reduced the COGS by $\approx 47\%$. The reduction in inferences from large model resulted in better user experience and reduced COGS.

2 Related Work

Multiple researchers (Nguyen and O'Connor, 2015), (Nguyen et al., 2015) have established that softmax prediction probability is a good baseline for error and out-of-distribution (OOD) detection across several architectures of Deep Neural Networks (DNNs). (Hendrycks and Gimpel, 2016) defined the confidence score as a maximum value of the predictive distribution. They demonstrated that, while these prediction probabilities create a consistently useful baseline, at times they are less effective. (Guo et al., 2017) improved their performance by using temperature scaling (that uses a single scalar parameter $T > 0$ for all classes to "soften" the softmax (i.e. raises the output entropy) with $T > 1$.

Although such inference methods are computationally simple, they depend on how well the base model was trained. So, there has been a lot of effort in improving the training of base DNN models for better OOD and uncertainty determination. (Liang et al., 2017) utilizes temperature scaling with input perturbations using the OOD validation dataset to tune hyper-parameters of base model. (Hendrycks et al., 2019) and (Rawat et al., 2021) proposed data augmentation methods to generate out-of-domain samples, then use them to train the base model for improved OOD detection. (Lee et al., 2018) proposed jointly training a generator and a classifier, the generator produces examples that appear to be at the boundary of the data manifold to serve as out-of-distribution examples, while the classifier is encouraged to assign these uniform class probabilities. (Kendall and Gal, 2017) and (DeVries and Taylor, 2018) train neural networks that produce two outputs: a prediction and an uncertainty estimate. (Woodward et al., 2020), (Li et al., 2021) proposed separate confidence estimation modules on top of end-to-end (E2E) models, which are classification models trained to minimize the binary cross entropy between the estimated confidence and the target. Bayesian probabilistic models (Louizos and Welling, 2017) and ensembles of classifiers (Lakshminarayanan et al., 2017), learn a distribution over weights during model training for estimating the predictive uncertainty. However, these require significant modifications to the training procedure and are computationally expensive compared to standard (non-Bayesian) neural networks (NNs).

Above methods improve the estimation of prediction confidence, thus, require model execution, while our method stops model execution if output would be unreliable. Most of them do uncertainty calibration with the base model training. However our method can be used for an existing model, without any information about their training data.

Our gatekeeper is similar to selective prediction approaches. Selective prediction is also commonly used to increase the reliability of machine learning models (Kamath et al., 2020). In selective prediction, a calibrator is used to preemptively filter out model inputs whose prediction score will not clear the system threshold. (Varshney et al., 2022) proposed a calibrator model trained using the difficulty level of the instances and confidence scores. This approach needs to be executed along with base model training for the difficulty level calculation. (Kamath et al., 2020) proposed a calibrator model for selective question answering under domain shift. The calibrator is trained using QA system prediction confidence scores on held-out source data and known OOD data. Thus it requires access and knowledge of base model training data. (Garg and Moschitti, 2021) distill the knowledge of QA models into Transformer-based question filtering model. To train such a student model knowledge of teacher QA model architecture is required. However, for a GK training the base model's architecture is not required, it can work for black-box base models.

Our approach uses the base model's test and/or execution data (input and output) to train the Gatekeeper model. Gatekeeper learns a relationship between inputs, in and/or out of model source domain, and outputs that are below confidence threshold.

3 Problem Formulation

The TP model (a LLM) returns accurate predictions but is costly. Current text prediction workflow calls this model, after the user types a character to get subsequent sequence of characters. The high call rate further increases the COGS (Cost Of Goods Sold) of text prediction workflow .

The server-side model returns a confidence score (ranging from -1000 to 1), along with text prediction, to indicate the quality of the prediction. The predictions having score less than a rendering threshold (based on model evaluation study) are not shown to user. In a production environment, the Suggestion Rate was less than 10%. This means that more than 90% of requests to the server model result in a prediction that is not good enough to show to user.

