Interpolation versus Self-Recovery: Does the *k*NN-LM Retriever Introduce Exposure Bias?

Anonymous ACL submission

Abstract

Interpolation-based Language Models (LMs), a type of retrieval-augmented LM, have shown impressive decreases in perplexity and as a result, have been adapted to a variety of downstream tasks. Interpolation-based LMs interpolate the next token probability distributions of a neural LM and that of a datastore retriever to generate text. Wang et al. (2023) claims that despite these decreases in perplexity, interpolation-based LMs like *k*NN-LM do not improve the quality of open-ended text generation. They hypothesize that the *k*NN-LM retriever introduces exposure bias and impedes the LM's self-recovery abilities. In this paper, we reassess these hypotheses. We do so by utilizing automatic evaluation metrics to measure exposure bias and self-recovery on a variety of LMs and configurations. Our findings are threefold: 1) The *k*NN-LM retriever generally does not induce any major differences in exposure bias and self-recovery metrics. 2) The exception to this is when the interpolation weight (λ) hyperparameter is tuned solely to minimize perplexity. 3) All LMs and configurations that we evaluate lack consistent self-recovery abilities. Our work aims to inform further evaluations of interpolation-based LMs through these findings.

1 Introduction

Interpolation-based LMs such as *k*NN-LM (Khandelwal et al., 2020b) generate text by interpolating a neural LM's next token distribution with that of a retriever to an external datastore and applying a decoding algorithm to the interpolated distribution (Wang et al., 2023). The addition of retrieval interpolation has demonstrated notable success and popularity, with interpolated LMs showing large decreases in perplexity compared to their base LM counterparts without the need for any additional training (Khandelwal et al., 2020b; Wang et al., 2023; Min et al., 2023; Liu et al., 2024b; Geng et al., 2024). Consequently, interpolation-based LMs have been adapted to many downstream tasks (Khandelwal et al., 2020a; Shi et al., 2022; Geng et al., 2024). However, the evaluation of interpolation-based LMs for open-ended text generation has been relatively sparse. To our knowledge, Wang et al. (2023) is the only work thus far to do so extensively. The lack of evaluation and understanding of these LMs for this task is a particularly pernicious issue, as recent interpolation-based LMs such as Infini-gram (Liu et al., 2024b) achieve impressive perplexity losses yet struggle with open-ended text generation, with there being little to no information as to *why* this issue occurs.

With this in mind, we turn towards Wang et al. (2023), which presents two interlinked hypotheses regarding this issue in the context of *k*NN-LM: 1) The *k*NN-LM retriever introduces exposure bias. 2) The introduction of exposure bias impedes the self-recovery abilities of the base LM. We choose these particular hypotheses as there are amble quantification methods per He et al. (2021b) which were not utilized in Wang et al. (2023)'s analysis, hence, a deeper investigation may shine new light regarding *k*NN-LM's effect on open-ended text generation. Yet before doing so, we first must disambiguate the definitions of exposure bias and self-recovery. During training time (under common MLE objectives), an LM is trained on ground truth tokens, yet at inference time can only rely on its own generated prefixes. **Exposure bias** is the hypothesized issue where this discrepancy leads to a significant imbalance in training and inference performance, as the LM is only trained to perform well on ground truth prefixes (Ranzato et al., 2015; Bengio et al., 2015; Schmidt, 2019; He et al., 2021b). This performance drop is characterized by the incremental accumulation of errors (e.g. hallucinations, degenerative text) during inference time (Wang and Sennrich, 2020; Chiang and Chen, 2021). **Self-recovery** refers to the hypothesized ability of LMs

to recover from distortions in its prefix by attending to more recent contexts versus irrelevant long-range contexts, acting as a counter to exposure bias (He et al., 2021b; Wang et al., 2023).

Wang et al. (2023) posits two reasons why the design of *k*NN-LM leaves the underlying neural LM particularly vulnerable to exposure bias. Firstly, the *k*NN-LM retriever, being constructed from the training set, may face challenges when processing increasingly out-of-distribution input text during inference. Secondly, the retriever may also over-rely on neural LM generated queries containing artifacts or further distributional differences at inference time. Similarly, Wang et al. (2023) hypothesizes that while a standard LM may be able to counter the harmful effects of exposure bias through self-recovery, it may be ill-equipped to deal with exposure bias reintroduced by the *k*NN-LM retriever as a result of these vulnerabilities. Wang et al. (2023) supports their hypotheses by demonstrating that the *k*NN-LM retriever's Shannon entropy (Shannon, 1948) relative to the base LM increases as the token length increases, indicating an incremental accumulation of errors commonly associated with exposure bias. This analysis namely focuses on the differences between the neural LM and the *k*NN-LM retriever as opposed to when they are interpolated together and lacks more robust quantification methods for measuring exposure bias such those used in He et al. (2021b).

Due to these limitations, we found it necessary to re-evaluate Wang et al. (2023)'s hypotheses that the *k*NN-LM retriever introduces exposure bias and impedes an LM's ability to self-recover. To quantify exposure bias and self-recovery, we use the metrics provided in He et al. (2021b). We evaluate a variety of LMs, including the same GPT-2 model (Radford et al., 2019; Alon et al., 2022) and *k*NN-LM implementation (Alon et al., 2022) utilized by Wang et al. (2023) on the same Wikitext-103 dataset (Merity et al., 2016) to investigate if the *k*NN-LM truly introduces exposure bias and impedes self-recovery. In our reassessment, we find the following:

1. The *k*NN-LM variants of LMs do not have any major differences regarding performance on exposure bias and self-recovery metrics compared to their standard counterparts under most circumstances.

2. The exception to this is if the interpolation weight (λ) hyperparameter is tuned only on the basis of minimizing perplexity.

3. All evaluated LMs, regardless of configuration, lack consistent self-recovery abilities.

We believe these findings can guide future evaluations of interpolation-based LMs, especially due to shedding light on the significance of a common interpolation hyperparameter (λ).

2 Related Work

2.1 Interpolation-Based LMs

While Wang et al. (2023) coined the term, to our knowledge, interpolation-based LMs began as a result of Grave et al. (2016) and Grave et al. (2017)'s work, wherein a cache containing past hidden representations was linearly interpolated with a neural LM's distribution. *k*NN-LM (Khandelwal et al., 2020b) utilized Grave et al. (2017)'s concept of linear interpolation to interpolate a neural LM with a *k*NN datastore. TRIME$_{ext}$ (Zhong et al., 2022) further expands on *k*NN-LM by utilizing in-batch memories during pretraining for the base LM. NEST (Li et al., 2024) improves the inference speed and fluency of *k*NN-LM through various techniques, including dynamic span selection and speculative decoding. Infini-gram (Liu et al., 2024b) combines a neural LM with a large datastore of unbounded *n*-grams, achieving significant perplexity reductions but faces challenges in open-ended text generation. Our work evaluates *k*NN-LM as a baseline, as these previous works (including *k*NN-LM) utilize Grave et al. (2017)'s concept of linear interpolation with an external datastore.

2.2 Evaluation & Analysis of *k*NN-LM

*k*NN-LM and its descendants remain popular as interpolation-based LMs and have been adapted to various downstream tasks. However, work specifically dedicated to evaluating *k*NN-LM and its behavior on these downstream tasks remains sparse. Xu et al. (2023) evaluates as to why *k*NN-LM decreases perplexity, noting the importance of *k*NN-LM's use of different input representations, approximate *k*NN search, and careful tuning of the retriever distribution's softmax temperature. BehnamGhader et al. (2022) evaluates several retrieval-augmented LMs (including *k*NN-LM) to investigate the reasons for their strengths and weaknesses in the task of reasoning, finding that the *k*NN-LM's retriever fails to retrieve statements crucial for reasoning due to their dissimilarity to *k*NN-LM's queries. Similarly, Geng et al. (2024) shows *k*NN-LM excels in memory-intensive tasks

but struggles with reasoning tasks, even with perfect retrieval. In our work, we evaluate the *k*NN-LM retriever's effect on exposure bias and self-recovery to identify if it is the reason for *k*NN-LM's failure to improve open-ended text generation per Wang et al. (2023).

3 Background and Notations

Language Models Language models (LM) gather the probability of the next token/word w_t based on the input sequence q_t (i.e. the query vector, or otherwise the context) $P(w_t|q_t)$ and apply decoding methods such as greedy decoding or nucleus sampling (Holtzman et al., 2019) to generate text. The next token probability can also be rewritten as $P(W_{l+1}|\, C, W_{1:l})$, where C is a prompt of fixed size and $W_i \in V$ is a discrete random variable distributed across the vocabulary V. In other words, W is a sentence in the Vocabulary V, $W_{1:l}$ is a prefix, and W_{l+1} is the next token. While equivalent, the former notation is more general and describes the base behavior of an LM in Wang et al. (2023)'s definition of *k*NN-LM. The latter definition is used to measure exposure bias in related metrics in He et al. (2021b)'s work, particularly as the prefix $W_{1:l}$ can be either sampled from the ground truth prefix distribution P_D or model prefix distribution P_M. Note for brevity in notation, these prefix distributions are often included in a prefix distribution set such that $P_H \in \{P_M, P_D\}$.

***k*NN-LM** We closely follow Wang et al. (2023)'s definition of *k*NN-LM with the addition of discussing the retriever's softmax temperature. *k*NN-LM (Khandelwal et al., 2020b) is an interpolation-based LM that interpolates the next token probability distributions between a neural LM model and a retriever to a *k*NN datastore. This datastore holds token-level representations of a corresponding dataset (in this work, the token-level training set of Wikitext-103) through key-value pairs. Each key is a vector k_i representing the context ($n-1$ preceding tokens) for each value v_i which stores the n-th word. During inference, the retriever's next token probability distribution $P_{KNN}(w_t|q_t)$ is calculated by taking the distances of a query vector q_j and all the keys in the datastore $k_1, k_2, \ldots k_{|V|}$ by utilizing a distance function $d(k, q_j)$ (we use Euclidean distance):

$$P_{KNN}(w_t|q_t) \propto \sum_{(k_i,v_i)} \mathbb{1}_{w_t=v_i} \exp(-d(k_i, q_t)) \tag{1}$$

Additionally, the uniformity of $P_{KNN}(w_t|q_t)$ can be controlled by a tunable hyperparameter, τ_{KNN}, otherwise known as the retriever's softmax temperature. After a softmax is applied to gather $P_{KNN}(w_t|q_t)$, the model then interpolates it with the base neural LM's next token probability distribution $P_{LM}(w_t|q_t)$ using a tunable hyperparameter λ to control the relative weight between the two distributions:

$$P^{'}(w_t|q_t) = \lambda P_{KNN}(w_t|q_t) + (1-\lambda) P_{LM}(w_t|q_t) \tag{2}$$

To generate text from the *k*NN-LM distribution $P^{'}(w_t|q_t)$, one can apply a decoding method similar to a standard LM, such as nucleus sampling (Holtzman et al., 2019) or greedy decoding.

