A. Task Details

A.1. GLUE

GLUE consists of 9 tasks. Two are single-sentence classification; *CoLA* (Corpus of Linguistic Acceptability; Warstadt et al., 2019) involves determining if a sentence is linguistically acceptable or not, while *SST-2* (Stanford Sentiment Treebank 2; Socher et al., 2013) involves predicting if a sentence has positive or negative sentiment. Three tasks involve determining if two sentences are similar or paraphrases of each other: *MRPC* (Microsoft Research Paragraph Corpus; Dolan & Brockett, 2005), *QQP* (Quora Question Pairs)[6], and *STS-B* (Semantic Textual Similarity Benchmark; Cer et al., 2017). The rest are NLI tasks: *QNLI* (Question NLI, derived from SQuAD; Rajpurkar et al., 2016), *RTE* (Recognizing Textual Entailment; Bentivogli et al., 2009), *WNLI* (Winograd NLI; Levesque et al., 2012), and *MNLI* (Multi-genre NLI; Williams et al., 2018).

B. Model Training Details

B.1. Distilled Language Model Decompositions

B.1.1. Language Model Decompositions

Large language models are highly effective at text generation (Brown et al., 2020) but have not yet been explored in the context of question decomposition. In particular, one obstacle is the sheer computational and monetary cost associated with such models. We thus use a language model to generate question decompositions while conditioning on a few labeled question-decomposition pairs, and then we train a smaller, sequence-to-sequence model on the generated question-decomposition pairs, which we use to efficiently decompose many questions. Our approach, which we call Distilled Language Model (DLM), leverages the large language model to produce pseudo- training data for a more efficient model.

As our language model, we use the 175B parameter, pretrained GPT-3 model (Brown et al., 2020) via the OpenAI API.[7] We label the maximum number of question-decomposition pairs that fit in the context window of GPT-3 (2048 tokens or 46 question-decompositions). For labeling, we sample questions randomly from HotpotQA's training set. To condition the language model, we format question-decomposition pairs as "[Question] = [Decomposition]", where the decomposition consists of several consecutive subquestions. We concatenate the pairs, each on a new line, with a new question on the final line to form a prompt. We then generate from the LM, conditioned on the prompt. For decoding, we found that GPT-3 copies the question as the decomposition with greedy decoding. Therefore, we use a sample-and-rank decoding strategy, to choose the best decoding out of several possible candidates. We sample 16 decompositions with top-p sampling (Holtzman et al., 2020) with $p = 0.95$, rank decompositions from highest to lowest based their average token-level log probability, and choose the highest-ranked decomposition which satisfies the basic sanity checks for decomposition from Perez et al. (2020). The sanity checks avoid the question-copying failure mode by checking if a decomposition has (1) more than one subquestion (question mark), (2) no subquestion which contains all words in the multi-hop question, and (3) no subquestion longer than the multi-hop question. We generate decompositions for HotpotQA dev questions, which we estimate costs $0.15 per example or $1.1k for the 7405 dev examples via the OpenAI API. Decomposing all 90447 training examples would roughly cost an extra $13.3k, motivating distillation.

B.1.2. Distilling Decompositions

As our distilled, sequence-to-sequence model, we use the 3B parameter, pretrained T5 model (Raffel et al., 2020) via HuggingFace Transformers (Wolf et al., 2020). We finetune T5 on our question-decomposition examples and then use it to generate subquestions for all training questions.

To finetune T5, we split our question-decomposition examples into train (80%), dev (10%), and test (10%) splits. We finetune T5 with a learning rate of $1e-4$, and we sweep over label smoothing $\in \{0.1, 0.2, 0.4, 0.6\}$, number of training epochs $\in \{3, 5, 10\}$, and batch size in $\in \{16, 32, 64\}$, choosing the best hyperparmeters ($0.1, 3, 64$, respectively) based on dev BLEU (Papineni et al., 2002). We stop training early when dev BLEU does not increase after one training epoch. We generate decompositions using beam search of size 4 and length penalty of 0.6 as in Raffel et al. (2020), achieving a test BLEU of 50.7. We then finetune a new T5 model using the best hyperparameters on all question-decomposition examples except for a small set of 200 examples used for early stopping.