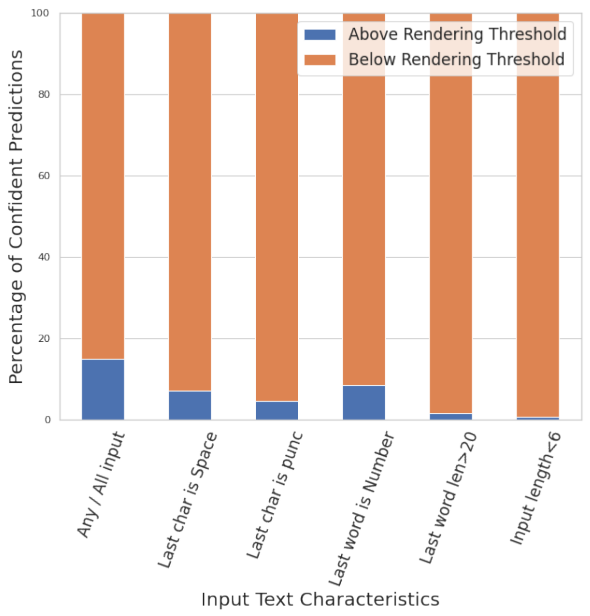

Figure 1: Selected results from exploratory data analysis

In offline analysis, we observed that TP model returned a very low confidence score for $\approx 18\%$ of the tested records. Figure 1 shows the percentage split of confidence scores for the key observed text characteristics. These observations made us think: *Can we identify such input texts which will get no or poor quality response from TP model? Can we ensure that every call made to large model returns useful response and is actually worth its cost?*

4 Gatekeeper

We propose a gatekeeper (GK) to suppress only the unrewarding calls to large TP model, while maintaining the overall performance. Gatekeeper is a light weight classifier on client-side that can predict the probability of getting a NULL from the TP model. Figure 2 shows the text prediction workflow with a GK. Here, for each user request, the client application invokes the local GK (passing up to the last 256 character) to find the probability of getting an incorrect prediction. If this probability is low, then client application calls large TP model and shows its prediction (based on rendering threshold); else doesn't show any suggestion (waits for the next character input and so on). This way the unrewarding executions of the server-model and COGS are reduced; user is not bothered with incorrect suggestions and delays.

The output of GK is 1 when the probability of poor response from the large TP model for a given

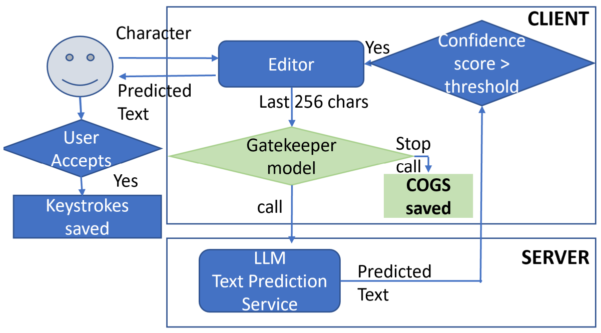

Figure 2: Gatekeeper to make or stop calls to LLMs

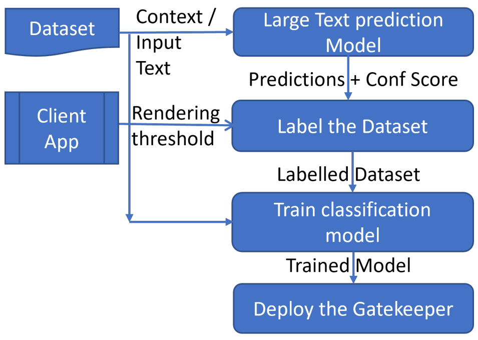

Figure 3: Gatekeeper development approach

input is high, else it is 0.

$$GK(x) = \begin{cases} 1, & if\ \text{TP}(x) < \textit{rendering threshold}, \\ 0, & otherwise. \end{cases} \tag{1}$$

We use LLM responses to build its gatekeeper. Our post-hoc approach can be used to add a GK in existing intelligence workflows to detect incorrect predictions, without retraining the LLM model in use. Figure 3 shows the approach used to train and deploy a gatekeeper model for the existing TP model. Given that there is a well trained TP model, we send the context text as input and get a text sequence prediction with its confidence score. Next, we use the rendering threshold to set the target labels for gatekeeper model training. For example, if the rendering threshold is -0.75 and confidence score (of predicted sequence) is -1, then the binary target is 1; as it is less than threshold. This data is used by gatekeeper to learn the function, f(x), shown in equation 1. We can use this approach to tune the gatekeeper for different products/domains using their input and output from the TP model.