3.1 Exposure Bias Metrics

Exposure Bias Marginal (EB-M): EB-M quantifies exposure bias by measuring the relative performance gain in generated text quality when swapping model prefixes for ground truth prefixes. Exposure Bias Marginal (EB-M), like all exposure bias and self-recovery metrics, originates from He et al. (2021b). The generation process for EB-M is defined in three steps: generations are produced by first gathering a prompt of fixed-size C from the ground truth distribution P_D. Then both the ground truth and model prefixes are sampled from $P_H(\cdot \mid C)$, each being of length l (the prefix size). Finally, the generations are created by sampling $W_{l+1:l+l_{\text{gen}}}$ from $P_M(\cdot \mid C, W_{1:l})$, where l_{gen} is the length of generation. EB-M is defined as:

$$\text{EB-M}(M, l, f_{\text{score}}) = \frac{f_{\text{score}}(P_{M|D}^{W_{l+1:l+l_{\text{gen}}}}, P_D^{W_{l+1:l+l_{\text{gen}}}})}{f_{\text{score}}(P_{M|M}^{W_{l+1:l+l_{\text{gen}}}}, P_D^{W_{l+1:l+l_{\text{gen}}}})}. \tag{3}$$

Where f_{score} denotes a scoring function to measure the quality or diversity of the outputs given a label. EB-M focused originally on the marginal distribution of $W_{l+1:l+l_{\text{gen}}}$, $P_{M|H}^{W_{l+1:l+l_{\text{gen}}}}$ (and observing the difference in performance when P_D was used as H as opposed to P_M). In our work, we use the term "marginal" loosely. We do so as our choice for f_{score}, BERTScore (Zhang et al., 2019) is embedding-based, meaning it can emulate the ground truth distribution P_D, making it not necessarily confined to a marginal distribution.

Exposure Bias Consistency (EB-C): In our work, EB-C measures the loss in generated text consistency when swapping ground truth prefixes for

model prefixes. Exposure Bias Consistency (EB-C) is a similar metric to EB-M, sharing many components, such as fixed prompt and prefix, yet focuses on the generation of 1 token instead. This is to curtail artificially induced exposure bias when scoring generations, a problem that He et al. (2021b) notes might affect EB-M. Before defining EB-C, He et al. (2021b) first defines a metric called the *Conditional Generation Deviation* (CGD) to measure the divergence between the two generations, acting as the backbone of EB-C:

$$\text{CGD}(M|H(l), f_{\text{div}}) = \mathop{\mathbb{E}}_{C\sim P_D, W_{1:l}\sim P_H(\cdot|C)} [f_{\text{div}}(P_M(\cdot|C, W_{1:l}), P_D(\cdot|C, W_{1:l}))]. \quad (4)$$

Where f_{div} is a divergence function between two probability distributions $\mathcal{P}$ on the vocabulary V, defined as: $f_{\text{div}} : \mathcal{P} \times \mathcal{P} \rightarrow \mathbb{R}_{\geq 0}$. EB-C is then defined as:

$$\text{EB-C}(M, l, f_{\text{div}}) = \frac{\text{CGD}(M|M(l), f_{\text{div}})}{\text{CGD}(M|D(l), f_{\text{div}})}. \quad (5)$$

Similar to EB-M, EB-C is substantially changed, as the limitation of CGD requiring the ground truth distribution P_D to calculate f_{div} becomes a non-issue. This is because our choice for f_{div}, GPT-2 MAUVE (Liu et al., 2021; Pillutla et al., 2021, 2023), uses the embeddings of GPT-2 to calculate divergence (i.e. Kullback–Leibler Divergence). Hence, unlike He et al. (2021b), our evaluation does not need a synthetic setting to calculate EB-C.

3.2 Self Recovery Metrics

Exposure Bias Marginal (EB-M$_{gap}$) EB-M$_{gap}$ measures the LM's ability to self-recover generated text quality after distortions in its prefix. Specifically, we corrupt (randomize) a percentage of tokens in the ground truth prefix (P_D) of length l to induce this distortion and generate a fixed model prefix of length l_{gap} conditioned on the corrupted prefix to allow the LM to attempt to recover. Then, a generation of length l_{gen} conditioned on a combination of the corrupted prefix and the model prefix (or "gapped" tokens) is produced. Finally, this generation is compared against the generated tokens conditioned on uncorrupted ground truth tokens of length $l + l_{\text{gap}}$. Formally, EB-M$_{gap}$ is given as:

$$\text{EB-M}_{\text{gap}}(M(l_{\text{gap}})|D_{\text{corrupt}}(l), f_{\text{score}}) = \frac{f_{\text{score}}(P_{M|D(l')}^{W_{l'+1:l'+l_{\text{gen}}}}, P_D^{W_{l'+1:l'+l_{\text{gen}}}})}{f_{\text{score}}(P_{M|D_{\text{corrupt}}(l)}^{W_{l'+1:l'+l_{\text{gen}}}}, P_D^{W_{l'+1:l'+l_{\text{gen}}}})}, \quad (6)$$

Where $l' = l + l_{\text{gap}}$ represents the combination of the corrupted prefix length and the model prefix/"gapped" tokens length. Unlike EB-M, we exclude the prompt C for conditioning, hence the scored generations (corrupted and non-corrupted) are instead conditioned on the two following prefixes respectively:

$$P_M(\cdot|W_{1:l}^{\text{corrupt}}, W_{l+1:l+l_{\text{gap}}}), P_D(\cdot|W_{1:l}, W_{l+1:l+l_{\text{gap}}}) \quad (7)$$

We specifically exclude C to be consistent with He et al. (2021b)'s implementation of EB-C$_{gap}$, where it is excluded for space. Our rationale is that treating EB-C$_{gap}$ and EB-M$_{gap}$ differently (beyond the differences of their original counterparts) may produce discrepancies in their results and thus limit their comparability.

Exposure Bias Consistency (EB-C$_{gap}$) EB-C$_{gap}$ measures an LM's ability to self-recover generated text consistency after distortions in its prefix. EB-C$_{gap}$ has many of the same modifications as EB-M$_{gap}$. This includes the exclusion of the prompt, and generation of "gap" tokens to measure self-recovery, alongside the corruption of prefixes. Similarly, the method of calculating CGD has changed with the introduction of l_{gap}:

$$\text{CGD}_{\text{gap}}(M(l_{\text{gap}})|D_{\text{corrupt}}(l)) = \mathop{\mathbb{E}}_{W_{1:l}^{\text{corrupt}}\sim P_D^{\text{corrupt}}, W_{l+1:l+l_{\text{gap}}}\sim P_M(\cdot|W_{1:l}^{\text{corrupt}})} [f_{\text{div}}(P_M(\cdot|W_{1:l}^{\text{corrupt}}, W_{l+1:l+l_{\text{gap}}}), P_D(\cdot|W_{1:l}^{\text{corrupt}}, W_{l+1:l+l_{\text{gap}}}))], \quad (8)$$

Likewise, it follows that the definition of EB-C $_{gap}$ is altered from EB-C:

$$\text{EB-C}_{\text{gap}}(M(l_{\text{gap}})|D_{\text{corrupt}}(l), f_{\text{div}}) = \frac{\text{CGD}_{\text{gap}}(M(l_{\text{gap}})|D_{\text{corrupt}}(l), f_{\text{div}})}{\text{CGD}(M|D(l+l_{\text{gap}}), f_{\text{div}})}. \quad (9)$$

4 Experimental Setup

Utilizing the previously defined exposure bias and self-recovery metrics, we evaluate three different models and their various configurations: GPT-2 (Radford et al., 2019; Alon et al., 2022), Transformer-XL (Dai, 2019), and Llama 3 (Dubey et al., 2024) on the test set of the Wikitext-103 dataset (Merity et al., 2016). Specifically, we use GPT-2 MAUVE (Pillutla et al., 2021; Liu et al., 2021; Pillutla et al., 2023) and BERTScore (Zhang et al., 2019) as scoring and divergence functions to do so. We aim to determine if the *k*NN-LM retriever introduces exposure bias and impedes self-recovery in these models.

4.1 Models and KNN-LM Hyperparameters

We utilize three models in our analysis: GPT-2 (Radford et al., 2019; Alon et al., 2022), Transformer-XL (Dai, 2019), and Llama 3 (Dubey et al., 2024). We chose these models to investigate if *k*NN-LM's effects on exposure bias and self-recovery are different across several factors, such as parameter count and architecture. We leveraged existing data stores where applicable, creating custom datastores for Transformer XL and Llama 3 out of necessity. For these created datastores, we chose two sets of hyperparameters: one based solely on minimizing perplexity, and another based on historical best values in previous works. To validate hyperparameters, we opted to follow a similar strategy to Geng et al. (2024), where we performed a grid search for λ and τ_{KNN} to minimize perplexity. Specifically, we searched $\lambda \in \{0.1, 0.15, 0.2, 0.25, 0.3\}$ and $\tau_{KNN} \in \{1, 3, 5, 7, 9, 10\}$. For alternate hyperparameters, we chose a λ defined in previous work and still chose a τ_{KNN} based on perplexity due to Xu et al. (2023) stressing the heavy importance of a well-tuned retriever softmax temperature.

GPT-2 We utilize the finetuned 124M parameter `gpt2-finetuned-wikitext103` (Radford et al., 2019; Alon et al., 2022) largely as it was the main model utilized in Wang et al. (2023). For its *k*NN-LM configuration, we utilize the datastore and hyperparameters as provided by Alon et al. (2022). Specifically, we set λ to 0.25 and τ_{KNN} to 1. We set k to 1024 for GPT-2's *k*NN-LM variant.

Transformer-XL We utilize the 247M parameter `transfo-xl-wt103` (Dai, 2019) largely due to sharing a common ancestor between the underlying models of the original *k*NN-LM implementation (Khandelwal et al., 2020b) and TRIME$_{ext}$ (Zhong et al., 2022): the Adaptive Input Transformer (Baevski and Auli, 2018). While there are still several differences (e.g. Transformer XL's recurrence and the underlying libraries), we feel the architectural similarities make the models and implementations largely comparable. In the creation of the *k*NN-LM datastore for Transformer XL, we set its recurrence parameter `mem_len` to 384, and during evaluation set it to 640. For hyperparameters, we set λ to 0.10 and τ_{KNN} to 10. For its set of alternative hyperparameters, we set λ to 0.25 and τ_{KNN} to 10 following Khandelwal et al. (2020b). We set k to 1024 for all *k*NN-LM variants.

Llama 3 We utilize the 8B parameter `Meta-Llama-3-8B` (Dubey et al., 2024) for its parameter size, alongside access to a scaled datastore from Shi et al. (2022); Geng et al. (2024), which allows us to observe if scaling the *k*NN-LM datastore and model size affects exposure bias and self-recovery. For hyperparameters, we set λ to 0.3 and τ_{KNN} to 5. For its set of alternate hyperparameters, we set λ to 0.1 and τ_{KNN} to 5 (incidentally the same as the extended datastore hyperparameters), following Geng et al. (2024). We set k to 2048 for all *k*NN-LM variants.

4.2 Exposure Bias and Self Recovery Metric Settings

For all Exposure Bias and Self Recovery Metrics, we set the prompt C and length l_{gen} to 20, following He et al. (2021b). Note that in EB-C and EB-C$_{gap}$, l_{gen} is always 1 following Sections 3.1 and 3.2. For Exposure Bias related metrics, we use five different prefix sizes: 20, 40, 60, 80, and 100. For self-recovery metrics, we use the prefix sizes of 20, 60, and 100 and gap sizes of 0 (or otherwise no gap), 30, and 60. Prefix and gap sizes were selected based on those used in He et al. (2021b) or for being within comparable ranges to said values. For EB-M$_{gap}$ and EB-C$_{gap}$, we set the corruption rate to 30% following He et al. (2021b).

