The Benefits of Bad Advice: Autocontrastive Decoding across Model Layers

Ariel Gera, Roni Friedman, Ofir Arviv, Chulaka Gunasekara,
Benjamin Sznajder, Noam Slonim, Eyal Shnarch

IBM Research
{ariel.gera1, ofir.arviv, chulaka.gunasekara}@ibm.com,
{roni.friedman-melamed, benjams, noams, eyals}@il.ibm.com

Abstract

Applying language models to natural language processing tasks typically relies on the representations in the final model layer, as intermediate hidden layer representations are presumed to be less informative. In this work, we argue that due to the gradual improvement across model layers, additional information can be gleaned from the contrast between higher and lower layers during inference. Specifically, in choosing between the probable next token predictions of a generative model, the predictions of lower layers can be used to highlight which candidates are best avoided. We propose a novel approach that utilizes the contrast between layers to improve text generation outputs, and show that it mitigates degenerative behaviors of the model in open-ended generation, significantly improving the quality of generated texts. Furthermore, our results indicate that contrasting between model layers at inference time can yield substantial benefits to certain aspects of general language model capabilities, more effectively extracting knowledge during inference from a given set of model parameters.

1 Introduction

For a wide range of natural language processing tasks, the standard practice is to rely on deep neural networks with a transformer-based architecture (Vaswani et al., 2017). Such models are composed of multiple transformer layers, where typically the representations of the final layer are used for the downstream task. As shown in prior works, some of the representational knowledge required for performing downstream tasks can already be found within intermediate layers of the model (Geva et al., 2021, 2022); at the same time, relying on the representations of lower model layers does result in decreased performance, specifically for inputs that are more challenging (Schwartz et al., 2020; Xin et al., 2020; Elbayad et al., 2020; Sun et al., 2022; Schuster et al., 2022; Din et al., 2023).

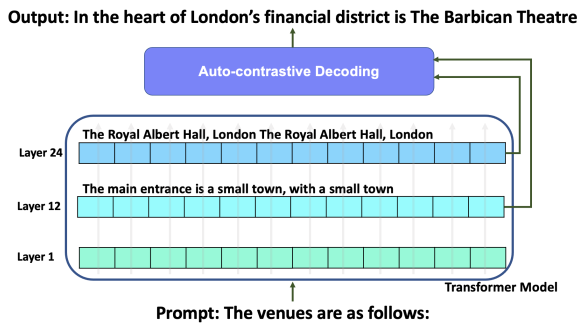

Output: In the heart of London's financial district is The Barbican Theatre
Auto-contrastive Decoding
The Royal Albert Hall, London The Royal Albert Hall, London
Layer 24
The main entrance is a small town, with a small town
Layer 12
Layer 1
Transformer Model
Prompt: The venues are as follows:

Figure 1: **An example of auto-contrastive decoding (*ACD*) with GPT2**, where the top layer (24) is taken as the expert and contrasted with layer 12, the amateur. As decoding is done token by token, we can only see the direct effect on the first token, where *ACD* leads to selecting an alternative high probability token - "In".

Recently, Li et al. (2022) considered a scenario involving two language models; one is a very large pre-trained model, termed the *expert*, and the other is a much smaller version of the same architecture, termed the *amateur*. Importantly, whereas these models share some failure modes and undesirable behaviors, the expert model clearly outperforms the amateur model in language model tasks. Focusing on an open-ended auto-regressive text generation task, they show that it is possible to exploit the contrast between the predictions of the expert and amateur to obtain an improved generated output. They term this method *Contrastive Decoding*. Specifically, they demonstrate that it is sometimes beneficial to prefer predictions to which only the expert model assigns a high probability, versus predictions to which both the expert and the amateur assign high probabilities. Intuitively, since the amateur model has a stronger propensity than the expert for problematic behaviors (e.g., repetitiveness in the case of text generation), we may be able to diminish such behaviors by demoting predictions that are strongly supported by the amateur model.

This scenario relies on a delicate balance: on the one hand, when making a prediction in a relatively simpler context, one would expect both the expert and amateur models to be highly confident about the prediction, and justifiably so; in contrast, where both of them assign very low likelihoods to a certain prediction, these prediction probabilities may be uninformative. Thus, the aim of considering the amateur's predictions during generation is to better inform a choice between a set of relatively plausible predictions given an input; in other words, the predictions of the amateur can serve as a tie-breaker of sorts, helping to highlight which out of a set of plausible alternative predictions is more "expert-like" and less "amateur-like".

Inspired by Li et al. (2022), in this work we ask whether within a *single* language model, intermediate hidden layers can similarly be viewed as "amateur" versions of the final "expert" output layer. Given indications that model representations gradually improve as an input progresses through its layers (Elbayad et al., 2020; Geva et al., 2022), we aim to examine whether the contrast or gap between the outputs at different model layers can be harnessed to obtain better generation predictions. In other words, we posit that the sub-optimal predictions of intermediate hidden layers carry additional information, which can be utilized during inference to obtain more desirable next-token predictions.

Our approach, which we term *Auto-contrastive Decoding (ACD)*, redistributes a given model's probability distribution for the next token, by maximizing the difference between the log-probabilities of the final layer and those of an intermediate hidden layer. This setting, where the expert and amateur are situated within the same language model, and their predictions can be carefully contrasted at inference, is a highly practical one and can be easily applied to language models of different sizes.

Our results show that *ACD* enables getting significantly better predictions out of a given language model, without changing its pre-trained weights.

Figure 1 illustrates an example of *ACD* applied to GPT2, considering layer 12 as the amateur and layer 24 as the expert. Both layers exhibit repetitiveness, but applying *ACD* generates a much improved output altogether.

The main contributions of this work are as follows:

1. We reproduce the findings of Li et al. (2022) using a *single medium-size* model, by suggesting a novel intra-model auto-contrastive setting.

2. We demonstrate that *ACD* improves some aspects of language generation capabilities of pre-trained language models, in essence extracting more knowledge from the model at inference time. We present human evaluation results showing that this brings it to par with larger language models.

3. We release our code and the pre-trained model checkpoints used for experiments in this paper, in order to facilitate further research in this area[1].

2 Related Work

There have been a number of studies on analyzing the characteristics of different layers of transformer models. Rogers et al. (2020); Van Aken et al. (2019) used probing to report that in BERT models the lower layers carry the most information about linear word order, syntactic information is most prominent in the middle layers, and the final layers of BERT are the most task-specific. Van Aken et al. (2019) also show that similar behavior is observed in other transformer models such as GPT2. Geva et al. (2021, 2022) studied the role of feed-forward layers in transformer models. They demonstrate that representations across different layers capture meaningful semantic and syntactic patterns, and describe how model predictions are gradually refined as they progress across the different layers.

Aiming to reduce the computational load of transformers, multiple works have explored early-exiting, i.e., performing some calculations without passing through all of the model layers. Such works allow for an early (fast) 'exit' from neural network calculations – for simple instances that can be solved with high accuracy by lower layers – while using a late (slow) 'exit' for more challenging instances (Simoulin and Crabbé, 2021; Schwartz et al., 2020; Xin et al., 2020; Elbayad et al., 2020; Sun et al., 2022; Schuster et al., 2022).