B.2. Longformer

Similar to Beltagy et al. (2020), we train Longformer models for up to 6 epochs, stopping training early if dev loss doesn't decrease after one epoch. We sweep over learning rate $\in \{3 \times 10^{-5}, 5 \times 10^{-5}, 1 \times 10^{-4}\}$.

B.3. Regression

STS-B is a regression task in GLUE where labels are continuous values in $[0, 5]$. Here, we learn to minimize mean-squared error, which is equivalent to minimizing

[6] data.quora.com/First-Quora-Dataset-Release-Question-Pairs
[7] https://beta.openai.com/

Hyperparam	LONGFORMER	ROBERTA	BART	ALBERT	GPT2
Learning Rate	{3e-5, 5e-5, 1e-4}	{1e-5, 2e-5, 3e-5}	{5e-6, 1e-5, 2e-5}	{2e-5, 3e-5, 5e-5}	{6.25e-5, 3.125e-5, 1.25e-4}
Batch Size	32	{16, 32}	{32, 128}	{32, 128}	32
Max Epochs	6	10	10	3	3
Weight Decay	0.01	0.1	0.01	0.01	0.01
Warmup Ratio	0.06	0.06	0.06	0.1	0.002
Adam β_2	0.999	0.98	0.98	0.999	0.999
Adam ϵ	1e-6	1e-6	1e-8	1e-6	1e-8
Grad. Clip Norm	∞	∞	∞	1	1

Table 1. Training hyperparameters for all transformer models, based on those from each model's original paper. Column names refer to model types, including models of different sizes or trained from scratch with the same architecture.

log-likelihood and thus codelength.[8] We treat each scalar prediction as the mean of a Gaussian distribution and tune a single standard deviation parameter shared across all predictions from a single model. We choose the variance based on dev log-likelihood using grid search over $[10^{-2.5}, 10^{1.5}]$ with 1000 log-uniformly spaced samples. To send the first block of labels, Alice and Bob use a uniform distribution over the interval $[0, 5]$.

The FastText library only supports classification, so we turn STS-B into a 26-way classification task by rounding label values to the nearest 0.2, following T5 (Raffel et al., 2020). We compute a real-valued, mean prediction by evaluating the average class label value when marginalizing over class probabilities. We then tune variance on dev as usual.

B.4. Ensemble Model

B.4.1. FASTTEXT

For the FastText classifier, we initialize with the 2M pretrained, 300-dimensional word vectors trained on Common Crawl (600B tokens).[9] We tune hyperparameters using the official implementation of automatic hyperparameter tuning, which we run for 2 hours, which is generally sufficient for 20+ hyperparmeter trials and convergence on dev accuracy. The tuning implementation chooses the hyperparameters based on dev accuracy instead of loss as we typically do, but our procedure of tuning a softmax temperature parameter helps FastText reach significantly below-baseline loss.

B.4.2. TRANSFORMER MODELS

The remaining models in our ensemble are transformer-based models trained with HuggingFace Transformers (Wolf et al., 2020). Table 1 shows the hyperparameter ranges used for each model, which we chose based on those used in each model's original paper for GLUE. For TRANSFORMER models trained from scratch on e-SNLI and GLUE, we use the ROBERTA$_{\text{BASE}}$ and ROBERTA$_{\text{LARGE}}$ architecture with the ROBERTA hyperparameters, except that we use a larger batch size ($\in \{64, 128\}$) for the LARGE transformer, which gave better results.

B.5. Training Hardware

We train FastText models on 1 CPU core (40GB of memory) and FiLM models on GeForce GTX 1080ti 11GB GPUs (10GB CPU memory). We train other models with "almost floating point 16" mixed precision arithmetic (Micikevicius et al., 2018) on 1 RTX-8000 48GB GPU (CPU memory of 100GB for HOTPOTQA and 30GB for e-SNLI and GLUE).

C. Additional Experiments

C.1. Is It Helpful to Answer Subquestions?