4.3 Dataset and Generation Details

Following Wang et al. (2023), we use the test set of `wikitext-103-raw-v1` (Merity et al., 2016) in our evaluation. To accurately measure exposure bias and self-recovery, we divided the dataset into chunks, extracting the necessary components (e.g. the prompt, ground truth prefix) from each. For EB-M and EB-C, we split the dataset into chunks of size C+l+l_{gen} and further split each chunk by using the first C+l tokens as a *prefix* for which the model will create a continuation of l_{gen} tokens. As all of our metrics require reference text (i.e. the ground truth data), we use the tokens proceeding the prefix of length l_{gen} in each chunk as the references. EB-M $_{gap}$ and EB-C $_{gap}$ split the dataset into chunks of size C+l+l_{gap}+l_{gen}, with the prefix becoming l+l_{gap}, as the prompt is excluded. We keep the prompt in the split to attempt to accurately mimic EB-M $_{gap}$ and EB-C $_{gap}$ as described in He et al. (2021b). Due to the many combinations of prefix size and gap size, we defer the total number of samples of each configuration to Appendix C. Regardless of LM or configuration, any dataset

split conforms to the minimum 1000 sample requirement for GPT-2 MAUVE (Liu et al., 2021; Pillutla et al., 2021, 2023), which is our choice for a f_{score} function. For decoding text, we largely follow Wang et al. (2023), using nucleus sampling where $p = 0.8$ as the primary method to generate text. We also investigate other decoding methods used by both He et al. (2021b) and Wang et al. (2023) in Appendix D.

4.4 Choice of Scoring and Divergence Functions

To measure generated text quality, EB-M and EB-M$_{gap}$ require a choice of corresponding scoring function f_{score}, and similarly, EB-C and EB-C$_{gap}$ require a corresponding divergence function f_{div} to measure generated text consistency. Since these metrics highly rely on their respective functions to do so on relatively short generations (1 or 20 tokens), we carefully chose the following functions:

GPT-2 MAUVE GPT-2 MAUVE (Pillutla et al., 2021; Liu et al., 2021; Pillutla et al., 2023) is an open-ended text generation metric which uses the embeddings of the 774M parameter `gpt2-large` model (Radford et al., 2019) to calculate the similarity between neural and human text via Kullback–Leibler Divergences. Its high levels of correlation with human judgment, alongside its extensive use as an open-ended text generation metric in Wang et al. (2023)'s work make it an ideal candidate for an f_{div} function.

BERTScore-F1 BERTScore (Zhang et al., 2019) is a text generation metric that utilizes BERT (Devlin, 2018) embeddings and cosine similarity to compute the precision, recall, and F1 score between a neural and human text. It correlates higher with human judgment than BLEU (Papineni et al., 2002) and ROUGE (Lin, 2004), which measure n-gram precision and recall between neural and human texts. He et al. (2021b) previously used BLEU a an f_{score}. As BERTScore improves upon a prior f_{score}, we chose BERTScore-F1 as an f_{score} function. We use the 750M parameter `deberta-xlarge-mnli` (He et al., 2021a) model as the BERTScore evaluator, as recommended by the authors[1].

5 Results

Despite the hypotheses made in Wang et al. (2023), we find that the *k*NN-LM retriever does not appear to introduce exposure bias nor impede an LM's ability to self-recover in most cases. However, we find a large exception to these results: when the interpolation weight (λ) hyperparameter is tuned only to minimize perplexity, specifically when adding *k*NN-LM to Llama 3. We also find that all evaluated LMs, regardless of configuration, cannot self-recover consistently. We present our results in tabular form in Appendix A and an extended evaluation for further verification of results in Appendix D.

5.1 Exposure Bias Metrics

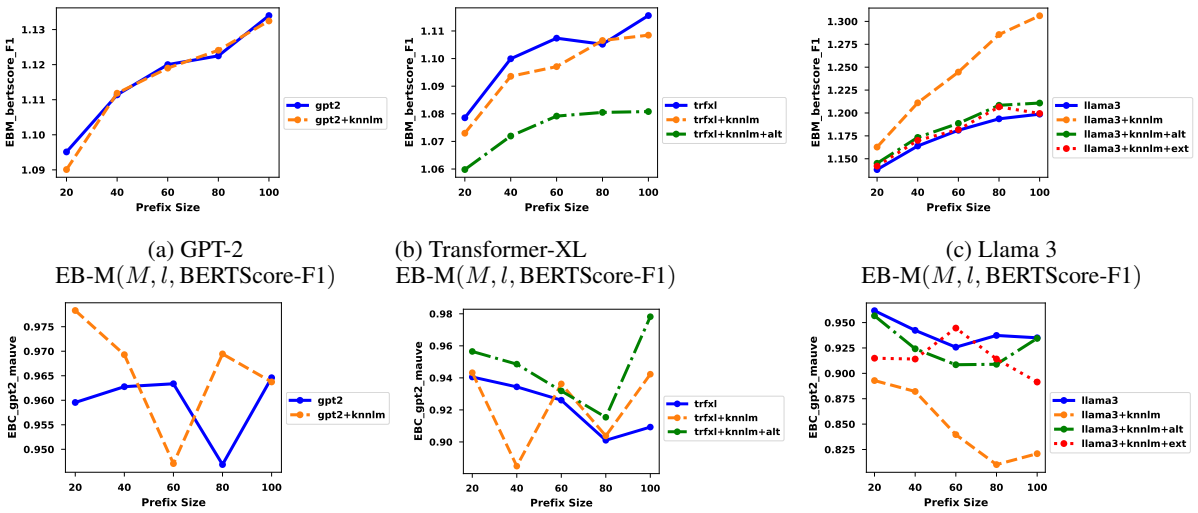
No Major Changes in Exposure Bias for Most *k*NN-LM Configurations As shown in Figure 1, there appears to be no major difference in the amount of exposure bias between interpolated and non-interpolated models for most configurations. While there are some points in which the *k*NN-LM variants may have a higher disparity for certain prefix sizes, such as in Subfigures 1d and 1e, this is not consistent trend-wise. On the contrary, we even see some evidence where the Transformer-XL *k*NN-LM variant with alternate hyperparameters(trfxl+knn+alt) consistently outperformed the base LM (trfxl) in subfigure 1b to a minor degree.

Tuning the Interpolation Weight (λ) Hyperparameter Solely to Minimize Perplexity Can Introduce Exposure Bias In both Subfigures 1c and 1f, we see that under the Llama *k*NN-LM configuration tuned solely for perplexity (llam3+knnlm) the disparities in quality and consistency greatly increased as the prefix size increased, indicating increased amounts of exposure bias. At a prefix size of 100, we see this configuration has a ~10% increase in BERTScore disparity and a ~12% increase in disparity for GPT-2 MAUVE compared to its non *k*NN-LM counterpart (llama3). This is also complemented by a much less severe case: in both Subfigures 1b and 1e, the alternate hyperparameter variant of *k*NN-LM (trfxl+knnlm+alt) tends to outperform its non-alternate counterpart (trfxl+knnlm), but does not necessarily worse than the stock LM (trfxl). This suggests that tuning λ may be more important than previously thought.

5.2 Self Recovery Metrics

All LMs and Configurations Lack Consistent Self-Recovery Abilities Figure 2 demonstrates evaluated LMs' lack of consistent self-recovery abilities across all subfigures. Any supposed recovery with smaller prefix sizes begins to diminish

[1] https://github.com/Tiiiger/bert_score



(a) GPT-2 EB-M(M, l, BERTScore-F1)

(b) Transformer-XL EB-M(M, l, BERTScore-F1)

(c) Llama 3 EB-M(M, l, BERTScore-F1)

(d) GPT-2 EB-C(M, l, GPT-2 MAUVE)

(e) Transformer-XL EB-C(M, l, GPT-2 MAUVE)

(f) Llama 3 EB-C(M, l, GPT-2 MAUVE)

Figure 1: EBM/EBC Ratios for several LM configurations detailing the disparities (exposure bias) in quality (via BERTScore) and consistency (via GPT-2 MAUVE) between generated text conditioned on model and ground truth prefixes of increasing size. Additional configurations include **alt**ernate kNN-LM hyperparamers and an **ext**ended datastore. **A ratio farther from 1 indicates more discrepancy**, i.e. exposure bias, **regardless of the exact ratio value**. In theory, all of the kNN-LM variants should see increased exposure bias compared to their standard LM counterparts. Yet in practice, they typically exhibit equal or lower levels of exposure bias (llama3+knnlm being the major exception).

with larger prefix sizes or simply does not occur at all. We see that in Subfigure 2f that the non kNN-LM variant of Llama 3 (llama3) can recover under a prefix of size 20, it cannot do so for a prefix size of 60 and 100, as initial gains in ratios begin to subside with a larger gap size. Worse, Subfigures 2a, 2b, and 2c see sharp disparity increases with no sort of improvement in sight.

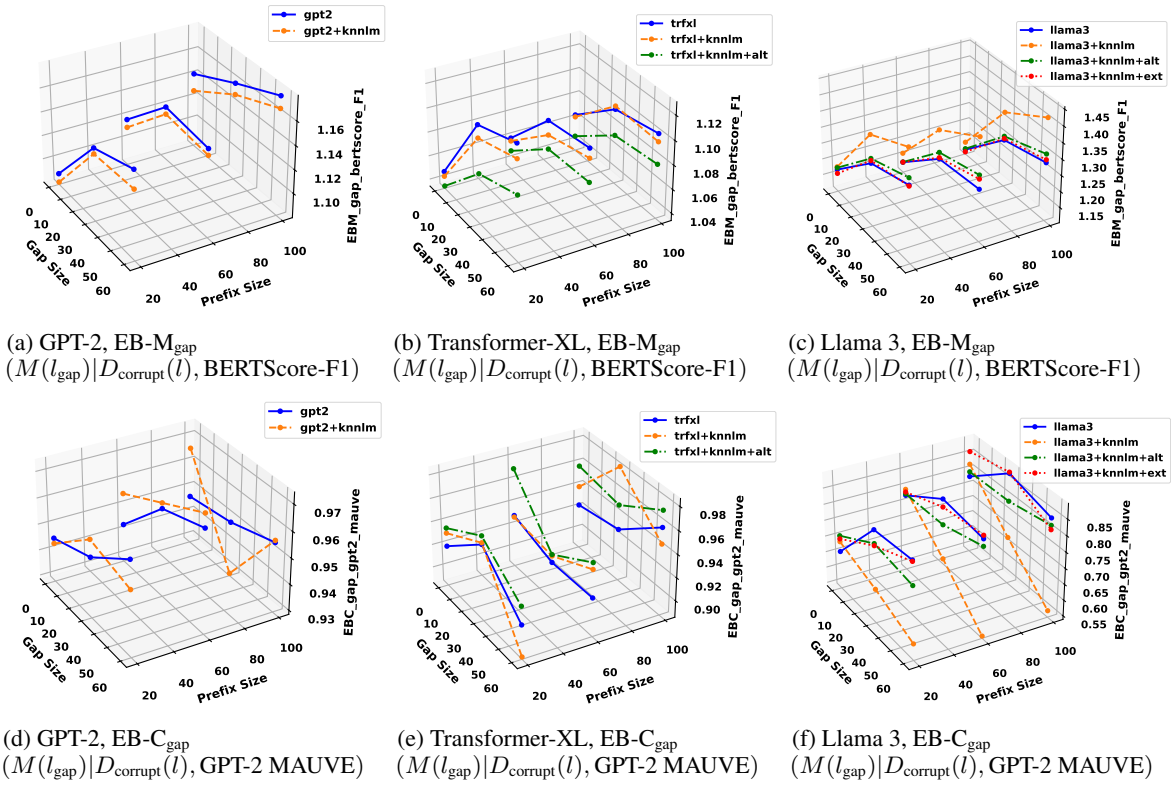
***k*NN-LM Generally Induces No Major Changes to Self-Recovery Disparities** Despite lacking self-recovery abilities, the evaluated LMs generally show no major changes regarding self-recovery disparities in their kNN-LM variants. Interestingly, these results are similar to those seen in Section 5.1. For example, the alternate kNN-LM hyperparameter variant (trfxl+knnlm+alt) of Transformer-XL still has consistent levels of reducing disparities in Subfigure 2b. Similarly, the Llama 3 kNN-LM variant (llama3+knnlm) is still an exception, greatly hampering performance relative to its standard LM counterpart (llama) in Subfigures 2c and 2f. However, the disparities in this exception are far worse: when the prefix size is 100, the difference in disparity ratios can reach as high as **~30%**. We believe the reason for this is the same: the choice of a λ based on the minimization of perplexity alone.

