Appendix

A Model Details

A.1 Data Processing

For tabular features/models, we can simply rely on the same preprocessing as AutoGluon-Tabular, which has been found to also work well for other learning algorithms [15]. For our subsequently introduced multimodal neural networks that operate on both text and tabular features, we simply rescale and center numeric features and impute their missing values via their average. Missing values for categorical features (and previously unseen categories encountered during inference) are represented via an additional Unknown category in order to avoid unrealistic missing at random assumptions. Missing text fields are handled as empty strings in our preprocessing pipeline.

AutoGluon also automatically infers the type of each feature via simple yet effective heuristics. One decision particular to our multimodal applications is when to designate a column of string values as a categorical vs. text feature. In this work, we simply threshold based on the number of unique values in the column, such that commonly reoccurring strings are treated as discrete categories rather than unstructured text. We choose the threshold to be 20 in all presented experiments, based on visually confirming the inferred feature types with this threshold agree with our beliefs regarding which columns should be handled as text.

A.2 Handling Text Fields in the Transformer

Given multiple text columns, we feed the tokenized text from all columns jointly into our Transformer, as illustrated in Figure S1. We follow the usual method to format text from multiple passages [13]: tokenized inputs from different text fields are merged with special [SEP] delimiter tokens between fields and a [CLS] prefix token is subsequently appended at the start of merged input. To further ensure that the network distinguishes boundaries between adjacent text fields, we alternate 0s and 1s as the segment IDs. Here segment IDs and the [SEP] token were previously used to demarcate boundaries between passages during pre-training [13]. After feeding the merged inputs into the Transformer, we can extract its intermediate representations at each position as token-level embeddings (each token has one embedding, which has been contextualized based on information from the other tokens).

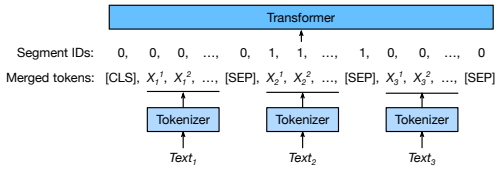

Figure S1: Inputting data from 3 text fields into Transformer.

When the total length of tokenized text fields exceed the maximum allowed length (set to be 512 throughout this work), we truncate the input by repeatedly removing one token from the longest individual text field until the length constraint is met. Since self-attention is permutation equivariant, a common practice is to assign an additional vector that encodes each position (namely positional encoding) so that the Transformer can distinguish between identical tokens occurring at different locations [62]. After merging multiple text fields into a single input, we simply assign positional encodings based on this larger input.

A.3 Network Architectures

In this paper, we used a single-hidden-layer MLP as the basic building block for encoding features and projecting the hidden states. It has one bottleneck layer and uses layer normalization. We use the leaky ReLU activation (with slope set to 0.1) for all basic MLP layers mentioned throughout the paper. For the 6-layer Transformer model in *Fuse-Early*, we used the GeLU activation like Devlin et al. [13]. We set the number of units, heads, and hidden size of FFN (the feedforward layers) in this Transformer to be 64, 4, 256 correspondingly. For the categorical features, we use an encoding network that is similar to the factorized embedding in ALBERT [42], in which we use an embedding

layer with 32 units and then project it with a basic MLP layer that has 64 bottleneck units. We further set the number of output units in the basic MLP to be the same as the token-embeddings used in the pretrained Transformer model (i.e., ELECTRA or RoBERTa) so that all vectors belong to the same space. In the *Fuse-Late* variant, we further concatenate all encoded categorical features and encode them with a second basic MLP layer. Numeric features are concatenated and encoded with one basic MLP layer. These MLP layers all utilize 128 bottleneck units and their output unit number matches the dimensionality of token embeddings for the pretrained Transformer.

A.4 Neural Network Optimization

All text/multimodal neural networks are trained with the slanted triangular learning rate scheduler [76] with initial learning rate set to 0.0, the maximal learning rate set to 5×10^{-5} and warmup set to 0.1. We use a batch size of 128, 10^{-4} weight decay, and the AdamW optimizer. Text/multimodal networks are trained for 10 epochs and we early stop based on their validation performance. These learning rate and weight decay values were determined via grid search on a single smaller (subsampled) dataset that we used for early initial experiments.

A.5 Details of AutoGluon Tabular Models in the Stack Ensemble

For better efficiency, we considered just the following tabular models when running AutoGluon [15]:

- Fully-connected Neural Network (MLP) with ReLU activations [15].
- LightGBM model with default hyperparameters (GBM) [38].
- A second LightGBM model with a different set of hyperparameter values. By default, AutoGluon uses this second model in conjunction with the first LightGBM model.
- An implementation of Extremely Randomized Trees from the LightGBM library [24].
- CatBoost gradient boosted trees for sophisticated handling of categorical features [49].