Decoding algorithms are commonly classified as search-based and sampling-based. Search-based methods (Steinbiss et al., 1994) optimize for the language model log-probabilities, while sampling methods (Holtzman et al., 2019; Fan et al., 2018) draw the next token from a truncated distribution. The idea of using contrast during decoding has been explored in several studies. Liu et al. (2021) combine a pretrained LM with 'expert' LMs and 'anti-expert' LMs, where tokens only get high probability if they are considered likely by the experts and unlikely by the anti-experts. Su et al. (2022)

[1] `https://github.com/IBM/auto-contrastive-generation`

propose constrastive search for decoding, where the generated output is selected from the set of most probable candidates predicted by the model while being discriminative with respect to the context. More recently, Li et al. (2022) suggested to contrast between the likelihood under a large LM (expert) and a small LM (amateur) during decoding. The present work differs significantly from the aforementioned contrastive approaches, in that we contrast the next-token distributions within a single LM, across expert and amateur layers.

3 Auto-contrastive Decoding

We set the goal of applying the **Contrastive Decoding** (CD) method, from Li et al. (2022), using a *single* model rather than two different models (in their setting, a large and a small model of the same model architecture). Thus, we generally follow the CD approach to calculate the next-token predictions, by contrasting the predictions of the expert with those of the amateur. However, in our setting, both the expert and the amateur are situated in the same model, and are defined by two different *layers* of that model. We term this new method **Auto-contrastive Decoding** (ACD). Note that this setting is more practical and less computationally demanding, as it does not require passing every input through two different models in parallel.

Next, we describe how we obtain the expert and the amateur from a single model; and in §3.2, we define the auto-contrastive next-token distribution, given the probability distributions of the expert and the amateur.

3.1 Expert and Amateur in One Model

Given a pre-trained language model, LM_{orig}, we take its final output layer as the *expert*. Similar to Li et al. (2022), we denote $p_{\text{EXP}}(x_t|x_{<t})$ as the next-token probability distribution of this layer, conditioned on the preceding context (x_t being the next token to predict, and $x_{<t}$ is the context that precedes it).

To obtain the *amateur* from the same model, we add a linear head to one of its intermediate hidden layers, making LM_{orig} a multi-exit model (Scardapane et al., 2020; Liu et al., 2022). This new head maps the output of the intermediate layer, given a preceding context, to a probability distribution over the vocabulary for the next token, denoted $p_{\text{AMA}}(x_t|x_{<t})$.

To train only this new head, we freeze all of the existing pre-trained weights of LM_{orig}; we then train the model, applying the same self-supervised objective that was used to pre-train LM_{orig}.

In this training we do not aim to fully reproduce the original pre-training of LM_{orig}; note that we are training a relatively small number of parameters, and thus can use less data and perform fewer training steps. This reduced training is likely to lead to certain disparities between the amateur head and the expert head, as the latter was trained as part of the original LM_{orig} pre-training. Thus, we also train a new expert head, using an identical procedure as the one used to train the amateur head[2].

To amplify the performance gap between the expert and the amateur, Li et al. (2022) introduced another limitation on the amateur model (apart from it being a small version of the expert model) – the preceding context given to the amateur model is restricted, notably shorter than the one provided to the expert model. In *ACD* we opt to abstain from this additional (and somewhat arbitrary) limitation, and both $p_{\text{EXP}}(x_t|x_{<t})$ and $p_{\text{AMA}}(x_t|x_{<t})$ are conditioned on the same full context.

3.2 Auto-contrastive Next-token Distribution

Next, we describe the auto-contrastive decoding, ACD. This method outputs a token-level probability distribution by contrasting the next-token distribution of the expert, $p_{\text{EXP}}(x_t|x_{<t})$, with that of the amateur, $p_{\text{AMA}}(x_t|x_{<t})$.

Following Li et al. (2022), we first implement the CD adaptive plausibility constraint, $\mathcal{V}_{head}(x_{<t})$, defined by:

$$\mathcal{V}_{head}(x_{<t}) = \{x_t \in \mathcal{V} : p_{\text{EXP}}(x_t|x_{<t}) \geq \alpha \max_{x'_t \in \mathcal{V}} p_{\text{EXP}}(x'_t|x_{<t})\} \quad (1)$$

Given a preceding context $x_{<t}$, this constraint selects a subset of plausible next tokens, out of the vocabulary $\mathcal{V}$, whose probabilities are above a threshold. The threshold is a fraction α of the probability of the token with the highest probability in the vocabulary. The hyperparameter α is in the range $[0, 1]$, and it is set to 0.1 in all our experiments, as done by Li et al. (2022).

The score for a *plausible* x_t, i.e., $x_t \in \mathcal{V}_{head}(x_{<t})$, indicating its likelihood to be the next

[2]Our motivation for training a new expert head was to explore a scientifically "cleaner" scenario, where there is a more straightforward relation between the amateur and expert heads. However, considering the results we report in App. C, from a practical standpoint this may not be necessary.

token given the context $x_{<t}$, is calculated by contrasting the probabilities given to it by the expert and by the amateur:

$$S(x_t|x_{<t}) = \log p_{\text{EXP}}(x_t|x_{<t}) - \log p_{\text{AMA}}(x_t|x_{<t}) \tag{2}$$

Note that this contrastive score is only applied to the tokens in $\mathcal{V}_{head}(x_{<t})$. This constraint serves an important purpose in that it helps avoid assigning high probabilities to very unlikely tokens, namely those for which p_{EXP} is very low; at the same time, where the expert is highly confident about a single top prediction, it helps ensure that p_{AMA} does not alter the final outcome[3].

Li et al. (2022) set the score of the rest of the tokens in the vocabulary – those not included in $\mathcal{V}_{head}(x_{<t})$ – to minus infinity. We argue that this design decision has the disadvantage of practically ignoring a large portion of the vocabulary, and thus losing information that can be useful.

For instance, search-based decoding algorithms that rely on $S(x_t|x_{<t})$ will be limited to considering a small subset of the possible next tokens. Additionally, applications that require comparing the probabilities of a predefined and closed set of token options (See Liu et al., 2023), will similarly lose valuable and pertinent information that was initially available in the LM_{orig} probability distribution.

Thus, in *ACD* we retain the probabilities of the tokens not included in $\mathcal{V}_{head}(x_{<t})$, keeping the distribution of the expert head:

$$S_{\text{ACD}}(x_t|x_{<t}) = \begin{cases} S(x_t|x_{<t}) & \text{if } x_t \in \mathcal{V}_{head}(x_{<t}) \\ p_{\text{EXP}}(x_t|x_{<t}) & \text{otherwise} \end{cases} \tag{3}$$

We further transform this score function into a probability distribution. The distribution of the expert head is split into two probability masses; one for the tokens in $\mathcal{V}_{head}(x_{<t})$, and another for the tokens not included in it. We redistribute the former probability mass, weighted by the scores given to each token by Eq. 2:

$$S_{\text{redist}}(x_t|x_{<t}) = \text{softmax}\Big(S(x_t|x_{<t})\Big) \cdot \sum_{x'_t \in \mathcal{V}_{head}(x_{<t})} p_{\text{EXP}}(x'_t|x_{<t}) \tag{4}$$

Replacing $S(x_t|x_{<t})$ with $S_{\text{redist}}(x_t|x_{<t})$ in Eq. 3, we obtain our auto-contrastive decoding probability distribution:

$$p_{\text{ACD}}(x_t|x_{<t}) = \begin{cases} S_{\text{redist}}(x_t|x_{<t}) & \text{if } x_t \in \mathcal{V}_{head}(x_{<t}) \\ p_{\text{EXP}}(x_t|x_{<t}) & \text{otherwise} \end{cases} \tag{5}$$

To summarize, auto-contrastive decoding, ACD, is a method to apply contrastive decoding over a single model. In §3.1 we explain how to create the amateur by adding and training a new head over an intermediate layer. In §3.2 we describe how to obtain a new probability distribution for the next token by contrasting the expert and the amateur.

