A. Appendix

All extended materials and source code related to this paper are avaliable on https://github.com/cyberyu/ava Our repo is composed of two parts: (1) Extended materials related to the main paper, and (2) Source code scripts. To protect proprietary intellectual property, we cannot share the question dataset and proprietary embeddings. We use an alternative data set from Larson *et al.*, *"An Evaluation Dataset for Intent Classification and Out-of-Scope Prediction"*, EMNLP-IJCNLP 2019, to demonstrate the usage of code.

A.1. Additional Results for the Main Paper

Some extended experimental results about MC dropout and optimization are presented on github.

A.1.1. Histogram of Uncertainties by Dropout Ratios

We compare histograms of standard deviations observed from random samples of predictions. The left side contains histograms generated by 381-class intent models trained for 10 epochs, with dropout ratio varied from 10 percent to 90 percent. The right side shows histograms generated by 381 class models trained for 30 epochs.

A.1.2. Uncertainty comparison between 381-class vs 5-class

To understand how uncertainties change vs. the number of classes in BERT, we train another intent classifier using only Tier 1 labels. We compare uncertainty and accuracy changes at different dropout rates between the original 381-class problem and the new 5-class problem.

A.1.3. Grid Search For Optimal Threshold on Dropout

Instead of using optimization, we use a grid search to find optimal combinations of average probability threshold and standard deviation threshold. The search space is set as a 100 x 100 grid on space [0,0] to [1,1], where thresholds vary by step of 0.01 from 0 to 1. Applying thresholds to outputs of BERT predictions give us classifications of relevance vs. irrelevance questions, and using the same combination of test and escalation questions we visualize the F1 score in contour map shown on github repo.

A.1.4. Optimal Threshold Learning on Dropout 381 classes vs 5 classes

Using the same optimization process mentioned in equation (2) of the main paper, we compare the optimal results (also CPU timing) learned from 381 classes vs. 5 classes.

A.1.5. Simple algorithm for sentence completion model

When multiple OOVs occur in a sentence, in order to avoid the computational burden using large beamsize to find the optimal joint probabilities, we assume all candidate words for OOVs are independent, and apply Algorithm 2 one by one to correct the OOVs.

Algorithm 2 Auto-correction of Single OOV token. Using MaskedLM function from Transformer API, each masked position has recommendations of top M candidate words W using their corresponding probabilities P. Then, from these candidates, we evaluate their Levenshtein distances to the OOV token, and select the token with smallest distance as auto-corrected token.

```
1: procedure AUTO-CORRECT($< t_0, .., t_N >, i$)    ▷ $i$ is OOV index
2:     $oov \leftarrow t_i$                         ▷ set OOV token as MASK
3:     $t_i \leftarrow$ [MASK]                      ▷ set OOV token as MASK
4:     $W, P \leftarrow$ MaskedLM($t_0, .., t_N, M$) ▷ ranked top M candidates
5:     for $w_j$ in $W$ do                          ▷ iterate top M candidates
6:         $d_{ij}$ = WordEditDistance($w_j, oov$)
7:     end for
8:     $\hat{j} = \arg\min_j d_{ij}$ ▷ The candidate with shortest edit distance
9:     $t_i = w_{\hat{j}}$                          ▷ Autocorrect the oov token
10:    return $< t_0, .., t_N >$ ▷ Return the autocompleted sentence
11: end procedure
```

A.2. Intent Classification Source Code

A.2.1. BERT embeddings Model Pretraining

The jupyter notebook for pretraining embeddings is at https://github.com/cyberyu/ava/blob/master/scripts/notebooks/BERT_PRETRAIN_Ava.ipynb. Our script is adapted from Denis Antyukhov's blog "Pre-training BERT from scratch with cloud TPU". We set the VOC_SIZE to 32000, and use SentencePiece tokenizer as approximation of Google's WordPiece. The learning rate is set to 2e-5, training batch size is 16, training setps set to 1 million, MAX_SEQ_LENGTH set to 128, and MASKED_LM_PROB is set to 0.15.

To ensure the embeddings is training at the right architecture, please make sure the bert_config.json file referred in the script has the right numbers of hidden and attention layers.

A.2.2. BERT model training and exporting

The jupyter notebook for BERT intent classification model training, validation, prediciton and exporting is at https://github.com/cyberyu/ava/blob/master/scripts/notebooks/BERT_run_classifier_Ava.ipynb. The main script run_classifier_inmem.py is tweaked from the default BERT script run_classifier.py, where a new function serving_input_fn(): is added. To export that model in the same command once training is finished, the '–do_export=true' need be set True, and the trained model will be exported to directory specified in '–export_dir' FLAG.

A.2.3. Model Serving API Script

We create a jupyter notebook to demonstrate how exported model can be served as in-memory classifier for intent classification, located at https://github.com/cyberyu/ava/scripts/notebooks/inmemory_intent.ipynb. The script will load the entire BERT graph in memory from exported directory, keep them in memory and provide inference results on new questions. Please notice that in "getSess()" function, users need to specify the correct exported directory, and the correct embeddings vocabulary path.