To avoid overfitting in stacking, all models are trained with 5 fold cross-validation (bagging) as described by Erickson et al. [15]. For classification tasks, the outputs of each base model which are aggregated in the ensemble are taken to be predicted class probabilities.

A.6 Notes on Hyperparameter Tuning

Note that hyperparameter tuning was not a major focus in this paper. Standard hyperparameter tuning strategies [80] are readily applicable to our multimodal setting, and the experiments presented here could easily employ the advanced Bayesian optimization techniques available in AutoGluon [81]. We expect the performance of all of our proposed AutoML strategies will grow even better with time devoted to hyperparameter tuning. However in this paper we did not conduct such a search and simply used the default hyperparameters supplied by AutoGluon for tabular models, which are already highly performant [15], and the text/multimodal network hyperparameters are listed here and are viewable in our released code. Over just a few datasets, we found that relative performance of different strategies did not qualitatively differ with other reasonable manually-chosen hyperparameter settings (i.e. hyperparameter values known to generally work well for these specific models such as alternative popular learning rate schedules or small changes to the size of the networks).

Rather than only reporting a couple thoroughly-tuned results, we instead preferred to spend our time/compute budget to explore more AutoML strategies over more datasets. Note that all H2O AutoML variants reported in Table 3 relied on extensive hyperparameter sweeps (automatically used within H2O), and yet were still unable to outperform our untuned methods. This further supports the claim that we have identified a broadly performant strategy for multimodal AutoML.

A.7 Compute Details

All experiments were run on Amazon Web Services EC2 cloud instances (P3.2xlarge). Each instance has two NVIDIA V100 Tensor Core GPUs. About 2000 hours of total compute was required for all experiments presented in this paper (15 instances used for about a week). Given a limited compute budget, we believe more meaningful conclusions may be drawn by running more algorithms over

more datasets rather than replicate runs of different seeds/splits on just a few (less diverse) datasets. We also omitted small datasets from our benchmark for which replicate runs would otherwise be required to get stable results.

B Dataset and Benchmark Descriptions

Note that a more detailed description of each dataset (and link to original data source) is provided in the benchmark GitHub repository, which can be previewed in `https://github.com/submission001/anonymoussubmission_automl` and we will update the link in our final version. The benchmark repository contains: (i) methods to easily retrieve the individual datasets and train/test splits, in which we randomly split 20% of the dataset as the test data by default, (ii) code to run all of the ML strategies studied in this paper and reproduce our results, and (iii) the scripts we used to produce each benchmark dataset from the original data source. Common modifications made to original data sources to produce the benchmark dataset versions included: defining a practically meaningful prediction task if there was not one in the original dataset, omitting duplicated rows, omitting non-predictive features (e.g. user ID) and those that were too correlated with the prediction target (making the benchmark too easy otherwise), and down-sampling overly large datasets (mercari, jigsaw) to ensure the benchmark remains computationally accessible. All results can be easily reproduced – all datasets, code, and evaluation procedures are accessible and documented in the repository. The code for our best multimodal AutoML strategies has been contributed into the open-source AutoGluon library, and a user-friendly tutorial may be found here: `https://auto.gluon.ai/stable/tutorials/tabular_prediction/tabular-multimodal-text-others.html`.

prod: Classify the sentiment of user reviews of products based on the review text and product type.

airbnb: Predict the price label of AirBnb listings (in Melbourne, Australia) based on the page of each listing which includes many miscellaneous features about the listing.

channel: Predict which news category (i.e. channel) a Mashable.com news article belongs to based on the text of its title, as well as auxiliary numerical features like the number of words in the article, its average token length, how many keywords are listed, etc.

wine: Classify the variety of wines based on tasting descriptions from sommeliers, their price, country-of-origin, and other features.

imdb: Predict whether or not a movie falls within the Drama category based on its name, description, actors/directors, year released, runtime, and other features.*

jigsaw: Predict whether online social media comments are toxic based on their text and additional tabular features providing information about the post (e.g. likes, rating, date created, etc.). This data originates from a Kaggle competition in which the 1st place solution[8] utilized dataset-specific tricks such as a Bucket Sequencing Collator, auxiliary domain-specific prediction tasks for models, and a custom mimic loss function for training. The 2nd place solution[9] also used a custom loss function, and the 3rd place solution[10] used custom target weighting, unlike our proposed AutoML solution which does utilize dataset-specific tricks.

fake: Predict whether online job postings are real or fake based on their text, amount of salary offered, degree of education demanded, etc.

kick: Predict whether a proposed Kickstarter project will achieve funding goal based on its title, description, amount of money requested, date posted, and other features.

*PromptCloud released the original version of the data from which the version of this dataset in our benchmark was created.