6 Discussion

In this section, we briefly speculate the reasons for our findings and their potential implications for future work regarding interpolation-based LMs.

6.1 Why Does The Interpolation Weight (λ) Hyperparameter Influence Exposure Bias and Self-Recovery so Heavily?

Coincidentally, we do not have to look far for a potential answer. Wang et al. (2023) claims that a λ tuned for perplexity may not produce the best MAUVE values. We agree, as our results showed that a λ tuned solely for perplexity consistently performed worse than its alternative hyperparameters. Digging deeper, we hypothesize this is due to perplexity's weakness: perplexity values tend to get lower with more repetitive tokens (Wang et al., 2022). Our rationale for this hypothesis is based on two previous findings: that kNN-LM only benefits a subset of tokens (Wang et al., 2023) and that kNN-LM tends to retrieve high-frequency entities (Geng et al., 2024). Since perplexity is potentially lowered due to the repetition of tokens (Wang et al., 2022), we hypothesize that tuning λ for perplexity alone can exacerbate kNN-LM's bias towards a minority of frequent tokens, leading to a cascading effect. We also discuss some early evidence to



(a) GPT-2, EB-M_{gap} $(M(l_{\text{gap}})|D_{\text{corrupt}}(l)$, BERTScore-F1)

(b) Transformer-XL, EB-M_{gap} $(M(l_{\text{gap}})|D_{\text{corrupt}}(l)$, BERTScore-F1)

(c) Llama 3, EB-M_{gap} $(M(l_{\text{gap}})|D_{\text{corrupt}}(l)$, BERTScore-F1)

(d) GPT-2, EB-C_{gap} $(M(l_{\text{gap}})|D_{\text{corrupt}}(l)$, GPT-2 MAUVE)

(e) Transformer-XL, EB-C_{gap} $(M(l_{\text{gap}})|D_{\text{corrupt}}(l)$, GPT-2 MAUVE)

(f) Llama 3, EB-C_{gap} $(M(l_{\text{gap}})|D_{\text{corrupt}}(l)$, GPT-2 MAUVE)

Figure 2: EBC/EBM (GAP) ratios for various LM configurations detailing the disparities in quality (via BERTScore) and consistency (via GPT-2 MAUVE) between generated text conditioned a ground truth prefix and its 30% corrupted version. Additional configurations include **alt**ernate kNN-LM hyperparamers and an **ext**ended datastore. Given an increasing model prefix of size "gap", the model should, in theory, "self-recover" by **bringing the ratio closer to 1**, indicating no discrepancy, **regardless of the exact ratio value**. Similarly, all kNN-LM configurations should in theory hamper this ability compared to their standard LM counterparts. In practice, no LM appears to be able to "self-recover" consistently. Yet, despite this, the kNN-LM variants generally perform similarly to their standard counterparts (with llama3+knnlm being the major exception).

support this hypothesis in Appendix B.

6.2 Do LMs Lack Self-Recovery?

We believe this question lacks a definitive answer, as to our knowledge, only one method to quantify self-recovery exists in the form of He et al. (2021b)'s metrics. However, we can turn towards a related concept for closed-end tasks in the form of self-correction, where models revise responses through intrinsic refinement or external feedback. Whether language models (LMs) possess this ability is still debated (Huang et al., 2023; Liu et al., 2024a; Kamoi et al., 2024; Wu et al., 2024; Zhang et al., 2024b). Given doubts about self-correction, which requires prompting, we also question whether LMs can self-recover since self-recovery theoretically occurs without prompting. If LMs potentially cannot self-recover, then Wang et al. 2023's claim that the kNN-LM retriever impedes this ability may rest on a false assumption. However, with only one definition of self-recovery, further evaluation is needed to confirm if this is the case.

7 Conclusion

In this work, we explored whether the kNN-LM retriever introduces exposure bias and impedes self-recovery in various LMs and configurations. Our findings are threefold: 1) The kNN-LM retriever generally does not introduce major differences regarding exposure bias and self-recovery. 2) The exception to this is when the interpolation weight (λ) hyperparameter of kNN-LM is tuned solely to minimize perplexity. 3) All LMs and configurations tested appear to lack consistent self-recovery abilities. We anticipate that our results will contribute to further evaluation of interpolation-based LMs.

8 Limitations

Our work only uses Alon et al. (2022)'s implementation of kNN-LM due to compatibility issues with other implementations. Namely, Khandelwal et al. (2020b) and Zhong et al. (2022) both utilize forks of fairseq [2][3] for their implementations, which lack compatibility with larger and more recent LMs such as Llama 3 (Dubey et al., 2024) (which was used in our evaluation) due to fairseq's age[4].

This work also does not conduct a human evaluation due to the high costs associated with conducting an evaluation to the same scale as He et al. (2021b). While exposure bias has typically been studied quantitatively as: "It would be difficult to judge whether the distortions are incremental via qualitative examination" (He et al., 2021b), future work can potentially use qualitative human evaluation as a means of verifying quantitative results regarding whether the kNN-LM or similar (i.e. interpolation-based) retrievers introduce exposure bias.

References

Uri Alon, Frank Xu, Junxian He, Sudipta Sengupta, Dan Roth, and Graham Neubig. 2022. Neuro-symbolic language modeling with automaton-augmented retrieval. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 468–485. PMLR.

Alexei Baevski and Michael Auli. 2018. Adaptive input representations for neural language modeling. *arXiv preprint arXiv:1809.10853*.

Parishad BehnamGhader, Santiago Miret, and Siva Reddy. 2022. Can retriever-augmented language models reason? the blame game between the retriever and the language model. *arXiv preprint arXiv:2212.09146*.

Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. 2015. Scheduled sampling for sequence prediction with recurrent neural networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'15, page 1171–1179, Cambridge, MA, USA. MIT Press.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Ting-Rui Chiang and Yun-Nung Chen. 2021. Relating neural text degeneration to exposure bias. In *Proceedings of the Fourth BlackboxNLP Workshop on Analyzing and Interpreting Neural Networks for NLP*, pages 228–239, Punta Cana, Dominican Republic. Association for Computational Linguistics.

Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. 2020. ELECTRA: Pre-training text encoders as discriminators rather than generators. In *ICLR*.

Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Unsupervised cross-lingual representation learning at scale. *CoRR*, abs/1911.02116.

Zihang Dai. 2019. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*.

Leon Derczynski. 2016. Complementarity, F-score, and NLP evaluation. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 261–266, Portorož, Slovenia. European Language Resources Association (ELRA).

Jacob Devlin. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Angela Fan, Mike Lewis, and Yann Dauphin. 2018. Hierarchical neural story generation. *arXiv preprint arXiv:1805.04833*.

Zihao Fu, Wai Lam, Anthony Man-Cho So, and Bei Shi. 2021. A theoretical analysis of the repetition problem in text generation. In *Thirty-Fifth AAAI Conference on Artificial Intelligence*.

Shangyi Geng, Wenting Zhao, and Alexander M Rush. 2024. Great memory, shallow reasoning: Limits of k nn-lms. *arXiv preprint arXiv:2408.11815*.

Edouard Grave, Moustapha M Cisse, and Armand Joulin. 2017. Unbounded cache model for online language modeling with open vocabulary. *Advances in neural information processing systems*, 30.

Edouard Grave, Armand Joulin, and Nicolas Usunier. 2016. Improving neural language models with a continuous cache. *arXiv preprint arXiv:1612.04426*.

Nicholas Harris, Anand Butani, and Syed Hashmy. 2024. Enhancing embedding performance through large language model-based text enrichment and rewriting. *arXiv preprint arXiv:2404.12283*.

[2] https://github.com/urvashik/knnlm
[3] https://github.com/princeton-nlp/TRIME
[4] https://github.com/facebookresearch/fairseq/issues/4493

Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. 2021a. Deberta: Decoding-enhanced bert with disentangled attention. In *International Conference on Learning Representations*.

Tianxing He, Jingyu Zhang, Tianle Wang, Sachin Kumar, Kyunghyun Cho, James Glass, and Yulia Tsvetkov. 2023. On the blind spots of model-based evaluation metrics for text generation. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12067–12097, Toronto, Canada. Association for Computational Linguistics.

Tianxing He, Jingzhao Zhang, Zhiming Zhou, and James Glass. 2021b. Exposure bias versus self-recovery: Are distortions really incremental for autoregressive text generation? In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 5087–5102, 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. *arXiv preprint arXiv:1904.09751*.

Ari Holtzman, Jan Buys, Maxwell Forbes, Antoine Bosselut, David Golub, and Yejin Choi. 2018. Learning to write with cooperative discriminators. *arXiv preprint arXiv:1805.06087*.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. 2023. Large language models cannot self-correct reasoning yet. *arXiv preprint arXiv:2310.01798*.

Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2019. Billion-scale similarity search with gpus. *IEEE Transactions on Big Data*, 7(3):535–547.

Ryo Kamoi, Yusen Zhang, Nan Zhang, Jiawei Han, and Rui Zhang. 2024. When can llms actually correct their own mistakes? a critical survey of self-correction of llms. *arXiv preprint arXiv:2406.01297*.

Urvashi Khandelwal, Angela Fan, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. 2020a. Nearest neighbor machine translation. *arXiv preprint arXiv:2010.00710*.

Urvashi Khandelwal, Omer Levy, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. 2020b. Generalization through memorization: Nearest neighbor language models. *Preprint*, arXiv:1911.00172.

Kalpesh Krishna, Yapei Chang, John Wieting, and Mohit Iyyer. 2022. Rankgen: Improving text generation with large ranking models. *arXiv preprint arXiv:2205.09726*.