4 Experimental Setup

To test our approach, we conduct experiments on open-ended text generation, as well as on general language modeling benchmarks, comparing various performance metrics with and without applying auto-contrastive decoding.

In order to analyze changes in performance across model layers, we add multiple new linear exit heads; thus, we also report and compare the baseline model behavior at different exit layers.

4.1 Models

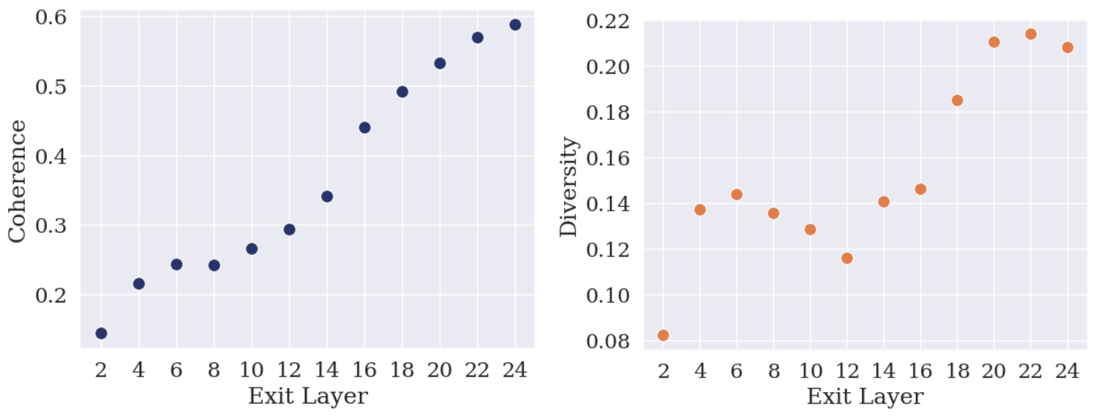
We use pre-trained auto-regressive language models from the GPT family – GPT-2 (Radford et al., 2019) and GPT-Neo[4] – as test models for exploring multi-exit performance and the effects of *ACD*. Specifically, we use the GPT-2 Medium (355M parameters, 24 layers) and GPT-Neo-125M (125M parameters, 12 layers) pre-trained model checkpoints[5].

As outlined in §3.1, we create multi-exit variants of these models, that are identical to the original pre-trained checkpoints, other than the newly-added parameters for several new linear exit heads. To present a more comprehensive analysis, we add multiple heads, one connected to each of the even-numbered layers; thus, we add a total of 12 and 6

[3]Consider for example a case where the token with the maximum probability is assigned a very high probability, e.g., $\max_{x'_t \in \mathcal{V}} p_{\text{EXP}}(x'_t|x_{<t}) > 0.9$, and where p_{AMA} for this token is also quite high. In this scenario, while Eq. 2 may give a very low contrast score $S(x'_t|x_{<t})$, this will be the only token that meets $\mathcal{V}_{head}(x_{<t})$ (Eq. 1), and thus it will nonetheless be selected as the next token despite its low score.

[4]`https://github.com/EleutherAI/gpt-neo`

[5]`https://huggingface.co/gpt2-medium`, `https://huggingface.co/EleutherAI/gpt-neo-125M`



(a) Coherence (b) Diversity

Figure 2: **Open-ended generation for different exit layers.** The plots depict greedy decoding results of a pre-trained *GPT2-Medium* model, using different exit layers for generation. Each point represents an average over the WikiText-103 test examples of the coherence (a) and n-gram diversity (b).

exit heads to *GPT2-Medium* and *GPT-Neo-125M*, respectively. Each head uses the same configuration as the original language modeling head, with outputs for the 50257 tokens in the vocabulary and an input size of 1024 (*GPT-2*) or 768 (*GPT-Neo-125M*).

We train these heads on language modeling using self-supervision over the CC-100 (Conneau et al., 2020) corpus, following a standard pre-training approach (see Appendix A for further details), keeping the original model parameters frozen. As described in §3.1, when training the heads we do not precisely replicate the original pre-training regime; specifically, we use different pre-training data and train for a smaller number of training steps[6]. Nevertheless, we verify the quality of the training process by comparing the performance of a newly-trained final layer exit head to that of the original exit head of the pre-trained model (cf. App. C).

The pre-trained multi-exit base models are used as-is for open-ended text generation and for the benchmarks reported in §5.2. Model training and text generation were performed using the Hugging Face transformers library (v4.22.2) with the pytorch machine learning framework (v1.11.0).

4.2 Tasks and Metrics

4.2.1 Open-ended generation

Following Li et al. (2022), we evaluate open-ended text generation in 3 domains: books, Wikipedia, and news, using the BookCorpus (Zhu et al., 2015), WikiText-103 (Merity et al., 2017), and Wikinews[7] text corpora, respectively. We test open-ended passage continuation by using the first 32 words of a passage as a prompt, and using the multi-exit variant of the pre-trained model to decode up to 100 tokens[8].

Since *ACD* outputs a full probability distribution (see §3.2), it can more naturally be combined with various existing decoding strategies. In this study we combine *ACD* with the following decoding methods: **Greedy search**, **Beam search** (Freitag and Al-Onaizan, 2017; *beam*=5), **Top-k sampling** (Fan et al., 2018, k=50), and **Nucleus (top-p) sampling** (Holtzman et al., 2019; p=0.95).

Generation quality is evaluated using automatic metrics focusing on different axes: aggregated **n-gram diversity** measures the repetitiveness within the generated continuations; **semantic coherence** estimates topic drift by calculating similarity between the prompt and continuation. For further details on these metrics, refer to Su et al. (2022).

We also report results of human evaluation of the generation quality, comparing a sample of generation results across different settings, as explained below in §5.1.

4.2.2 Language modeling benchmarks

We consider the pre-trained multi-exit model, which applies *ACD* at inference time and outputs complete next-token probability distributions (see

[6]for GPT-2, both the training corpus, and a comprehensive description of training details for the original pre-training, have not been publicly released; *GPT-Neo-125M* was originally trained for 572,300 steps over 300 billion tokens.

[7]`http://www.wikinews.org`

[8]Li et al. (2022) decode 256 tokens in continuation to the prompt, however they use stronger base models. With our models, generation deteriorates massively at those lengths.

	wikitext		wikinews		bookcorpus	
	div	*coh*	*div*	*coh*	*div*	*coh*
Greedy	0.21	0.59	0.23	0.57	0.14	0.40
Greedy+*ACD*	**0.75**	**0.63**	**0.74**	**0.61**	**0.62**	**0.50**
Beam-5	0.20	0.61	0.24	0.60	0.08	0.35
Beam-5+*ACD*	**0.57**	0.62	**0.58**	0.61	**0.37**	**0.48**
Top-k	0.96	0.57	0.96	0.55	0.97	0.42
Top-k+*ACD*	0.96	**0.61**	0.96	**0.59**	0.96	**0.47**
Top-p	0.98	0.50	0.98	0.49	0.98	0.36
Top-p+*ACD*	0.98	**0.55**	0.98	**0.54**	0.98	**0.41**

Table 1: **The effect of *ACD* on open-ended generation.** This table lists the automatic quality metrics of n-gram diversity (*div*) and topic coherence with the prompt (*coh*) of a pretrained *GPT2-Medium* model, using different decoding strategies. For each strategy we compare results using the probability distribution of the exit head of the final (24th) model layer, to those obtained using an *ACD* probability distribution, contrasting the final layer next-token predictions with those of exit layer 12.