[8] `https://www.kaggle.com/c/jigsaw-unintended-bias-in-toxicity-classification/discussion/103280`

[9] `https://www.kaggle.com/c/jigsaw-unintended-bias-in-toxicity-classification/discussion/100661`

[10] `https://www.kaggle.com/c/jigsaw-unintended-bias-in-toxicity-classification/discussion/97471`

ae: Predict the price of inner-wear items sold by retailer American Eagle based on features from their online product page.*

qaa: Given a question and an answer (from the Crowdsource team at Google) as well as additional category features, predict the (subjective) type of the answer in relation to the question.

qaq: Given a question and an answer (from the Crowdsource team at Google) as well as additional category features, predict the (subjective) type of the question in relation to the answer.

cloth: Predict the score of a customer review of clothing items (sold by an anonymous retailer) based on the review text, and product features like the clothing category.

mercari: Predict the price of items sold in the online marketplace of Mercari based on miscellaneous information from the product page like name, description, free shipping, etc. We transformed the price to log-scale when creating the benchmark version of this dataset. This data originates from a Kaggle competition, in which the 1st place[11] and 3rd place[12] teams engineered dataset-specific text features such as customized bag-of-words[13] and character N-grams from names, carefully-tuned learning-rate schedules (the 1st place solution doubles the batch-size for each epoch), and model ensembles that appear specifically crafted for just this dataset. Instead, the AutoML solution described in our paper requires very little human-engineering and generalizes to different types of multimodal text+tabular datasets.

jc: Predict the sale price of items sold on the website of the retailer JC Penney based on miscellaneous information on the product page like its title, description, rating, etc.*

pop: Predict the popularity (number of shares on social media, on log-scale) of Mashable.com news articles based on the text of their title, as well as auxiliary numerical features like the number of words in the article, its average token length, and how many keywords are listed, etc. While this dataset stems from the same original data source as **channel**, the two have different labels and do not share the same set of features (to ensure the prediction problems are neither too easy/hard).

Each dataset can be easily loaded into Python (or another programming language) via standard table/dataframe libraries like `pandas`. We release our modified versions of the datasets in our benchmark under a **CC BY-NC-SA** license, and note that any data from this benchmark which has previously been published elsewhere falls under the original license from which the data originated (links to the original sources are provided in our repository). We the authors bear all responsibility in case of violation of rights. Long-term preservation of our benchmark methodology will be ensured as the repository is hosted on GitHub, such that users can contribute their own improvements or publicly raise issues for us to address. The data files are currently hosted in AWS Simple Cloud Storage (S3), which is a reliable and highly-available medium. Upon acceptance, these files will be uploaded as an OpenML Study similar to the AutoML Benchmark[14] [25]: `https://www.openml.org/s/271/data` (which will ensure they remain permanently available). Ongoing maintanence will be provided by the authors and hopefully an organically-grown GitHub community interested in multimodal text/tabular AutoML. A datasheet [74] for our overall multimodal text/tabular benchmark is provided below.

B.1 Datasheet for our 15 dataset multimodal text/tabular benchmark

To avoid redundancy, we only provide details here not covered elsewhere in the paper or our benchmark repository. Table 2 lists statistics of each dataset. For details on how each dataset was collected, please refer to the original source linked in our benchmark repository.

For what purpose were the benchmark datasets created? We created the datasets in this benchmark to evaluate supervised machine learning (classification/regression) algorithms designed to jointly operate on text and tabular features. The original versions of these data were also initially

[11] `https://www.kaggle.com/c/mercari-price-suggestion-challenge/discussion/50256`

[12] `https://www.kaggle.com/c/mercari-price-suggestion-challenge/discussion/50272`

[13] `https://www.kaggle.com/whitebird/mercari-price-3rd-0-3905-cv-at-pb-in-3300-s#L362-L365`

[14] `https://github.com/openml/automlbenchmark`

aggregated primarily for a similar purpose.

Who created this benchmark? Who funded its creation? The authors of this paper, all scientists employed by Amazon, curated this benchmark. Curating the benchmark did not cost significant money, and the benchmark data are currently hosted on cloud servers (S3) provided by Amazon. The original data sources were created/curated/funded by various companies/individuals, please refer to each individual source for more details.

Do the datasets contain all possible instances or are they a sample (not necessarily random) of instances from a larger set? Each dataset is a sample of instances from a larger set. We caution these samples may not be at all representative of the larger set, and thus the benchmark should not be used to draw domain-specific conclusions/insights through scientific data analysis of individual datasets.

Is any information missing from individual instances? Yes there are many missing fields in certain datasets. It is unclear why they are missing or if the missingness mechanism satisfies the missing at random assumption.

Are relationships between individual instances made explicit? For evaluating ML performance, we simply assume the data are IID. However this may be violated by certain datasets. For example, product datasets may contain near duplicate products and products may be related (reviewed by the same users, price of a product can affect price of others, etc.). We do not explicitly know the relationships between instances in these data.