Nayeon Lee, Wei Ping, Peng Xu, Mostofa Patwary, Pascale N Fung, Mohammad Shoeybi, and Bryan Catanzaro. 2022. Factuality enhanced language models for open-ended text generation. *Advances in Neural Information Processing Systems*, 35:34586–34599.

Huayang Li, Tian Lan, Zihao Fu, Deng Cai, Lemao Liu, Nigel Collier, Taro Watanabe, and Yixuan Su. 2023a. Repetition in repetition out: Towards understanding neural text degeneration from the data perspective. In *Advances in Neural Information Processing Systems*, volume 36, pages 72888–72903. Curran Associates, Inc.

Minghan Li, Xilun Chen, Ari Holtzman, Beidi Chen, Jimmy Lin, Wen-tau Yih, and Xi Victoria Lin. 2024. Nearest neighbor speculative decoding for llm generation and attribution. *arXiv preprint arXiv:2405.19325*.

Zehan Li, Xin Zhang, Yanzhao Zhang, Dingkun Long, Pengjun Xie, and Meishan Zhang. 2023b. Towards general text embeddings with multi-stage contrastive learning. *arXiv preprint arXiv:2308.03281*.

Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*, pages 74–81.

Dancheng Liu, Amir Nassereldine, Ziming Yang, Chenhui Xu, Yuting Hu, Jiajie Li, Utkarsh Kumar, Changjae Lee, and Jinjun Xiong. 2024a. Large language models have intrinsic self-correction ability. *arXiv preprint arXiv:2406.15673*.

Jiacheng Liu, Sewon Min, Luke Zettlemoyer, Yejin Choi, and Hannaneh Hajishirzi. 2024b. Infini-gram: Scaling unbounded n-gram language models to a trillion tokens. *Preprint*, arXiv:2401.17377.

Lang Liu, Krishna Pillutla, Sean Welleck, Sewoong Oh, Yejin Choi, and Zaid Harchaoui. 2021. Divergence Frontiers for Generative Models: Sample Complexity, Quantization Effects, and Frontier Integrals. In *NeurIPS*.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2020. Ro{bert}a: A robustly optimized {bert} pretraining approach.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016. Pointer sentinel mixture models. *Preprint*, arXiv:1609.07843.

Sewon Min, Suchin Gururangan, Eric Wallace, Weijia Shi, Hannaneh Hajishirzi, Noah A Smith, and Luke Zettlemoyer. 2023. Silo language models: Isolating legal risk in a nonparametric datastore. *arXiv preprint arXiv:2308.04430*.

Niklas Muennighoff, Nouamane Tazi, Loïc Magne, and Nils Reimers. 2022. Mteb: Massive text embedding benchmark. *arXiv preprint arXiv:2210.07316*.

Feng Nan, Ramesh Nallapati, Zhiguo Wang, Cicero Nogueira dos Santos, Henghui Zhu, Dejiao Zhang, Kathleen McKeown, and Bing Xiang. 2021. Entity-level factual consistency of abstractive text summarization. *arXiv preprint arXiv:2102.09130*.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318.

Krishna Pillutla, Lang Liu, John Thickstun, Sean Welleck, Swabha Swayamdipta, Rowan Zellers, Sewoong Oh, Yejin Choi, and Zaid Harchaoui. 2023. MAUVE Scores for Generative Models: Theory and Practice. *JMLR*.

Krishna Pillutla, Swabha Swayamdipta, Rowan Zellers, John Thickstun, Sean Welleck, Yejin Choi, and Zaid Harchaoui. 2021. Mauve: Measuring the gap between neural text and human text using divergence frontiers. In *NeurIPS*.

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.

Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. 2015. Sequence level training with recurrent neural networks. *arXiv preprint arXiv:1511.06732*.

Florian Schmidt. 2019. Generalization in generation: A closer look at exposure bias. In *Proceedings of the 3rd Workshop on Neural Generation and Translation*, pages 157–167, Hong Kong. Association for Computational Linguistics.

Claude Elwood Shannon. 1948. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423.

Chufan Shi, Haoran Yang, Deng Cai, Zhisong Zhang, Yifan Wang, Yujiu Yang, and Wai Lam. 2024. A thorough examination of decoding methods in the era of llms. *arXiv preprint arXiv:2402.06925*.

Weijia Shi, Julian Michael, Suchin Gururangan, and Luke Zettlemoyer. 2022. knn-prompt: Nearest neighbor zero-shot inference. *arXiv preprint arXiv:2205.13792*.

Chaojun Wang and Rico Sennrich. 2020. On exposure bias, hallucination and domain shift in neural machine translation. *arXiv preprint arXiv:2005.03642*.

Shufan Wang, Yixiao Song, Andrew Drozdov, Aparna Garimella, Varun Manjunatha, and Mohit Iyyer. 2023. kNN-LM does not improve open-ended text generation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 15023–15037, Singapore. Association for Computational Linguistics.

Yequan Wang, Jiawen Deng, Aixin Sun, and Xuying Meng. 2022. Perplexity from plm is unreliable for evaluating text quality. *arXiv preprint arXiv:2210.05892*.

Sean Welleck, Ilia Kulikov, Stephen Roller, Emily Dinan, Kyunghyun Cho, and Jason Weston. 2019. Neural text generation with unlikelihood training. *arXiv preprint arXiv:1908.04319*.

Zhenyu Wu, Qingkai Zeng, Zhihan Zhang, Zhaoxuan Tan, Chao Shen, and Meng Jiang. 2024. Large language models can self-correct with minimal effort. *arXiv preprint arXiv:2405.14092*.

Frank F. Xu, Uri Alon, and Graham Neubig. 2023. Why do nearest neighbor language models work? In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org.

Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q Weinberger, and Yoav Artzi. 2019. Bertscore: Evaluating text generation with bert. *arXiv preprint arXiv:1904.09675*.

Xin Zhang, Yanzhao Zhang, Dingkun Long, Wen Xie, Ziqi Dai, Jialong Tang, Huan Lin, Baosong Yang, Pengjun Xie, Fei Huang, Meishan Zhang, Wenjie Li, and Min Zhang. 2024a. mgte: Generalized long-context text representation and reranking models for multilingual text retrieval. *Preprint*, arXiv:2407.19669.

Yunxiang Zhang, Muhammad Khalifa, Lajanugen Logeswaran, Jaekyeom Kim, Moontae Lee, Honglak Lee, and Lu Wang. 2024b. Small language models need strong verifiers to self-correct reasoning. *arXiv preprint arXiv:2404.17140*.

Zexuan Zhong, Tao Lei, and Danqi Chen. 2022. Training language models with memory augmentation. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 5657–5673, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

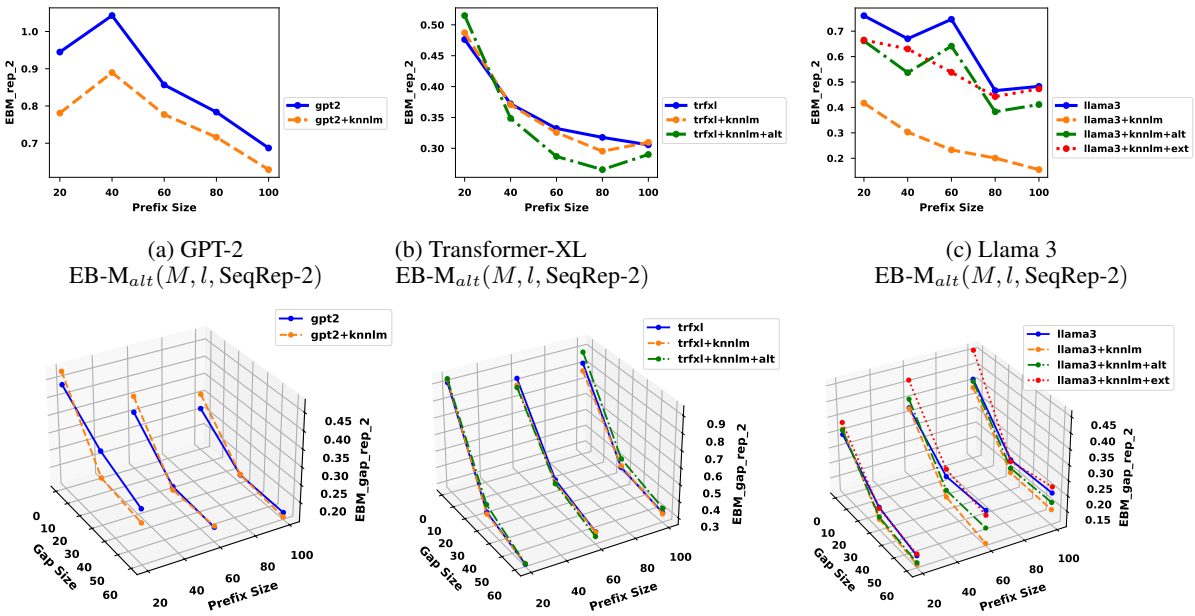
A Results In Tabular Form

EB-M / EB-C	prefix length (l)				
	20	**40**	**60**	**80**	**100**
EB-M (M_{gpt2}, BERTScore-F1)	1.0951	1.1114	1.12	1.1225	1.134
EB-M ($M_{\text{gpt2+knnlm}}$, BERTScore-F1)	1.0901	1.1119	1.119	1.1241	1.1325
EB-C (M_{gpt2}, GPT-2 MAUVE)	0.9595	0.9628	0.9634	0.9469	0.9646
EB-C ($M_{\text{gpt2+knnlm}}$, GPT-2 MAUVE)	0.9783	0.9693	0.9471	0.9694	0.9637
EB-M (M_{trfxl}, BERTScore-F1)	1.0786	1.0999	1.1074	1.1052	1.1156
EB-M ($M_{\text{trfxl+knnlm}}$, BERTScore-F1)	1.073	1.0936	1.0971	1.1065	1.1085
EB-M ($M_{\text{trfxl+knnlm+alt}}$, BERTScore-F1)	1.0598	1.0719	1.0792	1.0805	1.0808
EB-C (M_{trfxl}, GPT-2 MAUVE)	0.9405	0.9344	0.9261	0.901	0.9093
EB-C ($M_{\text{trfxl+knnlm}}$, GPT-2 MAUVE)	0.9432	0.8849	0.9362	0.9038	0.9423
EB-C ($M_{\text{trfxl+knnlm+alt}}$, GPT-2 MAUVE)	0.9565	0.9486	0.9319	0.9154	0.9782
EB-M (M_{llama3}, BERTScore-F1)	1.1382	1.1639	1.1811	1.1936	1.1987
EB-M ($M_{\text{llama3+knnlm}}$, BERTScore-F1)	1.1628	1.2111	1.2446	1.2857	1.3062
EB-M ($M_{\text{llama3+knnlm+alt}}$, BERTScore-F1)	1.145	1.1733	1.1886	1.2084	1.2108
EB-M ($M_{\text{llama3+knnlm+ext}}$, BERTScore-F1)	1.1419	1.1701	1.1818	1.2066	1.1995
EB-C (M_{llama3}, GPT-2 MAUVE)	0.9617	0.9423	0.9258	0.9373	0.935
EB-C ($M_{\text{llama3+knnlm}}$, GPT-2 MAUVE)	0.8929	0.8822	0.8396	0.8101	0.8209
EB-C ($M_{\text{llama3+knnlm+alt}}$, GPT-2 MAUVE)	0.9567	0.9242	0.9085	0.9089	0.9344
EB-C ($M_{\text{llama3+knnlm+ext}}$, GPT-2 MAUVE)	0.9149	0.9141	0.9446	0.9141	0.8914

Table 1: The tabular data of EBM/EBC Ratios for the various LMs and their configurations detailing the disparities (exposure bias) in quality (via BERTScore) and consistency (via GPT-2 MAUVE) between generated text conditioned on model and ground truth prefixes of increasing size. Additional configurations include **alt**ernate kNN-LM hyperparamers and an **ext**ended datastore. **A ratio farther from 1 indicates more discrepancy**, i.e. exposure bias, **regardless of the exact ratio value**. In theory, all of the kNN-LM variants should see increased exposure bias compared to their standard LM counterparts. Yet in practice, they generally exhibit equal or lower levels of exposure bias (with llama3+knnlm being the exception). This table's results are equivalent to Figure 1. The design was largely borrowed from He et al. (2021b) for the sake of consistency.