§3.2), to be a fully functional language model. This model contains the same parameters as LM_{orig} (apart from the added linear exit heads), but differs in its characteristics.

We therefore evaluate the *ACD*-enhanced model as a pre-trained language model, according to benchmarks that are commonly used (e.g., Black et al., 2022; Zhang et al., 2022) to measure language modeling capabilities.

LAMBADA (Paperno et al., 2016) is a popular benchmark that was proposed to encourage computational language models to keep track of information in the broader discourse, rather than paying attention to local context only. It has been shown that language models which exploit the context in a shallow manner perform poorly on this benchmark (Paperno et al., 2016). It is thus a relevant measure of more advanced language understanding abilities.

The typical measure used for reporting progress in language modeling is **Perplexity** (Jelinek et al., 1977), the inverse of the (geometric) average probability assigned to each word in the test set by the model. Perplexity is commonly used as a measure of model quality, due in part to its simplicity and its relation to the maximum likelihood framework.

For running the benchmark tests, we use the Language Model Evaluation Harness library[9] (v0.3.0).

[9] `https://github.com/EleutherAI/lm-evaluation-harness`

5 Results and Analysis

5.1 Open-ended generation

Results for open-ended text generation for the *GPT2-Medium* model are shown in Table 1. For the *greedy* and *beam-search* strategies, which exhibit low diversity of generated texts, we see a significant improvement in diversity when combining them with *ACD*. At the same time, semantic coherence scores with *ACD* are higher in almost all settings tested. Similar effects of *ACD* can be observed for the smaller *GPT-Neo-125M* model (App. Table 5).

The gains in text diversity highlight one major effect of *ACD*, which is that of reducing repetitiveness in generation. This is true both to short loops, such as two tokens being generated again and again, as well as longer ones. Also, in some cases texts generated by the top layer simply repeat/variate the prompt. See Table 2 for examples of the above failures and their mitigation.

Given the dramatic performance boost given by *ACD*, as seen in Tables 1 and 5, we further ask how *ACD*-enhanced generation outputs would compare to those of a *larger model* with more advanced capabilities. To this end, we perform open-ended generation using the *GPT2-XL* model (1.5B parameters). As can be seen in Table 3, *GPT2-Medium* (355M parameters) that is enhanced by *ACD* significantly outperforms its larger scale counterpart.

To verify that these results are robust and not an artifact of the automatic measures used, we conduct human evaluation of a sample of generation outputs from the results in Table 3, presenting the prompt and pairs of generated texts to human annotators and asking them to compare the quality of outputs. Results indicate that outputs from *GPT2-XL* were twice as likely to be judged as better compared to the baseline *GPT2-Medium*; but strikingly, *GPT2-Medium* outputs obtained using *ACD* were overall judged as slightly better than those of the much larger *GPT2-XL*. For details on the human evaluation task, refer to App. D.

In Fig. 2a we portray the behavior of the automatic coherence measure when relying on the outputs of different *GPT2-Medium* exits. It appears that the generation coherence, i.e., the semantic relatedness between the prompt and generated continuation, rises consistently as progressing from lower to higher layers. Presumably, this reflects a gradual decrease in topic drift behaviors and an increased ability to generate longer sequences that

Mitigated failure	Prompt	Greedy *ACD*	Greedy
Short loop	The use of iron instead of wood as the primary material of	furniture could have created problems, says study leader Prof. Iain Kelly from the School of Materials Science and Engineering at the University	the building blocks of the building blocks of the building blocks of the building blocks of the building blocks of the building blocks of the
Longer loop	Du Fu's political comments are based on emotion rather than calculation:	if his party loses power, he fears, China will face an even more uncertain future than it faces now. He fears a	he is a man who has been in the trenches for years, and he is a man who has been in the trenches for
Prompt repeated	The first ironclads to have all-steel armor were the Italian Caio Duilio	in 1230 and the Saxon Magnus in 1252, both of whom wore steel shields. Iron armor became so common that	and the German Wilhelm von Habsburg. The first ironclads to have all-steel armor were the Italian Caio

Table 2: **Examples of common failures diminished when applying *ACD*.**

	Diversity	Coherence
GPT2-Medium	0.22	0.63
GPT2-XL	0.31	0.63
GPT2-Medium + ACD	**0.75**	0.63

Table 3: **Scale effects of *ACD* on open generation.** Depicted are topic coherence and n-gram diversity of generation outputs over WikiText-103, for 3 settings: a large model (*GPT2-XL*, 1.5B parameters), a medium-sized model (*GPT2-Medium*, 355M parameters, using its original exit head), and the same medium-sized model applying *ACD* at inference time, contrasting the next-token predictions of the final (24th) layer and layer 12.

remain semantically coherent.

Fig. 2b depicts the diversity of open-ended generation across layers. Interestingly, this measure exhibits more complex patterns, rising and falling as we progress from lower to higher layers. As is common with automatic quality metrics for text generation, we see this as an indication that n-gram repetition provides only a partial window into the generation quality, particularly where the diversity is overall quite low. Moreover, the nature of outputs may undergo phase shifts as they improve. For instance, generated sequences may shift from being diverse but unrelated to the inputs in lower layers, to texts that are semantically related to the prompt but highly repetitive, and so on.

5.2 Language modeling benchmarks

Results for the LAMBADA benchmark task, for individual exit layers of *GPT2-Medium* and for *ACD* generation, are shown in Figure 3. The accuracy and the perplexity metrics of this benchmark dataset both improve as progressing along the model layers. In both cases, performance is further improved by applying *ACD*, with substantial gains in accuracy. Similar gains are obtained for the *GPT-Neo-125M* model (App. Figure 5).

This is a non-trivial finding, in that it provides an indication that by using *ACD* we enable the model to more accurately take into account the broader context and long-range dependencies in the text.

As in §5.1, one may further ask how these gains compare to the performance reached by a larger pre-trained model. Indeed, as shown in Table 4, *GPT2-Medium* enhanced by *ACD* is on par with the larger *GPT2-XL* model (1.5B parameters) on the LAMBADA benchmark, achieving improved accuracy but also somewhat inferior perplexity.

Figure 4 depicts the word-level perplexity over the general WikiText-2 dataset. As can be seen, perplexity behaves as expected across model layers. For this general corpus, *ACD* does not improve the overall perplexity beyond that of the final exit layer.

Thus, we see that *ACD* provides a substantial benefit for the challenging LAMBADA data, that specifically measures a model's advanced ability to look at broader context windows, but not for the overall perplexity over a general text corpus. While this is an initial finding that deserves further exploration, one interpretation is that *ACD* specifically strengthens "higher-layer behaviors", such as those measured by the challenging LAMBADA task, but also induces other types of biases into the model's output probability distributions.