Are there recommended data splits (e.g., training, development/validation, testing)? Yes the benchmark provides a recommended training/test split, but ML systems are free to split validation data from the training set as they see fit. The split was done randomly (stratified based on labels for classification) to best reflect an IID setting for which supervised learning methods are primarily intended.

Does the benchmark contain data that might be considered confidential? Not to our knowledge, but it is possible that a person entered confidential information into the text fields (although they knew these would be publicized).

Does the benchmark contain data that, if viewed directly, might be offensive, insulting, threatening, or might otherwise cause anxiety? The data are mostly non-offensive data used for business purposes. Exceptions are the text fields in the *jigsaw* dataset, which contain toxic online comments, and the *channel/pop* datasets, which contain news article titles that may be anxiety-inducing. Furthermore, some of the user reviews of products may be offensive to certain people, although we did not spot any.

Does the benchmark relate to people? Yes some datasets contain information from people. These all stem from commercial sources where people upload their data intentionally to share it with the world (e.g. user reviews, Kickstarter fundraising, public questions, etc.). There is no sensitive/personal information in these data, beyond what a person intended to publicize.

Is it possible to identify individuals, either directly or indirectly from the benchmark? Yes it may be possible as some datasets contain text fields where an individual may have entered arbitrary information (although they knew the information would appear publicly).

Does the benchmark contain data that might be considered sensitive in any way? Not to our knowledge given all this data was already publicly available, but it is possible given the nature of free form text fields.

How did you process the data from the original sources? Is the software used to preprocess/clean the datasets available? We processed each dataset from the original source using the

publicly available scripts in the **scripts/data_processing/** folder of our benchmark GitHub repository. To create versions for our benchmark, we omitted certain features (columns), badly formatted or duplicated rows and subsampled overly large datasets.

Have the benchmark data been used for any tasks already? Yes many of the datasets have been used to evaluate ML systems, some through formal prediction competitions. Other datasets have been used to demonstrate data analysis techniques. For the datasets originally stemming from Kaggle, one can find some of the previously considered tasks in the discussion forum or notebooks associated with the original dataset.

What (other) tasks could the benchmark data be used for? Are there tasks for which these data should not be used? We recommend these datasets only be used for evaluation of machine learning algorithms. One could select different target variables in each dataset to create new prediction tasks to evaluate, but these will likely be less practically meaningful (i.e. representative of a real application) than the target variable we have selected for each dataset. Also note that none of the datasets has extremely large sample-size (say over a million), so modeling conclusions drawn based on this benchmark may not translate to applications with massive datasets.

Will the benchmark be distributed to third parties outside of the entity on behalf of which the dataset was created? Yes the benchmark is made publicly available.

Have any third parties imposed IP-based or other restrictions on the data? Yes please refer to the licenses corresponding to each original data source (linked from our repository) for more details.

Do any export controls or other regulatory restrictions apply to the dataset or to individual instances? Not to our knowledge.

How can the curators of the benchmark be contacted? You can open a GitHub issue at the benchmark repository, or email the authors of this paper.

Will the benchmark be updated (e.g. to correct errors, add new datasets, add/delete instances)? Yes updates will be done via GitHub and publicly announced there.

If others want to extend/augment/build on/contribute to the dataset, is there a mechanism for them to do so? Yes anybody may open Pull Request with desired changes on GitHub.

Additional References for the Appendix

[13] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *NAACL HLT*, 2019.

[15] N. Erickson, J. Mueller, A. Shirkov, H. Zhang, P. Larroy, M. Li, and A. Smola. Autogluon-tabular: Robust and accurate automl for structured data. *arXiv preprint arXiv:2003.06505*, 2020.

[74] T. Gebru, J. Morgenstern, B. Vecchione, J. W. Vaughan, H. Wallach, H. Daumé III, and K. Crawford. Datasheets for datasets. *arXiv preprint arXiv:1803.09010*, 2018.

[24] P. Geurts, D. Ernst, and L. Wehenkel. Extremely randomized trees. *Machine learning*, 63(1): 3–42, 2006.

[76] J. Howard and S. Ruder. Universal language model fine-tuning for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, 2018.

[38] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu. LightGBM: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems*, 2017.

[42] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut. ALBERT: A lite bert for self-supervised learning of language representations. In *International Conference on Learning Representations*, 2020.

[49] L. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin. CatBoost: unbiased boosting with categorical features. In *Advances in Neural Information Processing Systems*, 2018.

[80] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. De Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2015.

[81] L. C. Tiao, A. Klein, C. Archambeau, and M. Seeger. Model-based asynchronous hyperparameter optimization. *arXiv preprint arXiv:2003.10865*, 2020.

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