5 Experimental Setup

5.1 Evaluation Metrics

The TP model is used to improve user productivity by reducing the number of characters to be typed in the text editor. So we use following metrics to measure the effectiveness of TP model:

- **Character savings** - Number of characters in predicted sequences that had confidence score greater than threshold and matched the characters typed by user. (This is a proxy metric based on the assumption that by predicting correct sequence we save users' keystrokes)
- **Suggestion rate** – Percentage of times the large model was able to provide predictions with a confidence score higher than rendering threshold.

Introduction of TP gatekeeper should not have a negative impact on the performance of TP model and should reduce the COGS. So, we measure the effectiveness and efficiency of gatekeeper using following metrics:

- **COGS reduction percentage** (**$COGSRed\%$**) (Higher is Better)– Number of times server model is invoked, estimated as server hit rate. For example: For a paragraph of length of 100 characters, server model is called 100 times, then with a gatekeeper model, server model will be called 90 times to get 10% saving.

$$\frac{(COGS_{org} - COGS_{withGK})}{COGS_{org}} * 100 \quad (2)$$

- **Character savings loss percentage** (**$CharSavLoss\%$**) (Lower is Better) - Lesser number of calls to TP model may reduce the number of correct predictions also. Thus measuring the percentage loss in character saving due to use of gatekeeper.

$$\frac{(CharSav_{org} - CharSav_{withGK})}{CharSav_{org}} * 100 \quad (3)$$

- **COGS reduction to Character savings loss Ratio** (**$GKEfficiency$**) (Higher is Better) - To measure the trade-off between COGS saving and loss in character savings, their ratio is used. A GK is efficient if COGS saving is multiple times of the resulting loss in character savings.

$$\frac{COGSRed\%}{CharSavLoss\%} \quad (4)$$

- **Suggestion rate improvement (SugRateImprv%)** (Higher is Better) – Percentage of times the large model was able to provide predictions with a confidence score higher than rendering threshold.

$$\frac{(SugRate_{org} - SugRate_{withGK})}{SugRate_{org}} * 100 \quad (5)$$

We use "No Gatekeeper" as baseline to calculate the above metrics. "No Gatekeeper" is the original TP scenario, where client sends all requests to server and responses having confidence score greater than rendering threshold are considered.

5.2 Datasets

We collected data from public data sources – wiki, books, documents, news and Technical Support Guide (TSG). The sentences in the data sets were converted into input-output (context, prediction) format, for testing the TP model. The input (i.e. context) from this formatted data was used to obtain the text predictions and prediction scores from TP model.

We used the input and output from TP model evaluation to build its gatekeeper. The input to gatekeeper is same as the input to TP model, as it determines the prediction. The expected output from the gatekeeper was determined based on the confidence score from the TP model. If confidence score is greater than rendering threshold then output is 0 else 1. The PROD environment used a rendering threshold (ren_thresh) derived after multiple experiments, so we used following criteria to define labels for Gatekeeper model training:

$$GK(x) = \begin{cases} 1, & if \text{ TP}(x) < \text{ren\_thresh}, \\ 0, & otherwise. \end{cases} \quad (6)$$

Table 3 in Appendix A shows a sample of data used for building the TP GK.

We used same steps to generate labels for each of the five datasets, merged all the data sets. We created the training (60%), validation (20%) and test (20%) splits from the 5M+ records. We used training split for model training, validation split for model fine-tuning after each epoch and test split for the final model evaluation.

The TP model is a proprietary model, tuned using internal data. The data used to train and test GK model was generated using this model, so cannot share the dataset.

5.3 Gatekeeper

We experimented with 2 types of gatekeeper - rule-based and model-based.

5.3.1 Rule-based Gatekeeper

We formulated rule-based GK on the analysis of TP evaluation results, where we observed that certain input texts almost always got no response or low confidence from the TP model. The rule-based gatekeeper uses one or more of these simple rules to stop the calls to the server model. For the "all rules" scenario, we combined following checks using "OR" operation for this rule-based gatekeeper:

- length of input text is less than 6 characters
- last character is space
- last character is a punctuation mark
- last char is a digit
- last word is name of a number
- length of last word is greater than 20 characters

5.3.2 Model-based Gatekeeper

We developed a model-based GK for TP model using the below approach.

Model selection: First, we evaluated multiple classification models such as logistic regression, tree based ensemble models (Adaboost, LGBM) and neural networks with 2 types of NLP features:.

- Character count vectorizer, specifically, bi-gram of characters.
- Text features such as the number (#) of words, # of capital words, # of punctuation, # of stop-words, input length, etc. These features were based on our exploratory data analysis.