6 Discussion

In this work we develop an approach that contrasts different model layers, improving the output probabilities of a generative model. Applying it to existing pre-trained language models, we demonstrate that intermediate low-performing model layers can in some cases inform the predictions of the high-performance final layer. This setting is of particular interest due to its practicality and flexibility, as it can be applicable to models of different sizes and is

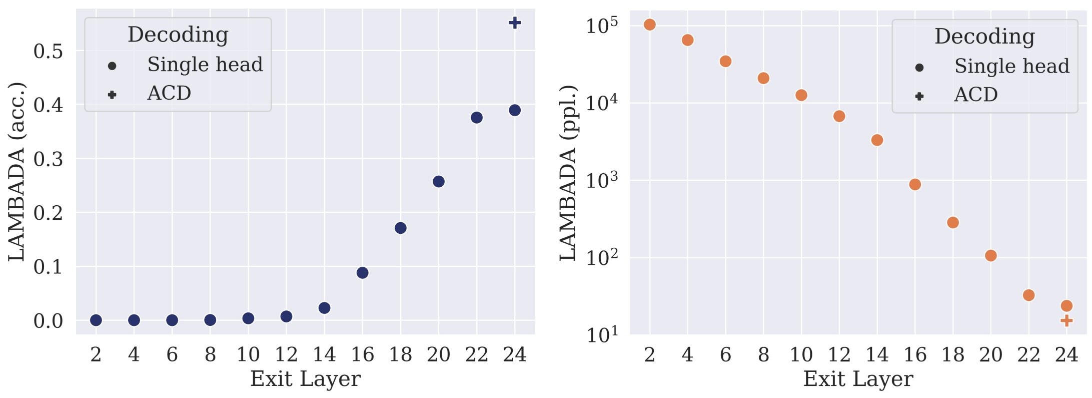


Figure 3: **GPT2 performance on the LAMBADA benchmark**. Plots depict accuracy (left, higher is better) and perplexity (right, lower is better; presented in log scale) on LAMBADA across layers. Individual *GPT2-Medium* exits are denoted by •; results for the *ACD* probability distribution, contrasting layers 24 and 12, are denoted by +.

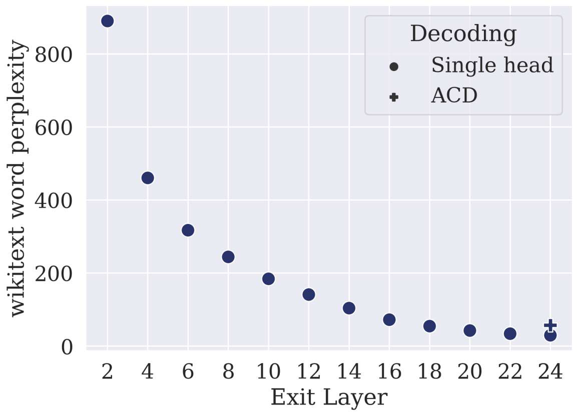


Figure 4: **GPT2 performance on the WikiText-2 benchmark**. The plot depicts the word-level perplexity (lower is better) over the WikiText-2 corpus. Individual *GPT2-Medium* exit layers are denoted by •, while results for the *ACD* probability distribution, contrasting layers 24 and 12, are denoted by +.

LAMBADA	**Acc.** $\uparrow$	**Ppl.** $\downarrow$
GPT2-Medium	0.43	18.3
GPT2-XL	0.51	**10.6**
GPT2-Medium + ACD	**0.55**	15.4

Table 4: **Scale effects of *ACD* on LAMBADA.** Depicted are the accuracy and perplexity scores of the LAMBADA benchmark, for 3 settings: a large model (*GPT2-XL*, 1.5B parameters), a medium-sized model (*GPT2-Medium*, 355M parameters, using its original exit head), and the same medium-sized model applying *ACD* at inference time, contrasting the next-token predictions of the final (24th) layer and layer 12.

utilized during inference via a single forward pass.

But more broadly, our findings bring forth an enticing notion, that one would be able to make more out of an existing model simply by considering the predictions of intermediate layers (which are typically ignored). This idea is somewhat counter-intuitive, as language models are in a sense optimized – and often in a long pretraining process over massive corpora – for the quality of their final layer representations. At the same time, thematically this notion is in line with works that describe the computations in transformer models as a linear-like progression, where each layer refines the representations of the previous ones, and where even the representations of specific tokens can shift in a consistent direction along with the progression across layers (Geva et al., 2021, 2022). Loosely speaking, if the changes from one layer to the next can sometimes track a vector of improvement with a discernible direction, then in theory one could try and "extend" this vector; and doing so may help estimate what a *larger* model, one with additional layers, would have said about a particular instance. We see these as interesting avenues both for theoretical study, and for empirical explorations as to whether surprising findings such as those presented here can be applied to real-world use-cases.

Here we present an initial, and relatively simple, algorithm for performing the *ACD* contrast between layers. As in Li et al. (2022), our formulation still relies on a somewhat arbitrary hyperparameter α; also, contrast is always done with respect to a single particular exit layer, and choosing the most appropriate layer for contrast may not be trivial. Here, for simplicity and robustness, we did not attempt to optimize these two important hyperparameters, and used a single configuration throughout our experiments. However, we see much room for future work on improving these details, and finding ways to intelligently choose which layers to contrast and how to combine between them.

An interesting avenue for future work concerns the effect of *ACD* when applied not just to a pre-trained model, but to one fine-tuned for a particular downstream task. Specifically, it may be that specific types of generation tasks may derive more benefit from *ACD*, depending on their reliance on more "high-level" model capabilities, and also on the importance of diversity in generated outputs.

The present work focuses specifically on generative models, and on improving the quality of text generation outputs and next-token predictions. However, the basic approach of looking at the outputs of intermediate layers and using them to inform model predictions is a general one, and is thus also worth exploring in other contexts, such as classification tasks.

To sum, our findings indicate that our proposed approach, *ACD*, can be of great practical value, in that it significantly boosts the performance of a generative language model with a minimal computational cost. This approach suggests new avenues on how to best extract knowledge from a language model and more efficiently utilize its parameters.

Limitations

One of the primary limitations of this work is that this is essentially an empirical study. Although we provide extensive experiments to show that the proposed approach demonstrates significantly better results in different settings, currently we do not provide any theoretical guarantees for this approach. Second, many of our experiments would not be easily reproduced in languages other than English, that lack sufficient linguistic resources. During this study we used the *GPT-2* and *GPT-Neo* language models, which have been trained on large amounts of English text. Finally, anecdotally we observed that this approach can also increase hallucination behaviors, which are a common issue with many text generation models. During application, one would have to take necessary measures to monitor the hallucinations produced by the model.

Acknowledgements

We thank our many colleagues for their valuable input on this research effort, and owe particular thanks to Liat Ein-Dor and Leshem Choshen for their advice and assistance.

References

Sid Black, Stella Biderman, Eric Hallahan, Quentin Anthony, Leo Gao, Laurence Golding, Horace He, Connor Leahy, Kyle McDonell, Jason Phang, et al. 2022. GPT-NeoX-20B: An open-source autoregressive language model. *arXiv:2204.06745*.

Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. 2020. Unsupervised cross-lingual representation learning at scale. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8440–8451, Online. Association for Computational Linguistics.

Alexander Yom Din, Taelin Karidi, Leshem Choshen, and Mor Geva. 2023. Jump to conclusions: Short-cutting transformers with linear transformations. *arXiv:2303.09435*.

Maha Elbayad, Jiatao Gu, Edouard Grave, and Michael Auli. 2020. Depth-adaptive transformer. In *ICLR 2020-Eighth International Conference on Learning Representations*, pages 1–14.

Angela Fan, Mike Lewis, and Yann Dauphin. 2018. Hierarchical neural story generation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 889–898, Melbourne, Australia. Association for Computational Linguistics.

Markus Freitag and Yaser Al-Onaizan. 2017. Beam search strategies for neural machine translation. In *Proceedings of the First Workshop on Neural Machine Translation*, pages 56–60, Vancouver. Association for Computational Linguistics.

Mor Geva, Avi Caciularu, Kevin Ro Wang, and Yoav Goldberg. 2022. Transformer feed-forward layers build predictions by promoting concepts in the vocabulary space. *arXiv:2203.14680*.

Mor Geva, Roei Schuster, Jonathan Berant, and Omer Levy. 2021. Transformer feed-forward layers are key-value memories. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 5484–5495, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. 2019. The curious case of neural text degeneration. In *International Conference on Learning Representations*.