In §4.1, we found that decompositions increase in usefulness as the original, no-decomposition model's loss decreases, up until some point after which decompositions decrease in usefulness. To examine if the same trend holds for other models, we show the same plot for TRANSFORMER$_{\text{BASE}}$ in Fig. 9. Here, decompositions increase in usefulness as the codelength (loss) of the original model decreases. However, for most decomposition methods, we do not find a point at which decompositions begin to decrease in usefulness, which fits our hypothesis that, for decompositions to become less useful, the model must have enough examples to learn task-relevant capabilities directly from the data. However, the slope of improvement decreases (i.e., the second derivative is negative), suggesting that, given more training data, decompositions will also peak in usefulness for TRANSFORMER$_{\text{BASE}}$.

[8] Cover & Thomas (2006) justifies the relationship between log-likelihood and codelength for continuous random variables.

[9] https://fasttext.cc/docs/en/english-vectors.html

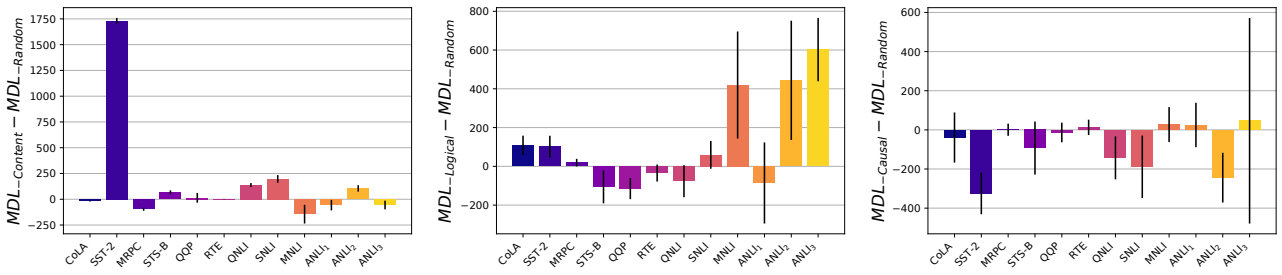

Figure 8. Difference between MDL (in bits) when we mask input words that are (1) of a given type and (2) randomly chosen with the same frequency as (1). Mean and std. err. over 5 random seeds for content words (**left**), logical words (**middle**), and causal words (**right**).

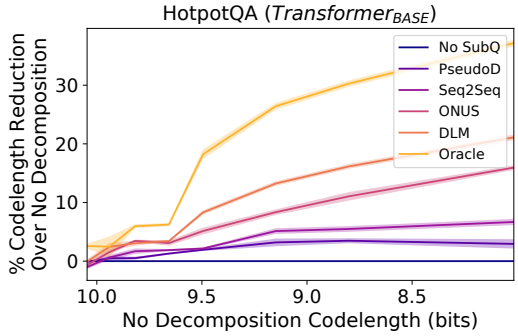

Figure 9. The reduction in codelength over the no-decomposition baseline from using subanswers from various decomposition methods (mean and std. error for TRANSFORMER$_{\text{BASE}}$).

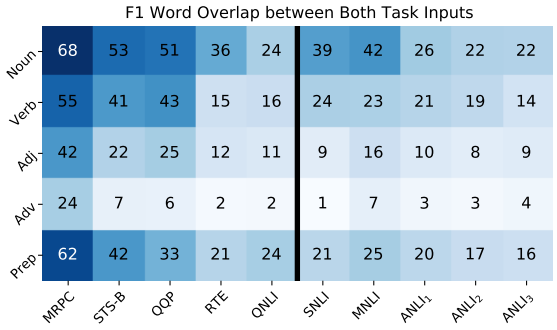

Figure 10. The average F1 word overlap between the two inputs for examples in different GLUE and NLI datasets with two inputs. For NLI, the overlap between premises and hypotheses is larger for nouns and verbs compared to adjectives and adverbs, a possible reason for why the latter are more important.

C.2. Examining Text Datasets

Why are certain POS more useful for NLI? In §4.3.1, we found that adjectives and adverbs were generally more important that nouns and verbs for NLI. One possible explanation of this finding is that annotators often form hypotheses by modifying premises via adding adjectives or adverbs instead of changing verbs and nouns. To test this hypothesis, we examine the amount of overlap between the premise and hypothesis for words of different POS tags. In particular, we measure the F1 overlap between words of a given POS in the premise vs. hypothesis (averaged across all training examples). For reference, we also measure the overlap between the first and second input for other two-input tasks in GLUE. Fig. 10 shows the results.