However most of these models had low F1-scores (0.44-0.55).

Next, we experimented with Transformer-based gatekeeper. Given the fact the input is text sequence and the large TP model is a transformer model, these models had higher performance. Based on the model performance and size, we finalized on using Tiny BERT. Tiny BERT is a smaller variant of BERT model (Turc et al., 2019) that gives good results and satisfies our computational constraints of 3-5MB disk size and 20MB RAM usage at peak. So, we fine-tuned the TinyBert model[1]. Our model architecture consists of standard Transformer encoder followed by a single classification layer that performs binary classification. We used area under the receiver operating characteristics (ROC) curve (AUC) to tune the model.

The model with selected hyper-parameters (detailed in Appendix B) converged to 0.88 AUC. Model had AUC of 0.864 and 0.858 on validation and test sets, respectively. As *tiny-bert tokenizer* is not available in ONNX (Open Neural Network Exchange[2]), we used standard "bert-base-cased" tokenizer and included it in the model pipeline, converted it to ONNX format and quantized it to *uint8* for optimized execution. The final size of transformer-model gatekeeper was ≈ 4 megabytes (MB). It had peak memory usage of 24.3 MB on $x64$ and took 3.52 milliseconds (ms) on average (including tokenization) for inference.

[1] https://huggingface.co/prajjwal1/bert-mini

[2] https://onnx.ai/

6 Results Analysis and Discussion

We evaluated the performance of GK by executing a pipeline of gatekeeper and TP model on test set. The data in test set was not used during gatekeeper model training or validation. We used these results to determine the threshold for transformer-model GK and select the type of gatekeeper.

6.1 Model-based GK at different thresholds

Figure 4 shows the improvement in COGS saving and reduction in character saving metrics at different thresholds of the Gatekeeper (GK) model. We observe that as the threshold increases, the COGS saving reduce at a high rate while loss in character saving reduces at a lower rate. GK model provides the probability of not getting a response from the large model.When a low threshold is used for GK, it stops the call even if probability of getting a wrong response is low. This reduces the number of predictions which lowers the probability of getting expected response and thus reduces the saving on character typing. However, GK model has high precision (0.9 on average) at high threshold, it allows more calls and correct text predictions, which result in higher character savings.

6.2 Rule-based and/or Model-based GK

Considering that space rule provides high COGS savings and *bert-tokenizer* removes the spaces, we combined them to create a rule+model based GK. We created GK using different combinations of rule and model. Table 1 shows results of using different types of gatekeepers on the combined test set. Based on these evaluation results, we finalized on the transformer-based GK for the TP model.

We observed that combining of various rules increased the loss in char savings, almost incrementally, but didn't increase COGS saving proportionally. Also selecting a set of heuristics by means of A/B experiments would require a significant number of experiments. So, it was hard to find out the best way to combine them.

In fact, the model-based GK can be used with

Gatekeeper	SugRateImprv% (↑)	COGSRed% (↑)	CharSavLoss% (↓)	GKefficiency(↑)
Space Rule	10.30	16.55	7.84	2.11
All rules (except space)	6.65	8.50	2.07	4.11
All rules	16.95	22.74	9.62	2.36
Model@0.9	**73.80**	**46.61**	**4.50**	**10.36**
Space-Rule+Model@0.9	78.99	52.03	11.81	4.41
All rules+Model@0.9	80.72	53.29	13.45	3.96

Table 1: Text prediction performance metrics on complete test set when Rule and/or Model Gatekeeper is used.

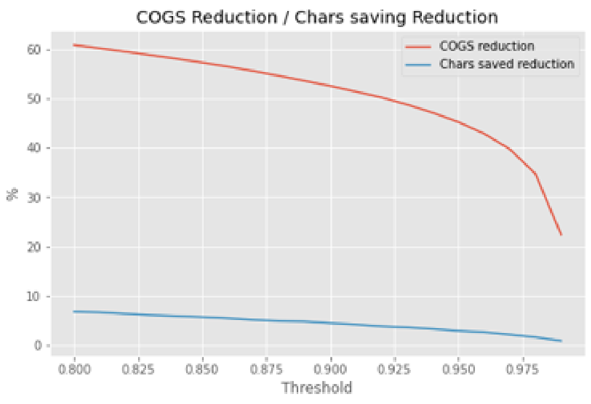

Figure 4: COGS saving and loss in Character saving at different thresholds on doc's test set

TP Prediction	IsAMatch	GK Prediction
nan	FALSE	TRUE
ight and	TRUE	FALSE
nizations	TRUE	FALSE
nan	FALSE	TRUE
nan	FALSE	TRUE

Table 2: Sample of text prediction and gatekeeper predictions on docs dataset.

different thresholds. We observed that the trade-off (between COGS saving and char saving loss) varies for different evaluation sets. So a tuned threshold, transformer-based GK can be used for different products, such as for docs and emails. Likewise, different thresholds can be used for different customers/domains i.e books, wiki.