Fred Jelinek, Robert L Mercer, Lalit R Bahl, and James K Baker. 1977. Perplexity—a measure of the difficulty of speech recognition tasks. *The Journal of the Acoustical Society of America*, 62(S1):S63–S63.

Xiang Lisa Li, Ari Holtzman, Daniel Fried, Percy Liang, Jason Eisner, Tatsunori Hashimoto, Luke Zettlemoyer, and Mike Lewis. 2022. Contrastive decoding: Open-ended text generation as optimization. *arXiv:2210.15097*.

Alisa Liu, Maarten Sap, Ximing Lu, Swabha Swayamdipta, Chandra Bhagavatula, Noah A Smith, and Yejin Choi. 2021. DExperts: Decoding-time controlled text generation with experts and anti-experts. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 6691–6706. Association for Computational Linguistics.

Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2023. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Computing Surveys*, 55(9):1–35.

Xiangyang Liu, Tianxiang Sun, Junliang He, Jiawen Wu, Lingling Wu, Xinyu Zhang, Hao Jiang, Zhao Cao, Xuanjing Huang, and Xipeng Qiu. 2022. Towards efficient NLP: A standard evaluation and a strong baseline. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3288–3303, Seattle, United States. Association for Computational Linguistics.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2017. Pointer sentinel mixture models. In *International Conference on Learning Representations*.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Ngoc-Quan Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. 2016. The LAMBADA dataset: Word prediction requiring a broad discourse context. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1525–1534, Berlin, Germany. Association for Computational Linguistics.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Anna Rogers, Olga Kovaleva, and Anna Rumshisky. 2020. A primer in BERTology: What we know about how BERT works. *Transactions of the Association for Computational Linguistics*, 8:842–866.

Simone Scardapane, Michele Scarpiniti, Enzo Baccarelli, and Aurelio Uncini. 2020. Why should we add early exits to neural networks? *Cognitive Computation*, 12(5):954–966.

Tal Schuster, Adam Fisch, Jai Gupta, Mostafa Dehghani, Dara Bahri, Vinh Q Tran, Yi Tay, and Donald Metzler. 2022. Confident adaptive language modeling. In *Advances in Neural Information Processing Systems*, volume 35, pages 17456–17472. Curran Associates, Inc.

Roy Schwartz, Gabriel Stanovsky, Swabha Swayamdipta, Jesse Dodge, and Noah A. Smith. 2020. The right tool for the job: Matching model and instance complexities. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6640–6651, Online. Association for Computational Linguistics.

Antoine Simoulin and Benoit Crabbé. 2021. How many layers and why? An analysis of the model depth in transformers. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing: Student Research Workshop*, pages 221–228, Online. Association for Computational Linguistics.

Volker Steinbiss, Bach-Hiep Tran, and Hermann Ney. 1994. Improvements in beam search. In *Third international conference on spoken language processing*.

Yixuan Su, Tian Lan, Yan Wang, Dani Yogatama, Lingpeng Kong, and Nigel Collier. 2022. A contrastive framework for neural text generation. *arXiv:2202.06417*.

Tianxiang Sun, Xiangyang Liu, Wei Zhu, Zhichao Geng, Lingling Wu, Yilong He, Yuan Ni, Guotong Xie, Xuanjing Huang, and Xipeng Qiu. 2022. A simple hash-based early exiting approach for language understanding and generation. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 2409–2421, Dublin, Ireland. Association for Computational Linguistics.

Betty Van Aken, Benjamin Winter, Alexander Löser, and Felix A Gers. 2019. How does BERT answer questions? a layer-wise analysis of transformer representations. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pages 1823–1832.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Ji Xin, Raphael Tang, Jaejun Lee, Yaoliang Yu, and Jimmy Lin. 2020. DeeBERT: Dynamic early exiting for accelerating BERT inference. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2246–2251, Online. Association for Computational Linguistics.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. 2022.

OPT: Open pre-trained transformer language models. *arXiv:2205.01068*.

Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. 2015. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *Proceedings of the IEEE international conference on computer vision*, pages 19–27.

A Pre-training details

For training the additional linear heads in our multi-exit versions of *GPT2-Medium* and *GPT-Neo-125M*, we apply a training regime to the pre-trained models, while freezing the parameters of the original pre-trained model checkpoints (see §3.1).

For runtime considerations, we train all the added linear heads (12 and 6 heads in total for *GPT2-Medium* and *GPT-Neo-125M*, respectively) within a single training run, where a cross-entropy loss is calculated for the outputs of each individual linear head with respect to the labels, and the total training loss is calculated as the sum of these losses. Note that since each head is only connected to its exit layer m, and the shared pre-trained model parameters are kept frozen, this setup is roughly equivalent to training each of the linear heads separately.

Training was conducted with self-supervision over the English portion of the CC-100 (Conneau et al., 2020) corpus[10]. We used 20M instances out of the full dataset. Each text was tokenized, and the different tokenized instances were then joined together into chunks with a maximum sequence length of 512. Thus, no padding was applied to the examples. Following the tokenization and chunking, the training data consisted of $\sim$ 1.3M training examples ($\sim$ 650M tokens). Training was performed using a causal language modeling objective, where the cross-entropy loss is calculated between the autoregressively generated outputs of the language modeling head and the input tokens (of length 512), which serve as the label.

The linear heads of each model were trained for 3 epochs over the chunked texts, using the AdamW optimizer, a learning rate of 2×10^{-4} with a linear decay scheduler, and a train batch size of 64. Training runs totalled approximately 24 / 55 GPU hours for *GPT-Neo* / *GPT2-Medium*, respectively, on Nvidia A100 GPUs.

[10]`https://huggingface.co/datasets/cc100`

	wikitext		wikinews		bookcorpus	
	div	*coh*	*div*	*coh*	*div*	*coh*
Greedy	0.09	0.57	0.08	0.54	0.06	0.35
Greedy+*ACD*	**0.32**	**0.62**	**0.32**	**0.61**	**0.20**	**0.49**
Beam-5	0.08	0.59	0.08	0.56	0.05	0.33
Beam-5+*ACD*	**0.15**	0.60	**0.15**	**0.60**	**0.10**	**0.48**
Top-k	**0.95**	0.56	**0.95**	0.54	**0.95**	0.40
Top-k+*ACD*	0.91	**0.62**	0.92	**0.60**	0.92	**0.48**
Top-p	0.98	0.48	0.98	0.47	0.98	0.35
Top-p+*ACD*	0.97	**0.56**	0.97	**0.54**	0.97	**0.41**

Table 5: **The effect of *ACD* on open-ended generation.** This table lists the automatic generation quality metrics of n-gram diversity (*div*) and topic coherence with the prompt (*coh*) of a pretrained *GPT-Neo-125M* model, using different decoding strategies. For each strategy we compare results using the probability distribution of the exit head of the final (12th) model layer, to those obtained using an *ACD* probability distribution, contrasting the final layer next-token predictions with those of exit layer 8.

B *GPT-Neo-125M* results

The open-generation results for the *GPT-Neo-125M* model are shown in Table 5. The results for this model over the LAMBADA benchmark are depicted in Fig. 5.

C Comparison to the original LM heads

As noted in §3.1, in order to reduce training disparities between the expert and the amateur we train a new expert head, rather than using the model's original exit head as the expert. Here, we compare the performance of the newly-trained expert heads to that of the original language modeling heads. In addition, we report the effects of *ACD* when using the original expert head for the *ACD* contrast procedure.