EB-M$_{gap}$ / EB-C$_{gap}$	gap length (l_{gap})		
	0	**30**	**60**
EB-M$_{gap}$ ($M_{\text{gpt2+20}}$, BERTScore-F1)	1.0888	1.141	1.1541
EB-M$_{gap}$ ($M_{\text{gpt2+knnlm+20}}$, BERTScore-F1)	1.0815	1.1361	1.1384
EB-C$_{gap}$ ($M_{\text{gpt2+20}}$, GPT-2 MAUVE)	0.9454	0.9517	0.9647
EB-C$_{gap}$ ($M_{\text{gpt2+knnlm+20}}$, GPT-2 MAUVE)	0.9433	0.9583	0.9538
EB-M$_{gap}$ ($M_{\text{gpt2+60}}$, BERTScore-F1)	1.1189	1.1577	1.153
EB-M$_{gap}$ ($M_{\text{gpt2+knnlm+60}}$, BERTScore-F1)	1.1121	1.1519	1.1476
EB-C$_{gap}$ ($M_{\text{gpt2+60}}$, GPT-2 MAUVE)	0.9431	0.9622	0.9681
EB-C$_{gap}$ ($M_{\text{gpt2+knnlm+60}}$, GPT-2 MAUVE)	0.9553	0.9643	0.9735
EB-M$_{gap}$ ($M_{\text{gpt2+100}}$, BERTScore-F1)	1.1426	1.1614	1.1786
EB-M$_{gap}$ ($M_{\text{gpt2+knnlm+100}}$, BERTScore-F1)	1.1279	1.152	1.1684
EB-C$_{gap}$ ($M_{\text{gpt2+100}}$, GPT-2 MAUVE)	0.9469	0.9495	0.9551
EB-C$_{gap}$ ($M_{\text{gpt2+knnlm+100}}$, GPT-2 MAUVE)	0.9658	0.9293	0.9559
EB-M$_{gap}$ ($M_{\text{trfxl+20}}$, BERTScore-F1)	1.0483	1.1152	1.1287
EB-M$_{gap}$ ($M_{\text{trfxl+knnlm+20}}$, BERTScore-F1)	1.0442	1.1046	1.1167
EB-M$_{gap}$ ($M_{\text{trfxl+knnlm+alt+20}}$, BERTScore-F1)	1.0358	1.0758	1.0885
EB-C$_{gap}$ ($M_{\text{trfxl+20}}$, GPT-2 MAUVE)	0.9203	0.9513	0.915
EB-C$_{gap}$ ($M_{\text{trfxl+knnlm+20}}$, GPT-2 MAUVE)	0.9318	0.9527	0.8876
EB-C$_{gap}$ ($M_{\text{trfxl+knnlm+alt+20}}$, GPT-2 MAUVE)	0.9362	0.9585	0.9306
EB-M$_{gap}$ ($M_{\text{trfxl+60}}$, BERTScore-F1)	1.0607	1.1029	1.1085
EB-M$_{gap}$ ($M_{\text{trfxl+knnlm+60}}$, BERTScore-F1)	1.0582	1.091	1.1005
EB-M$_{gap}$ ($M_{\text{trfxl+knnlm+alt+60}}$, BERTScore-F1)	1.0496	1.0794	1.0812
EB-C$_{gap}$ ($M_{\text{trfxl+60}}$, GPT-2 MAUVE)	0.9308	0.9187	0.9191
EB-C$_{gap}$ ($M_{\text{trfxl+knnlm+60}}$, GPT-2 MAUVE)	0.9294	0.9243	0.9431
EB-C$_{gap}$ ($M_{\text{trfxl+knnlm+alt+60}}$, GPT-2 MAUVE)	0.9717	0.9256	0.9488
EB-M$_{gap}$ ($M_{\text{trfxl+100}}$, BERTScore-F1)	1.065	1.0965	1.1039
EB-M$_{gap}$ ($M_{\text{trfxl+knnlm+100}}$, BERTScore-F1)	1.0633	1.0992	1.0973
EB-M$_{gap}$ ($M_{\text{trfxl+knnlm+alt+100}}$, BERTScore-F1)	1.0463	1.0747	1.079
EB-C$_{gap}$ ($M_{\text{trfxl+100}}$, GPT-2 MAUVE)	0.9241	0.9305	0.961
EB-C$_{gap}$ ($M_{\text{trfxl+knnlm+100}}$, GPT-2 MAUVE)	0.9406	0.9846	0.9471
EB-C$_{gap}$ ($M_{\text{trfxl+knnlm+alt+100}}$, GPT-2 MAUVE)	0.9588	0.9518	0.9753
EB-M$_{gap}$ ($M_{\text{llama3+20}}$, BERTScore-F1)	1.1724	1.2981	1.3402
EB-M$_{gap}$ ($M_{\text{llama3+knnlm+20}}$, BERTScore-F1)	1.1812	1.3847	1.4506
EB-M$_{gap}$ ($M_{\text{llama3+knnlm+alt+20}}$, BERTScore-F1)	1.1783	1.3127	1.362
EB-M$_{gap}$ ($M_{\text{llama3+knnlm+ext+20}}$, BERTScore-F1)	1.1599	1.3063	1.3373
EB-C$_{gap}$ ($M_{\text{llama3+20}}$, GPT-2 MAUVE)	0.6571	0.8283	0.8428
EB-C$_{gap}$ ($M_{\text{llama3+knnlm+20}}$, GPT-2 MAUVE)	0.6885	0.6452	0.5907
EB-C$_{gap}$ ($M_{\text{llama3+knnlm+alt+20}}$, GPT-2 MAUVE)	0.7075	0.7864	0.7677
EB-C$_{gap}$ ($M_{\text{llama3+knnlm+ext+20}}$, GPT-2 MAUVE)	0.6975	0.7799	0.8387
EB-M$_{gap}$ ($M_{\text{llama3+60}}$, BERTScore-F1)	1.135	1.2508	1.2646
EB-M$_{gap}$ ($M_{\text{llama3+knnlm+60}}$, BERTScore-F1)	1.1644	1.3406	1.421
EB-M$_{gap}$ ($M_{\text{llama3+knnlm+alt+60}}$, BERTScore-F1)	1.1358	1.2712	1.3076
EB-M$_{gap}$ ($M_{\text{llama3+knnlm+ext+60}}$, BERTScore-F1)	1.1352	1.2546	1.2953
EB-C$_{gap}$ ($M_{\text{llama3+60}}$, GPT-2 MAUVE)	0.7782	0.8634	0.8443
EB-C$_{gap}$ ($M_{\text{llama3+knnlm+60}}$, GPT-2 MAUVE)	0.7971	0.6785	0.5459
EB-C$_{gap}$ ($M_{\text{llama3+knnlm+alt+60}}$, GPT-2 MAUVE)	0.783	0.7847	0.822
EB-C$_{gap}$ ($M_{\text{llama3+knnlm+ext+60}}$, GPT-2 MAUVE)	0.7887	0.839	0.8549
EB-M$_{gap}$ ($M_{\text{llama3+100}}$, BERTScore-F1)	1.1165	1.2505	1.2835
EB-M$_{gap}$ ($M_{\text{llama3+knnlm+100}}$, BERTScore-F1)	1.1414	1.3377	1.4195
EB-M$_{gap}$ ($M_{\text{llama3+knnlm+alt+100}}$, BERTScore-F1)	1.1198	1.2625	1.3101
EB-M$_{gap}$ ($M_{\text{llama3+knnlm+ext+100}}$, BERTScore-F1)	1.1094	1.2553	1.2926
EB-C$_{gap}$ ($M_{\text{llama3+100}}$, GPT-2 MAUVE)	0.7835	0.8865	0.8478
EB-C$_{gap}$ ($M_{\text{llama3+knnlm+100}}$, GPT-2 MAUVE)	0.8228	0.6859	0.5595
EB-C$_{gap}$ ($M_{\text{llama3+knnlm+alt+100}}$, GPT-2 MAUVE)	0.7986	0.8006	0.8255
EB-C$_{gap}$ ($M_{\text{llama3+knnlm+ext+100}}$, GPT-2 MAUVE)	0.8628	0.8902	0.8124

Table 2: The tabular data of EBC/EBM (GAP) ratios for various LM configurations detailing the disparities in quality (via BERTScore) and consistency (via GPT-2 MAUVE) between generated text conditioned a ground truth prefix and its 30% corrupted version. Given an increasing model prefix of size "gap", the model should, in theory, "self-recover" by **bringing the ratio closer to 1**, indicating no discrepancy, **regardless of the exact ratio value**. Similarly, all *k*NN-LM configurations should in theory hamper this ability compared to their standard LM counterparts. In practice, no LM appears to be able to "self-recover" consistently, and *k*NN-LM variants still generally perform similarly to their standard counterparts (with llama3+knnlm being the exception). This table's results are equivalent to Figure 2. The design was modified from He et al. (2021b)'s original EBC/EBM table. **Numbers after the configuration name are the prefix size**, (e.g. gpt2+20 has a prefix size of 20).



(a) GPT-2
EB-M$_{alt}(M, l, \text{SeqRep-2})$

(b) Transformer-XL
EB-M$_{alt}(M, l, \text{SeqRep-2})$

(c) Llama 3
EB-M$_{alt}(M, l, \text{SeqRep-2})$

(d) GPT-2, EB-M$_{\text{gapalt}}$
$(M(l_{\text{gap}})|D_{\text{corrupt}}(l), \text{SeqRep-2})$

(e) Transformer-XL, EB-M$_{\text{gapalt}}$
$(M(l_{\text{gap}})|D_{\text{corrupt}}(l), \text{SeqRep-2})$

(f) Llama 3, EB-M$_{\text{gapalt}}$
$(M(l_{\text{gap}})|D_{\text{corrupt}}(l), \text{SeqRep-2})$

Figure 3: EBC/EBM and EBC/EBM (GAP) ratios for various LM configurations detailing the disparities in diversity (given SeqRep-2) for generations created with greedy decoding, each pair of ratios measuring exposure bias and self-recovery respectively. In theory, if *k*NN-LM has a bias towards a minority of frequent tokens, the most obvious sign would be major differences in the repetition ratios. While our self-recovery metrics appear somewhat the same, the exposure bias metrics indicate an increase in diversity disparities, particularly in the poorly performing configuration, llama3+knnlm. This may suggest that while repetition may be an important factor in diagnosing *k*NN-LM's bias toward frequent tokens, more work is needed to verify this hypothesis. More detailed information about SeqRep-2 is found in Appendix D.1, and as SeqRep-2 does not need labels, it has alternate ratio definitions found in Appendix D.2.