6.3 Model-based GK errors analysis

In this section we analyze the errors of GK model. GK model error is defined as blocking a call for which large TP model predicts correctly. Table 2 shows a random sample of results with 0.9 as the GK's threshold for "docs" test set. We observed that for a large number of rows where GK model predicted "True" (stop the call), prediction from TP model was not matching with the expected output or was "NULL". Mostly, when GK model predicted "FALSE", the TP response was matching with expected response. Overall, only 0.11% of stopped calls would have gotten correct response for the end user, in case of "docs" test set. More examples are provided in Appendix C

7 Conclusion and Future Work

In this paper we presented a gatekeeper to improve the usage efficiency of LLM. The model GK is designed to the reduce number of executions of LLM without negatively impacting the overall performance of scenario. We developed a gatekeeper for large TP model using its evaluation results. We demonstrated that the model-based gatekeeper improves large TP model's efficiency (i.e. ratio of COGS increase to char saving decrease) by $\approx$ 10 times at a threshold. In production, we observed that GK (transformer + rule) provided $\approx$ 55% COGS saving with less than 1% loss in character saving (when 5% is acceptance criteria) for a set of web-client users.

We plan to test and tune the transformer-based gatekeeper for a few large TP models, to establish the generality of the GK. We will develop gatekeepers for other text sequence models, such as grammar and sentence correction, for reducing their COGS without impacting the user experience. We need to ensure that model gatekeepers are developed and updated at the same pace as the LLM are being released. So, we plan to use Continuous Learning algorithms to update the gatekeeper models in dynamic environments.

Limitations

We acknowledge following limitations in current work. We plan to address them in future.

- In this work, we focused on English text editor and experimented with only English datasets. In the future, we would like to develop and test COGS saving gatekeepers for other languages.
- We understand that the current approach requires the GK to be tuned/updated for every change in server side TP model.

Ethics Statement

We adhered to following principles during the design, data collection, analysis, and reporting of this work.

- We used open data sources to evaluate TP model and train/test/validate GK model.
- We have done the data analysis and reporting without any data manipulation or hiding. We have comprehensively shared the research methods, results and observations in this paper without any bias. We have tried to share any potential conflicts of interest that we know.
- We have shared all sources of information, including previous research contributions, to the best of our knowledge.

This ethics statement demonstrates our commitment to conducting research with the utmost ethical considerations, upholding the welfare and rights of all participants involved.

Acknowledgments

We would like to thank Paul Karimov and Olivier Gauthier for guiding, helping and supporting us at different stages of this work.

References

Terrance DeVries and Graham W Taylor. 2018. Learning confidence for out-of-distribution detection in neural networks. *arXiv preprint arXiv:1802.04865*.

Siddhant Garg and Alessandro Moschitti. 2021. Will this question be answered? question filtering via answer model distillation for efficient question answering. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 7329–7346, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. 2017. On calibration of modern neural networks. In *International conference on machine learning*, pages 1321–1330. PMLR.

Dan Hendrycks and Kevin Gimpel. 2016. A baseline for detecting misclassified and out-of-distribution examples in neural networks. *arXiv preprint arXiv:1610.02136*.

Dan Hendrycks, Norman Mu, Ekin D Cubuk, Barret Zoph, Justin Gilmer, and Balaji Lakshminarayanan. 2019. Augmix: A simple data processing method to improve robustness and uncertainty. *arXiv preprint arXiv:1912.02781*.

Amita Kamath, Robin Jia, and Percy Liang. 2020. Selective question answering under domain shift. In *Annual Meeting of the Association for Computational Linguistics*.

Alex Kendall and Yarin Gal. 2017. What uncertainties do we need in bayesian deep learning for computer vision? *Advances in neural information processing systems*, 30.

Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. 2017. Simple and scalable predictive uncertainty estimation using deep ensembles. *Advances in neural information processing systems*, 30.

Kimin Lee, Kibok Lee, Honglak Lee, and Jinwoo Shin. 2018. A simple unified framework for detecting out-of-distribution samples and adversarial attacks. *Advances in neural information processing systems*, 31.

Qiujia Li, David Qiu, Yu Zhang, Bo Li, Yanzhang He, Philip C. Woodland, Liangliang Cao, and Trevor Strohman. 2021. Confidence estimation for attention-based sequence-to-sequence models for speech recognition. *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6388–6392.

Shiyu Liang, Yixuan Li, and Rayadurgam Srikant. 2017. Principled detection of out-of-distribution examples in neural networks. *ArXiv*, abs/1706.02690.

Christos Louizos and Max Welling. 2017. Multiplicative normalizing flows for variational bayesian neural networks. In *International Conference on Machine Learning*, pages 2218–2227. PMLR.

Anh Nguyen, Jason Yosinski, and Jeff Clune. 2015. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 427–436.

Khanh Nguyen and Brendan O'Connor. 2015. Posterior calibration and exploratory analysis for natural language processing models. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1587–1598, Lisbon, Portugal. Association for Computational Linguistics.

Mrinal Rawat, Ramya Hebbalaguppe, and Lovekesh Vig. 2021. Pnpood: Out-of-distribution detection for text classification via plug andplay data augmentation. *arXiv preprint arXiv:2111.00506*.

Iulia Turc, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Well-read students learn better: On the importance of pre-training compact models. *arXiv preprint arXiv:1908.08962*.

Neeraj Varshney, Swaroop Mishra, and Chitta Baral. 2022. Towards improving selective prediction ability of NLP systems. In *Proceedings of the 7th Workshop on Representation Learning for NLP*, pages 221–226, Dublin, Ireland. Association for Computational Linguistics.

Alejandro Woodward, Clara Bonnín, Issey Masuda, David Varas, Elisenda Bou, and Juan Carlos Riveiro. 2020. Confidence measures in encoder-decoder models for speech recognition. In *INTERSPEECH*.

A Data Samples

Here's a snapshot of the training data created using TP model evaluation.

Input	confScore	Label
"If you aren't co	-1000	1
"If you aren't com	-0.5923	0
"If you aren't comp	-0.71796	0
"If you aren't compl	-1000	1
"If you aren't comple	-1000	1

Table 3: Sample of training data for GK model.

B GK model Hyper-parameters

We tuned the model using *AdamW* optimizer and *BSEWithLogitsLoss* as loss function on 4 Nvidia A100 GPUs. The training batch size is set to 8. The distribution of labels was highly skewed; in the 3 splits, almost 85% of examples had prediction score less than the rendering threshold. To ensure that a training batch contains equal number of examples of the two classes, we use a weighted random sampler utility, *WeightedRandomSampler*, of pytorch library for data sampling in each batch. We ran a sweep over learning rate, maximal input sequence length and optimizer epsilon to find out their optimal values for our data. The model is trained for 5 epochs, with a learning rate of 0.0003, sequence length of 128 and epsilon of $1e-8$. Our tuned model consists of standard Transformer Encoder followed by a single classification layer that performs binary classification.

C Model-based GK samples

Tables 4 and 5 show random samples of responses from TP model and if that predicted string was matching to expected response for "wiki" and "TSG" test sets, respectively. These Tables also have a column indicating if GK would have stopped that call (and thus that prediction). We observe that for a large number of rows, GK model predicted "True" (stop the call), when prediction from TP model was "NULL". Also a large number of rows, when GK model predicted "FALSE", the TP response was matching with expected response. Overall, only 0.48% and 0.28% of stopped calls would have gotten correct response for the user, in case of "wiki" and "TSG" test set.

TP Pred	IsAMatch	GK Pred
nan	FALSE	TRUE
nd	FALSE	FALSE
nan	FALSE	TRUE
hare to	TRUE	FALSE
e	FALSE	TRUE

Table 4: Sample of text prediction and gatekeeper predictions on wiki dataset.

TP Pred	IsAMatch	GK Pred
ow	TRUE	FALSE
nan	FALSE	TRUE
to	TRUE	TRUE
nan	FALSE	TRUE
resents	TRUE	FALSE

Table 5: Sample of text prediction and gatekeeper predictions on TSG dataset.