As can be seen in Table 6, our newly-trained expert heads are slightly inferior to the original language modeling heads, presumably due to the more limited pre-training regime of the new heads. Nevertheless, *ACD* that relies on the newly-trained expert head clearly outperforms the original language modeling head in open-generation and LAMBADA metrics (as also shown in Tables 3 and 4).

The results of *ACD* when contrasting between the *original* LM head and our newly-trained amateur head are overall rather similar. Thus, despite

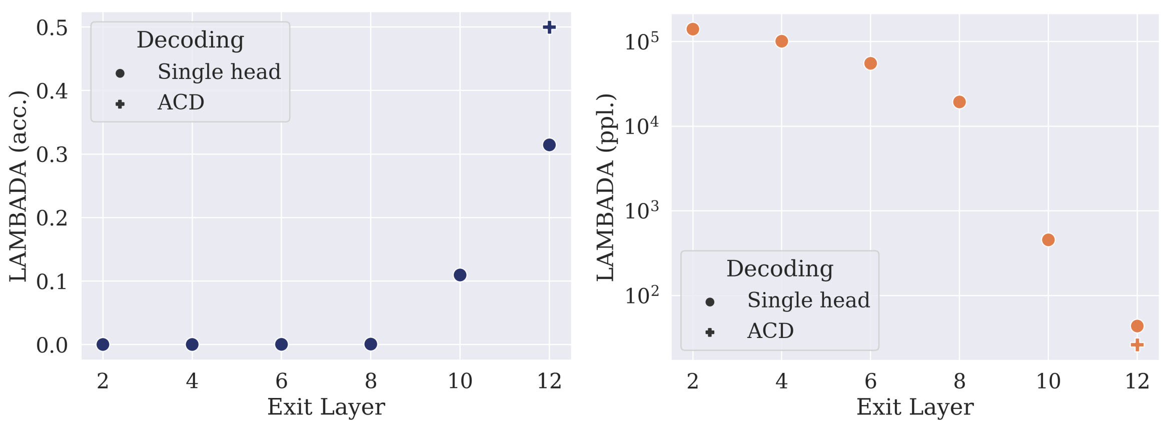


Figure 5: **GPT-Neo performance on the LAMBADA benchmark**. Plots depict accuracy (left, higher is better) and perplexity (right, lower is better; presented in log scale) on the LAMBADA language modeling task across different layers. Individual *GPT-Neo-125M* exit layers are denoted by •, while results for the *ACD* probability distribution, contrasting layers 12 and 8, are denoted by +.

the more noisy or unpredictable nature of the disparities between the exit heads in this case (given that they were trained in a different pre-training regime over different training examples), it appears the effects of applying *ACD* are relatively robust to such a scenario.

D Human evaluation

We conducted two evaluations for open-ended generation quality of the models:

- Comparing greedy decoding outputs of ***GPT2-XL*** and ***GPT2-Medium***
- Comparing greedy decoding outputs of ***GPT2-XL*** to ***GPT2-Medium* with *ACD***

As input for inference, we randomly sampled 40 texts from the WikiText-103 dataset. Following the setting described in §4.2.1, we used the first 32 words of those texts as prompts and for each evaluated model extracted up to 100 tokens of the decoded text. The same prompts were used for the two sets of evaluations, and thus also identical generation outputs of the *GPT2-XL* Greedy setting.

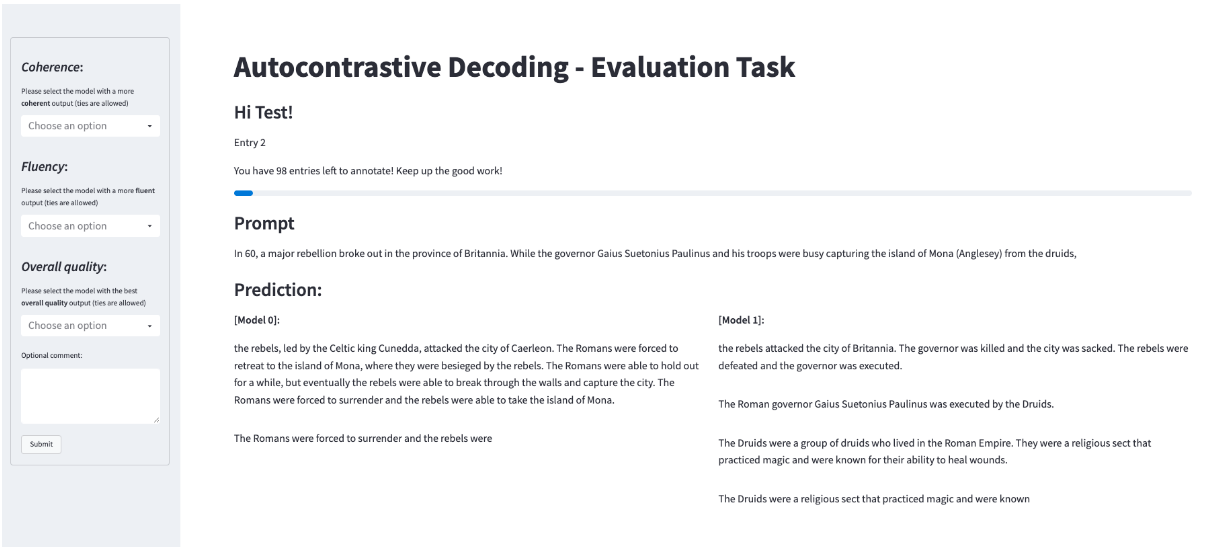
3 NLP experts labeled the 80 resulting instances, consisting of a prompt and inferences from two models. For each instance, they were asked to select the better model on 3 different aspects, in separate questions: *fluency*, *coherence* and *overall quality* (Figure 6). For each question they could select either 'model A', 'model B' or a tie. The inferences were shuffled such that 'model A' for each displayed instance was randomly selected from either the *GPT2-XL* Greedy model or its counterpart. The sets of evaluations (i.e., *GPT2-XL* vs. *GPT2-Medium* and *GPT2-XL* vs. *GPT2-Medium* + *ACD*) were also shuffled, such that annotators did not know which pair of models they are annotating.

The final label for each instance is obtained by the majority choice of the annotators. A *tie* majority label is achieved either when the majority of annotations is *tie* or when no majority is obtained (which in this setting can only occur when annotations are equally distributed - one for each model and one *tie*).

Label distributions are shown in Figures 7, 8. Inter-annotator agreement for those tasks, obtained by averaging Cohen's Kappa for all annotator pairs, in each task, for each question is as follows - 0.15 for the fluency question, 0.34 for the coherence question and 0.42 for the overall quality question. An image of the task is shown in Figure 6.

	Diversity ↑	Coherence ↑	LAMBADA acc. ↑	Perplexity ↓
GPT2-Medium L-24_{orig}	0.22	0.63	0.43	26.8
GPT2-Medium L-24_{new}	0.21	0.59	0.39	30.0
GPT2-Medium L-24_{orig} + *ACD*	0.62	0.64	0.56	71.3
GPT2-Medium L-24_{new} + *ACD*	0.75	0.63	0.55	57.2
GPT-Neo-125M L-12_{orig}	0.12	0.60	0.37	32.3
GPT-Neo-125M L-12_{new}	0.09	0.57	0.31	38.5
GPT-Neo-125M L-12_{orig} + *ACD*	0.30	0.62	0.53	68.1
GPT-Neo-125M L-12_{new} + *ACD*	0.32	0.62	0.50	71.8

Table 6: **Comparison to the original LM exit heads.** Depicted are open-generation metrics (using greedy decoding over WikiText-103), LAMBADA benchmark accuracy, and WikiText-2 perplexity of the *GPT2-Medium* and *GPT-Neo-125M* models. For each model, 4 settings are shown: using its original exit head (*L*-$_{orig}$), using our newly-trained final layer exit head (*L*-$_{new}$), and the results of applying *ACD* at inference time, contrasting the next-token predictions of a newly-trained intermediate layer exit head with those of either the original (*L*-$_{orig}$ + *ACD*) or newly-trained (*L*-$_{new}$ + *ACD*) final layer exit.