B Is the *k*NN-LM Retriever Biased Towards a Minority of Frequent Tokens?

In this section, we go over early evidence that we believe supports our hypothesis that the *k*NN-LM retriever may be biased towards a frequent minority of tokens. The most surface-level reason for this would be an increase in disparities relating to lexical diversity. Hence, we utilize SeqRep-2 (Welleck et al., 2019; Fu et al., 2021; Li et al., 2023a) as an f_{score} to measure whether or not this is the case. SeqRep-2 does so by calculating the bigram repetition of a given generation. We utilize greedy decoding in our investigation as it falls victim to neural text degeneration (Holtzman et al., 2019), and hence is where repetition likely would be the most observable.

Our results are somewhat surprising: Subfigures 3a and 3b only show minor disparities in diversity, yet Subfigure 3c shows a high level of disparity associated with the poorly performing configuration (llama3+knnlm). This might be bizarre, given that Wang et al. (2023)'s results generally showed the *k*NN-LM retrievers tend to induce more lexical diversity. We suspect that our hypothesis still stands for one main reason: 1) that the frequency bias is largely influenced by hyperparameter tuning. The only configuration marked by great disparities in diversity is the one in which its interpolation weight (λ) is tuned solely for perplexity. Hence, the issue can potentially be avoided entirely when careful hyperparameter tuning is paired with stochastic sampling methods such as nucleus or top-k sampling. We recommend the latter to avoid neural text degeneration and poor open-ended text generation performance (Shi et al., 2024).

However, we should warn that we believe this hypothesis is not the only factor to consider: the difference in repetition disparities is much smaller in Subfigures 3d, 3e, 3f. This suggests that there are more factors involved in this bias than simply repetitive text, and we encourage future work to look towards this direction.

Prefix Size	GPT-2	Trf-XL	Llama 3
20	8840	4147	4886
40	6626	3109	3665
60	5302	2487	2930
80	4418	2072	2441
100	3786	1776	2092

Table 3: Sample Sizes for EB-M and EB-C

Prefix Size	Gap Size	GPT-2	Trf-XL	Llama 3
20	0	8840	4147	4886
20	30	5892	2763	3256
20	60	4418	2072	2441
60	0	5302	2487	2930
60	30	4077	1913	2254
60	60	3311	1553	1831
100	0	3786	1776	2092
100	30	3116	1462	1723
100	60	2649	1242	1464

Table 4: Sample Sizes for EB-M$_{gap}$ and EB-C$_{gap}$

C Additional Evaluation Information

This section largely covers additional evaluation information in the form of sample sizes for evaluation and computational resources.

Computational Resources The creation of the datastores and experiments of this work was completed on a local machine with a single RTX 4090 GPU alongside an i7-14700k processor, supplemented with 96 GB of CPU RAM (we note that less than 64 GB of RAM were occupied for the experiments and datastore creation). The datastore creation and evaluations took approximately 182 hours to complete with significant usage of both the GPU and CPU. The creation of the Transformer-XL datastore took approximately 27 hours to complete and the creation of the Llama-3 datastore took approximately 35 hours to complete. The evaluation took a total of approximately 120 hours. Because of the extended duration of the evaluation, all reported results are from a single run. For development, we estimate that the total active hardware hours would amount to 2-3 months.

We estimate that approximately 150 GB of storage space was needed during the evaluation to store the datastore indices and models. However, since the datastore's keys needed to be saved onto the disk before the creation of a FAISS (Johnson et al., 2019) index, approximately 1.2 TB of solid-state storage were needed, with Llama 3's keys requiring 991 GB and Transformer-XL's keys requiring 212 GB. Llama 3 particularly required larger storage due to its embedding dimension of 4096 (as opposed to Transformer-XL's 1024), as the required storage size scales with an LM's embedding dimension. Another factor that can impact required storage size is how the LM tokenizes the dataset, as some LMs generate more tokens for a text of the same length (see sample sizes for an example).

Sample Sizes This page contains the respective sample sizes for each exposure bias and self-recovery metric. We see that all prefix and gap sizes conform to the 1000 sample size requirement of the MAUVE divergence function (Pillutla et al., 2023).

D Extended Evaluation

This section covers the extended evaluation of this work, taking into account far more scoring and divergence functions. **We found that overwhelmingly, these alternate scoring and divergence functions supported our claims**. While there were some exceptions within certain divergence functions, we found that these did not occur consistently. For example, while we find Subfigures 6m and 9n to be a potential indication of *k*NN-LM variants introducing exposure bias/ impeding self-recovery for the RankGen (Krishna et al., 2022) divergence/similarity function, this behavior is quickly proven inconsistent by Subfigures 8m, 7n, 5n, and 4m, where this negative trend either does not appear consistently or at all.

Decoding Algorithms In our extended analysis, we also evaluated three other decoding algorithms aside from nucleus sampling (Holtzman et al., 2019): greedy decoding, top-*k* (where *k*=40) (Fan et al., 2018; Holtzman et al., 2018; Radford et al., 2019) sampling, and ancestral(pure) sampling. Ancestral sampling was specifically studied per the recommendation of He et al. (2021b) to measure exposure bias. However, as nearly all decoding algorithms supported our results, we chose to only report nucleus sampling in our extended results and greedy decoding to study *k*NN-LM's effects on repetition in Appendix B for the sake of space.

Regarding Similarity Functions Two alternate divergence functions, RankGen (Krishna et al., 2022) and GTE-Similarity[5] (Li et al., 2023b) are similarity functions as opposed to divergence functions. To accommodate these two functions, we loosen our definition of f_{div} to include similarity functions in this analysis.

D.1 Alternate Scoring and Divergence Functions

This section is dedicated towards explaining the additional scoring and divergence functions we utilized, their motivations, and potential edge cases in which we observed the functions became unstable. All functions that are considered potentially unstable (although not always) are listed with a *. Some alternate scoring and divergence functions require modified definitions for them to be suitable in our evaluations, and we defer these to Appendix D.2.

Alternate Divergence/Similarity Functions

GTE-Similarity MAUVE has the issue that it requires at least 256 tokens to determine the differences between two texts confidently (Pillutla et al., 2023). As we are limited to single token generations in our consistency metrics, we developed a metric to create a second opinion for MAUVE: GTE Similarity. Functionally, GTE-Similarity is simply the cosine similarity between each prediction and labels utilizing the `gte-large-en-v1.5` model as the means to encode the predictions and the labels (Li et al., 2023b; Zhang et al., 2024a). We chose `gte-large-en-v1.5` model for its high performance on the Massive Text Embedding Benchmark (MTEB) (Muennighoff et al., 2022) and its low parameter count (434M), allowing for quick evaluation. At the time of writing, it is currently ranked 22nd on the MTEB Leaderboard[6]. GTE Similarity is calculated by taking the inputs of the encoded generation and label vectors (V_g, V_l), and computing the cosine similarity between them:

$$\text{GTE Similarity}(V_g, V_l) = \frac{V_g \cdot V_l}{|V_g||V_l|} \quad (10)$$

It should also be noted that because this similarity function is highly derivative, it also likely shares the same limitations and biases as cosine similarity and `gte-large-en-v1.5`. Namely, the largest of these is that cosine similarity does not account for the magnitude of the encoded vectors due to normalization in its equation, meaning certain differences in embeddings may not be potentially detected as a result. While we were not able to find any biases specifically related to `gte-large-en-v1.5`, it likely shares many limitations common to embedding models such as being limited to grammatical errors and lack of context (Harris et al., 2024). Due to this, we recommend only interpreting GTE Similarity's results in tandem with other divergence/similarity functions.

MAUVE Variants In particular, MAUVE utilizes the embeddings of the 774M parameter GPT-2 Large to rate the consistency between two given texts. However, the GPT-2 variant of this divergence function has several flaws, such as ignoring errors in the beginning and middle of text (He et al., 2023). This is particularly concerning as EB-C

[5]GTE-Similarity was created in Appendix D.1, using `gte-large-en-v1.5` (Li et al., 2023b) as the embedding model.

[6]https://huggingface.co/spaces/mteb/leaderboard

and EB-C$_{gap}$ score single token generations, meaning the token is both the beginning, middle, and end. Hence, we also employ MAUVE variants such as RoBERTa MAUVE (Conneau et al., 2019; Liu et al., 2020; Pillutla et al., 2021; He et al., 2023), and ELECTRA MAUVE (Clark et al., 2020; He et al., 2021b), which do not have this weakness and correlate more with human judgment. For calculating both MAUVE variants, we use the 561M parameter `xlm-roberta-large`, and the 335M parameter `electra-large-discriminator` respectively. It should be noted that these come with their caveats, such as RoBERTa MAUVE heavily penalizing smaller models such as the 117M parameter `gpt-2` LM score-wise (Pillutla et al., 2021), and ELECTRA MAUVE penalizing minor erroneous output harshly (He et al., 2023).

RankGen RankGen (Krishna et al., 2022) is a similarity function utilized to score the relevance of a model's generated text (suffix) and its prompt (prefix) utilizing the dot product- a higher RankGen score indicates a higher relevancy. As RankGen uses the dot product, we take the mean of the diagonal of the dot product as the RankGen score so as only to score the relevance of a prediction with its corresponding label. We implement this metric as it was used extensively in Wang et al. (2023). We also considered utilizing RankGen potentially as a scoring function instead due to its divergence variant comparing 1 token against a 20 token prompt, which theoretically may have caused issues. In practice, it performed similarly to its divergence counterpart, hence we decided against utilizing it further.

Alternate Scoring Functions

BERTScore Precision/Recall He et al. (2023) states that BERTScore Precision and BERTScore F1 can be potentially thrown off by truncations in an LM's labels, with BERTScore Recall being the only BERTScore variant not afflicted. Hence, we report all three variants of BERTScore.

Entity F1 Wang et al. (2023) largely uses Entity F1 (Derczynski, 2016; Nan et al., 2021; Lee et al., 2022) as a means to evaluate the rate of hallucinated entities in a text and by proxy, the potential general rate of hallucination. A higher Entity F1 score would indicate less of these hallucinations. However, as our evaluations deal with short generations (20 tokens) as opposed to Wang et al. (2023)'s large generations (256 tokens), we felt the need to acclimatize Entity F1 accordingly due to this constraint. Hence, rather than taking the mean Entity F1 between all label and prediction pairs, we take the Entity F1 between the entirety of the labels and predictions, as otherwise, we believe the function would be largely unstable. In our scoring, we only consider exact matches and remove repeated entities for simplicity.

GPT-2 Perplexity* Wang et al. (2023) previously used GPT-3 Perplexity (Brown et al., 2020) as a quality measure. Specifically, they used the 6.7B parameter model `gpt3-curie` to do so. Because the model used to gather GPT-3 perplexity, was shut down[7], we instead use the 774M parameter `gpt2-large` (Radford et al., 2019) as a replacement to score perplexity. We take the mean perplexity of all the generated outputs.