For NLI tasks, premise-hypothesis word overlap is consistently higher for nouns and verbs than for adjectives and adverbs. For example, MNLI premise-hypothesis pairs have a noun overlap of 43 F1 and verb overlap of 24 F1, compared to an adjective overlap for 15 F1 and adverb overlap of 7 F1. Our finding suggests that nouns and verbs are not as important for NLI tasks due to the fact that annotators often leave nouns and verbs unchanged when writing hypotheses. Encouraging annotators to change nouns and verbs more often may thus increase the importance of nouns and verbs in future NLI tasks.

How useful are content words? Sugawara et al. (2020) hypothesized that "content" words are particularly useful for NLP tasks, taking content words to be nouns, verbs, adjectives, adverbs, or numbers. We test their utility on GLUE, SNLI, and ANLI using RDA, by evaluating MDL$_{-\text{Content}}$ − MDL$_{-\text{Random}}$ (Fig. 8 left). The value is positive for SST-2, STS-B, QNLI, SNLI, and ANLI$_2$ and negative for MRPC, MNLI, ANLI$_1$, and ANLI$_3$. In particular, the value for SST-2 is very high (1732 bits), indicating that content words are important for sentiment classification, likely due to the importance of adjectives as found in §4.3.1. For QNLI, content words are important, despite earlier findings that each individual POS group (nouns, verbs, adjectives, or adverbs) was not important for QNLI (§4.3.1, Fig. 5), indicating that QNLI requires reasoning over multiple POS in tandem.

How useful are "logical" words? Sugawara et al. (2020) hypothesized that words that have to do with the logical meaning of a sentence (e.g., quantifiers and logical

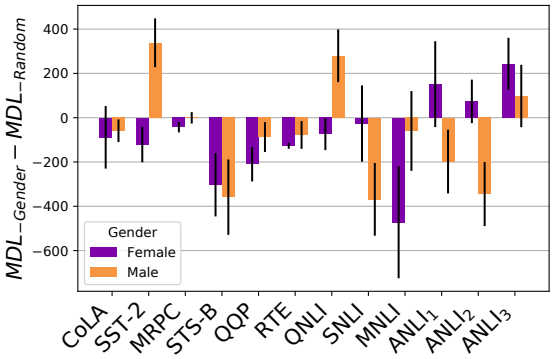

Figure 11. The difference between MDL (in bits) when (1) masculine/feminine words are masked and (2) the same fraction of input words are masked uniformly at random.

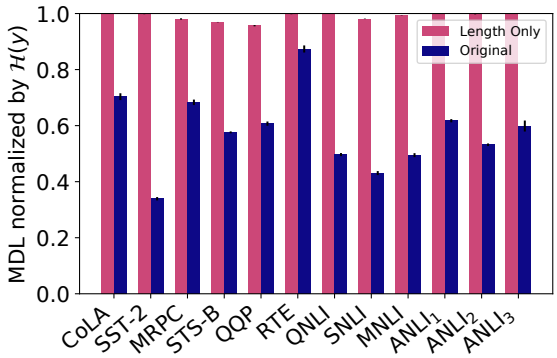

Figure 12. **MDL with length-only input** compared to MDL with original input, normalized by the MDL when encoding labels with $p(y)$ for reference. Length information reduces MDL over $p(y)$ on MRPC, STS-B, QQP, and SNLI, though not significantly.

connectives) are useful for NLP tasks. Using GLUE, SNLI, and ANLI, we test the usefulness of logical words, which we take as: *all, any, each, every, few, if, more, most, no, nor, not, n't, other, same, some,* and *than* (following Sugawara et al., 2020). As shown in Fig. 8 (middle), $\text{MDL}_{-\text{Logical}} - \text{MDL}_{-\text{Random}}$ is positive for CoLA, SST-2, MNLI, ANLI_2, and ANLI_3 and negative for STS-B and QQP. Notably, $\text{MDL}_{-\text{Logical}} - \text{MDL}_{-\text{Random}}$ is large for MNLI, ANLI_2, and ANLI_3, three entailment detection tasks, where we would expect logical words to be important.