Coherence:

Please select the model with a more **coherent** output (ties are allowed)

Choose an option

Fluency:

Please select the model with a more **fluent** output (ties are allowed)

Choose an option

Overall quality:

Please select the model with the best **overall quality** output (ties are allowed)

Choose an option

Optional comment:

Submit

Autocontrastive Decoding - Evaluation Task

Hi Test!

Entry 2

You have 98 entries left to annotate! Keep up the good work!

Prompt

In 60, a major rebellion broke out in the province of Britannia. While the governor Gaius Suetonius Paulinus and his troops were busy capturing the island of Mona (Anglesey) from the druids,

Prediction:

[Model 0]:

the rebels, led by the Celtic king Cunedda, attacked the city of Caerleon. The Romans were forced to retreat to the island of Mona, where they were besieged by the rebels. The Romans were able to hold out for a while, but eventually the rebels were able to break through the walls and capture the city. The Romans were forced to surrender and the rebels were able to take the island of Mona.

The Romans were forced to surrender and the rebels were

[Model 1]:

the rebels attacked the city of Britannia. The governor was killed and the city was sacked. The rebels were defeated and the governor was executed.

The Roman governor Gaius Suetonius Paulinus was executed by the Druids.

The Druids were a group of druids who lived in the Roman Empire. They were a religious sect that practiced magic and were known for their ability to heal wounds.

The Druids were a religious sect that practiced magic and were known

Figure 6: **Human Evaluation Task UI**

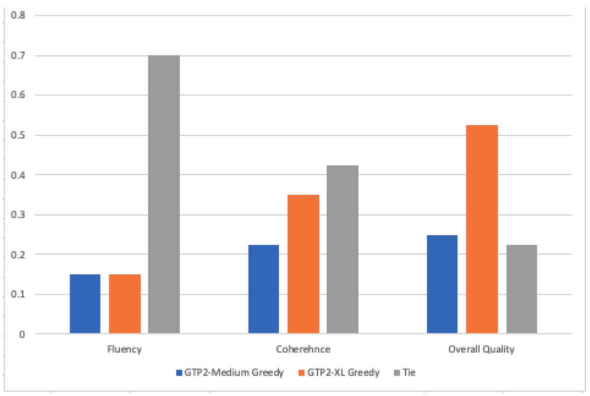


Figure 7: **Human Evaluation results comparing greedy decoding generation outputs of *GPT2-XL* and *GPT2-Medium*.** This figure shows the distribution of majority labels for each of the 3 task questions.

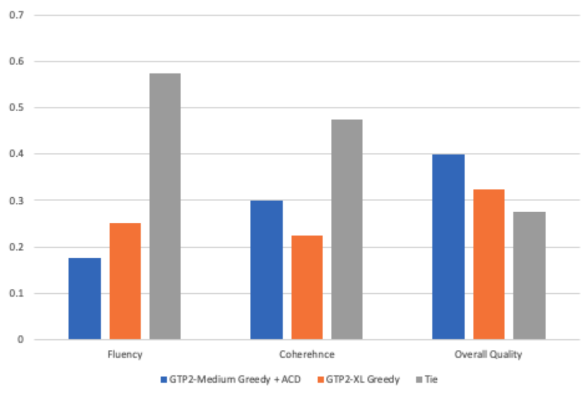


Figure 8: **Human Evaluation results comparing greedy decoding generation outputs of *GPT2-XL* and *GPT2-Medium + ACD*.** This figure shows the distribution of majority labels for each of the 3 task questions.

ACL 2023 Responsible NLP Checklist

A For every submission:

☑ A1. Did you describe the limitations of your work?
Limitations section

☑ A2. Did you discuss any potential risks of your work?
Limitations section

☑ A3. Do the abstract and introduction summarize the paper's main claims?
Abstract and Section 1

☒ A4. Have you used AI writing assistants when working on this paper?
Left blank.

B ☑ Did you use or create scientific artifacts?

Sections 3, 4

☑ B1. Did you cite the creators of artifacts you used?
Section 4

☑ B2. Did you discuss the license or terms for use and / or distribution of any artifacts?
Section 4

☒ B3. Did you discuss if your use of existing artifact(s) was consistent with their intended use, provided that it was specified? For the artifacts you create, do you specify intended use and whether that is compatible with the original access conditions (in particular, derivatives of data accessed for research purposes should not be used outside of research contexts)?
All artifacts, existing and created, are under permissive licenses.

☒ B4. Did you discuss the steps taken to check whether the data that was collected / used contains any information that names or uniquely identifies individual people or offensive content, and the steps taken to protect / anonymize it?
While the data used does contain some unique identifiers and offensive content, all of it comes from publicly available sources, and was only used for quantitative and qualitative analysis.

☒ B5. Did you provide documentation of the artifacts, e.g., coverage of domains, languages, and linguistic phenomena, demographic groups represented, etc.?
As stated in the Limitations section, the models and data are limited to English corpora (both general and domain-specific).

☑ B6. Did you report relevant statistics like the number of examples, details of train / test / dev splits, etc. for the data that you used / created? Even for commonly-used benchmark datasets, include the number of examples in train / validation / test splits, as these provide necessary context for a reader to understand experimental results. For example, small differences in accuracy on large test sets may be significant, while on small test sets they may not be.
Appendix A

C ☑ Did you run computational experiments?

Section 4

☑ C1. Did you report the number of parameters in the models used, the total computational budget (e.g., GPU hours), and computing infrastructure used?
Section 4, Appendix A

The Responsible NLP Checklist used at ACL 2023 is adopted from NAACL 2022, with the addition of a question on AI writing assistance.

☑ C2. Did you discuss the experimental setup, including hyperparameter search and best-found hyperparameter values?
Section 4

☑ C3. Did you report descriptive statistics about your results (e.g., error bars around results, summary statistics from sets of experiments), and is it transparent whether you are reporting the max, mean, etc. or just a single run?
Section 4, 5

☑ C4. If you used existing packages (e.g., for preprocessing, for normalization, or for evaluation), did you report the implementation, model, and parameter settings used (e.g., NLTK, Spacy, ROUGE, etc.)?
Section 4; Our full implementation is released on GitHub

D ☑ Did you use human annotators (e.g., crowdworkers) or research with human participants?
Section 5.1, Appendix D

☑ D1. Did you report the full text of instructions given to participants, including e.g., screenshots, disclaimers of any risks to participants or annotators, etc.?
Appendix D

☒ D2. Did you report information about how you recruited (e.g., crowdsourcing platform, students) and paid participants, and discuss if such payment is adequate given the participants' demographic (e.g., country of residence)?
The annotators are part of the research group that authored the paper.

☒ D3. Did you discuss whether and how consent was obtained from people whose data you're using/curating? For example, if you collected data via crowdsourcing, did your instructions to crowdworkers explain how the data would be used?
The annotators are part of the research group that authored the paper.

☑ D4. Was the data collection protocol approved (or determined exempt) by an ethics review board?
Appendix D

☑ D5. Did you report the basic demographic and geographic characteristics of the annotator population that is the source of the data?
Appendix D