A.2.4. Model inference with Dropout Sampling

We provide a script that performs Monte Carlo dropout inference using in-memory classifier. The script assumes three groups of questions are saved in three separate files: training.csv, test.csv, escalation.csv. Users need to specify the number of random samples, and prediction probabilities results are saved as corresponding pickle files. The script is available at https://github.com/cyberyu/ava/scripts/dropout_script.py

A.2.5. Visualization of Model Accuracy and Uncertainty

The visualization notebook https://github.com/cyberyu/ava/scripts/notebooks/BERT_dropout_visualization.ipynb uses output pickle files from the previous script to generate histogram distribution figures and figures 4(b) and (c).

A.3. Threshold Optimization Source Code

A.3.1. Threshold for entropy

Optimization script finding best threshold for entropy is available at https://github.com/cyberyu/ava/blob/master/scripts/optimization/optimize_entropy_threshold.py. The script requires Python 3.6 and Gurobi 8.1.

A.3.2. Threshold for mean probability and standard deviation

Optimization script finding best mean probability threshold and standard deviation threshold is available at https://github.com/cyberyu/ava/blob/master/scripts/optimization/optimize_dropout_thresholds.py

A.4. Sentence Completion Source Code

The complete Sentence Completion RESTFUL API code is in https://github.com/cyberyu/ava/scripts/sentence_completion/serve.py. The model depends on BertForMaskedLM function from Transformer package (ver 2.1.1) to generate token probabilities. We use transformers-cli (https://huggingface.co/transformers/converting_tensorflow_models.html) to convert our early pretrained embeddings to PyTorch formats. The input parameters for API are:

- Input sentence. The usage can be three cases:
  - The input sentence can be noisy (containing misspelled words) that require auto-correction. As shown in the example, the input sentence has some misspelled words.
  - Alternatively, it can also be a masked sentence, in the form of "Does it require [MASK] signature for IRA signup". [MASK] indicates the word needs to be predicted. In this case, the predicted words will not be matched back to input words. Every MASKED word will have a separate output of top M predict words. But the main output of the completed sentence is still one (because it can be combined with misspelled words and cause a large search) .
  - Alternatively, the sentence can be a complete sentence, which only needs to be evaluated only for Perplexity score. Notice the score is for the entire sentence. The lower the score, the more usual the sentence is.
- Beamsize: This determines how many alternative choices the model needs to explore to complete the sentence. We have three versions of functions, predict_oov_v1, predict_oov_v2 and predict_oov_v3. When there are multiple [MASK] signs in a sentence, and beamsize is larger than 100, v3 function is used as independent correction of multiple OOVs. If beamsize is smaller than 100, v2 is used as joint-probability based correction. If a sentence has only one [MASK] sign, v1 (Algorithm 2 in Appendix) is used.
- Customized Vocabulary: The default vocabulary is the encoding vocabulary when the bidirectional language model was trained. Any words in the sentence that do not occur in vocabulary will be treated as OOV, and will be predicted and matched. If you want to avoid predicting unwanted words, you can include them in the customized vocabulary. For multiple words, combine them with "—" and the algorithm will split them into list. It is possible to turn off this customized vo-

cabulary during runtime, which simply just put None in the parameters.

- Ignore rule: Sometimes we expect the model to ignore a range of words belonging to specific patterns, for example, all words that are capitalized, all words that start with numbers. They can be specified as ignore rules using regular expressions to skip processing them as OOV words. For example, expression ”[A-Z]+” tells the model to ignore all uppercase words, so it will not treat 'IRA' as an OOV even it is not in the embeddings vocabulary (because the embeddings are lowercased). To turn this function off, use None as the parameter.

The model returns two values: the completed sentence, and its perplexity score.

A.5. RASA Server Source Code

The proposed chatbot utilizes RASA's open framework to integrate RASA's "chitchat" capability with our proposed customized task-oriented models. To achieve this, we set up an additional action endpoint server to handle dialogues that trigger customized actions (sentence completion+intent classification), which is specified in actions.py file. Dialogue management is handled by RASA's Core dialogue management models, where training data is specified in stories.md file. So, in RASA dialogue_model.py file run_core function, the agent loads two components: nlu_interpreter and action_endpoint.

The entire RASA project for chatbot is shared under `https://github.com/cyberyu/ava/intent_bot`. Please follow the github guidance in README file to setup the backend process.

A.6. Microsoft Teams Setup

Our chatbot uses Microsoft Teams as front-end to connect to RASA backend. We realize setting up MS Teams smoothly is a non-trivial task, especially in enterprise controlled enviornment. So we shared detailed steps on Github repo.

A.7. Connect MS Teams to RASA

At RASA side, the main tweak to allow MS Team connection is at dialogue_model.py file. The BotFrameworkInput library needs to be imported, and the correct app_id and app_password specified in MS Teams setup should be assigned to initialize RASA InputChannel.