How useful are causal words? Another group of words that Sugawara et al. (2020) hypothesized are useful are words that express causal relationships: *as, because, cause, reason, since, therefore,* and *why*. As shown in Fig. 8 (right), $\text{MDL}_{-\text{Causal}} - \text{MDL}_{-\text{Random}}$ to be within std. error of 0 for all tasks except SST-2, QNLI, SNLI, and ANLI_2, where $\text{MDL}_{-\text{Causal}} - \text{MDL}_{-\text{Random}} < 0$. Thus, causal words do not appear particularly useful for GLUE.

Do datasets suffer from gender bias, even when controlling for word frequency? In §4.3.3, we assessed if datasets rely more on male- or female- gendered words by comparing MDL when masculine vs. feminine input words are masked, looking at $\text{MDL}_{-\text{Male}} - \text{MDL}_{-\text{Female}}$. However, we may wish to focus on gender bias present in datasets beyond easy-to-detect differences in male- and female- gendered word frequency. To control for frequency, we evaluate $\text{MDL}_{-\text{Male}} - \text{MDL}_{-\text{Random}}$ and $\text{MDL}_{-\text{Female}} - \text{MDL}_{-\text{Random}}$, as we did for our word type experiments. We show results in Fig. 11. For SST-2 and QNLI, masculine words are more useful than randomly-chosen words, while feminine words are less useful than randomly-chosen words, a sign of male-favored gender bias. Most tasks, however, do not show similar patterns of bias as SST-2 and QNLI do.

How useful is input length? Input text length can be highly predictive of the class label (see, e.g., Dixon et al., 2018). RDA can be used to evaluate text datasets for such length bias. We evaluate MDL when only providing the input length, in terms of number of tokens (counted via spaCy).[10] As shown in Fig. 12, the labels in MRPC, STS-B, QQP, and SNLI can be compressed using the input length, though not to a large extent. Other tasks cannot be compressed using length alone. Our results on SNLI agree with Gururangan et al. (2018) who found that hypotheses were generally shorter for entailment examples and longer for neutral examples. Similarly, they also found that length is less discriminative on MNLI compared to SNLI.

D. Additional Related Work

Recent work has raised increasing awareness of the importance of characterizing the datasets we release, via datasheets (Gebru et al., 2018) or data statements (Bender & Friedman, 2018). A key motivation for datasheets is to inform machine learning practitioners of (1) biases that models may learn when trained on the data or (2) model weaknesses that may not be caught by testing on biased data. As we saw earlier, RDA is a useful tool for uncovering such biases (e.g., gender bias) and thus for writing datasheets.

RDA shares high-level motivation with other data analysis methods that aim to measure intrinsic properties of the data. For example, on NLI, Gururangan et al. (2018) measure the point-wise mutual information (PMI) between the label and occurrence of different keywords to find heuristics for SNLI. Similarly, Rudinger et al. (2017) measure PMI between premise words and hypothesis words to uncover race, age, and gender stereotypes in crowdsourced SNLI hypotheses. On MNLI, McCoy et al. (2019) measure the accuracy

[10] Masking all input tokens gave similar results.

of heuristics such as "Assume that a premise entails all hypotheses constructed from words in the premise." These methods capture the relationship between output labels and an input feature, considered *in isolation*, but some features may only be useful when provided along with other features. In such cases, RDA can still capture the utility of the feature.

Other work aims to analyze properties of individual examples in a dataset. For instance, Koh & Liang (2017) use influence functions to determine which training instances are most responsible for a particular test-time prediction, e.g., for image classification. Brunet et al. (2019) use influence functions to find the training documents most responsible for producing gender-biased word embeddings. Item response theory (Baker & Kim, 2004) has been used to find the most challenging examples for current models (Hopkins & May, 2013; Lalor et al., 2019; Martínez-Plumed et al., 2019). Swayamdipta et al. (2020) look at the variability and confidence of model predictions on individual examples throughout training, to determine which ones are easy, hard, or ambiguous. Instead of examining individual examples, we examine general characteristics of the dataset as a whole.