Unfortunately, for many reasons, GPT-2 Perplexity remains unstable. This is largely because of the design of exposure bias and self-recovery. Namely, we found that perplexity will always prefer generations built on a model prefix in our exposure bias evaluations. Similarly, GPT-2 perplexity tends to skyrocket as a result of our corrupted prefix in our self-recovery evaluations. We largely kept GPT-2 Perplexity in our results as some interesting phenomena occurred. For example, in Subfigure 9i, the base model and the *k*NN-LM variant with an extended datastore appear to have an extremely low ratio (caused by an abnormally high corrupted prefix perplexity), while the other variants able to be stable. While we believe this is coincidental, we chose to report the results due to their abnormality.

SeqRep1-4* Wang et al. (2023) utilized SeqRep (Welleck et al., 2019; Fu et al., 2021; Li et al., 2023a) as a means to gauge lexical (*n*-gram) diversity, hence, we also implement this scoring function. We report unigram, bigram, trigram, and four-gram repetition (or otherwise, SeqRep1-4).

Why this function is unstable is fairly interesting: it tends to be stable with repetitive text, yet tends to be unstable when a model's outputs are sufficiently diverse, to the point of the ground truth/uncorrupted prefixes often appearing higher. While this could be due to the setup of our evaluation (i.e. it would make sense an artificially corrupted/diverse prefix may lead to artificial diversity), we still keep it to study if repetition can be a factor in exposure bias, specifically in Appendix B.

[7] https://platform.openai.com/docs/deprecations/instructgpt-models

D.2 Modified Exposure Bias and Self Recovery Metrics

The issue with the current definitions given in Section 3 is that some metrics do not necessarily *conform* to them. In particular, we denoted such changes by each metric having a subscript named "alt"(short for alternative), appended to the label. For instance, a modified EB-M is called EB-M$_{alt}$.

Alternate EB-M Definitions For Perplexity and SeqRep Perplexity and SeqRep(1-4) (Welleck et al., 2019; Fu et al., 2021; Li et al., 2023a) are scoring functions that do not require labels ($P_D^{W_{l+1:l+l_{\text{gen}}}}$) for scoring, as they only score the generation ($P_{M|D}^{W_{l+1:l+l_{\text{gen}}}}$) in terms of quality and diversity respectively. Hence, the alternate EB-M definition for the aforementioned scoring functions is as follows:

$$\text{EB-M}_{alt}(M, l, f_{\text{score}}) = \frac{f_{\text{score}}(P_{M|D}^{W_{l+1:l+l_{\text{gen}}}})}{f_{\text{score}}(P_{M|M}^{W_{l+1:l+l_{\text{gen}}}})}. \quad (11)$$

Alternative EB-C Metric Definition for RankGen RankGen (Krishna et al., 2022) instead of scoring the consistency between a generation ($P_M \cdot |C, W_{1:l}$) and its corresponding label ($P_D(\cdot|C, W_{1:l})$) score the consistency between the generation and its corresponding prompt. Specifically, the context of a generation is its fixed prompt combined with its prefix, ($P_D(C, W_{1:l})$). Hence, the modified EB-C definition for RankGen is as follows:

$$\text{CGD}_{alt}(M|H(l), f_{\text{div}}) = \mathop{\mathbb{E}}_{C\sim P_D, W_{1:l}\sim P_H(\cdot|C)}[f_{\text{div}}(P_M(\cdot|C, W_{1:l}), P_D(C))]. \quad (12)$$

Where the modified CGD is part of the following equation:

$$\text{EB-C}_{alt}(M, l, f_{\text{div}}) = \frac{\text{CGD}_{alt}(M|M(l), f_{\text{div}})}{\text{CGD}_{alt}(M|D(l), f_{\text{div}})}. \quad (13)$$

It should be noted that the labels are larger than the generation(in our work, this is a ratio of 20:1 tokens). However as RankGen uses embeddings, the dimensions of the encoded labels and inputs will be the same (from the embedding dimension). Hence, the mismatch is a non-issue for RankGen.

Alternate EB-M$_{gap}$ Definitions For Perplexity and SeqRep1 Similar to their behavior in EB-M, Perplexity, and SeqRep1 largely do not require scoring labels. The alternative definition of EB-M $_{gap}$ is as follows for these scoring functions:

$$\text{EB-M}_{\text{gapalt}}(M(l_{\text{gap}})|D_{\text{corrupt}}(l), f_{\text{score}}) = \frac{f_{\text{score}}(P_{M|D(l')}^{W_{l'+1:l'+l_{\text{gen}}}})}{f_{\text{score}}(P_{M|D_{\text{corrupt}}(l)}^{W_{l'+1:l'+l_{\text{gen}}}})} \quad (14)$$

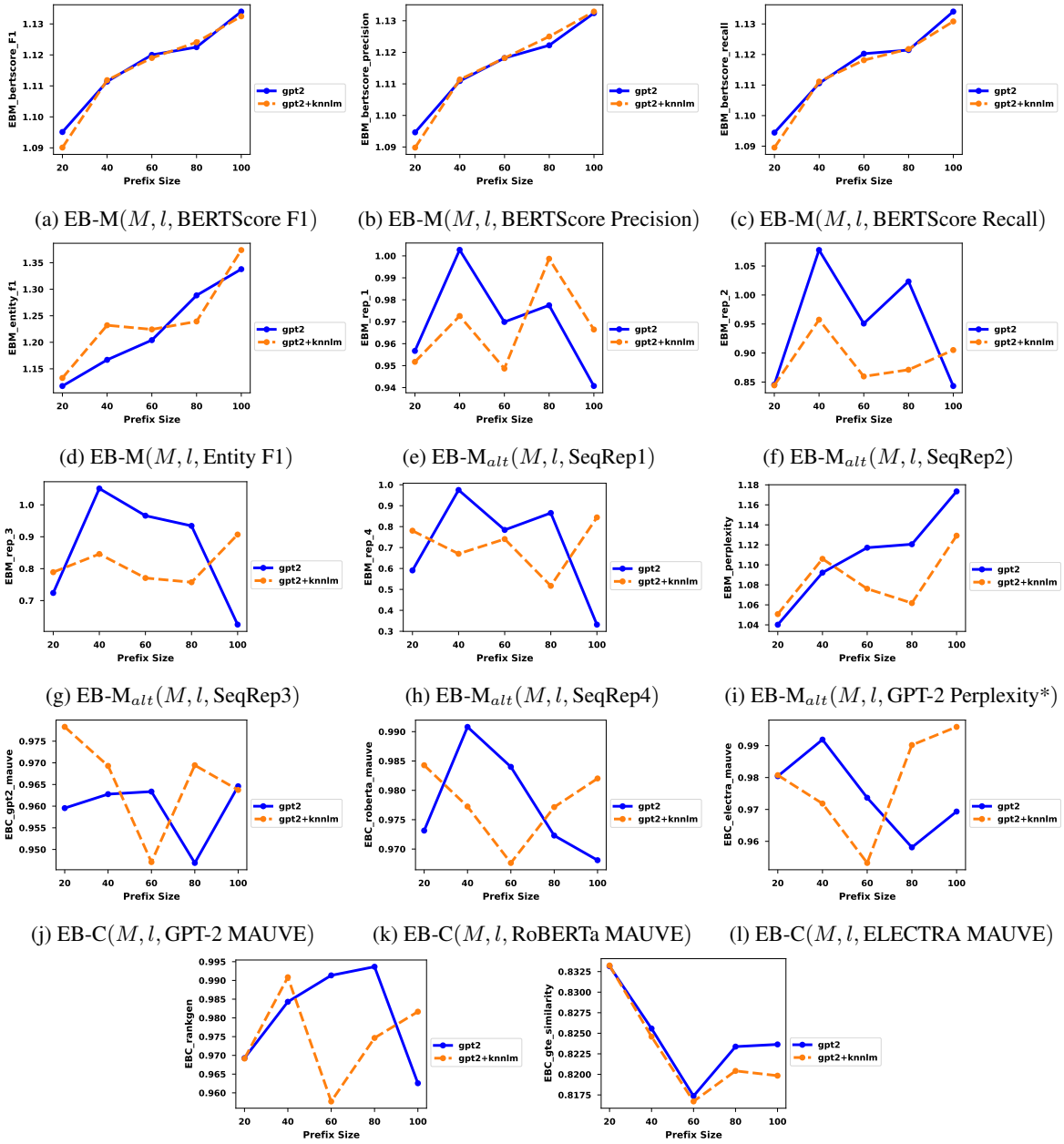
Alternative EB-C$_{gap}$ Metric Definition for RankGen Similar to its behavior in EB-C, RankGen utilizes associated prompts instead of associated labels. Hence, the altered definition is as follows:

$$\text{CGD}_{\text{gapalt}}(M(l_{\text{gap}})|D_{\text{corrupt}}(l)) = \mathop{\mathbb{E}}_{W_{1:l}^{\text{corrupt}}\sim P_D^{\text{corrupt}}, W_{l+1:l+l_{\text{gap}}}\sim P_M(\cdot|W_{1:l}^{\text{corrupt}}), C\sim P_D} [f_{\text{div}}(P_M(\cdot|W_{1:l}^{\text{corrupt}}, W_{l+1:l+l_{\text{gap}}}), P_D(C)], \quad (15)$$

Likewise, it follows that the definition of EB-C $_{gapalt}$ is altered from EB-C$_{gapalt}$:

$$\text{EB-C}_{\text{gapalt}}(M(l_{\text{gap}})|D_{\text{corrupt}}(l), f_{\text{div}}) = \frac{\text{CGD}_{\text{gap}}(M(l_{\text{gap}})|D_{\text{corrupt}}(l), f_{\text{div}})}{\text{CGD}(M|D(l+l_{\text{gap}}), f_{\text{div}})}. \quad (16)$$

D.3 GPT-2 Results



(a) EB-M$(M, l,$ BERTScore F1) (b) EB-M$(M, l,$ BERTScore Precision) (c) EB-M$(M, l,$ BERTScore Recall)

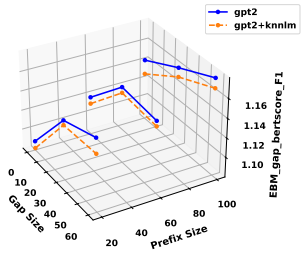
(d) EB-M$(M, l,$ Entity F1) (e) EB-M$_{alt}(M, l,$ SeqRep1) (f) EB-M$_{alt}(M, l,$ SeqRep2)

(g) EB-M$_{alt}(M, l,$ SeqRep3) (h) EB-M$_{alt}(M, l,$ SeqRep4) (i) EB-M$_{alt}(M, l,$ GPT-2 Perplexity*)

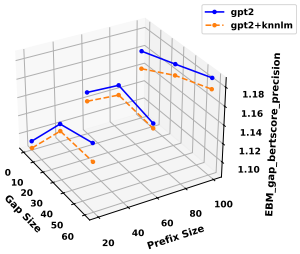
(j) EB-C$(M, l,$ GPT-2 MAUVE) (k) EB-C$(M, l,$ RoBERTa MAUVE) (l) EB-C$(M, l,$ ELECTRA MAUVE)

(m) EB-C$_{alt}(M, l,$ RankGen) (n) EB-C$(M, l,$ GTE Similarity)

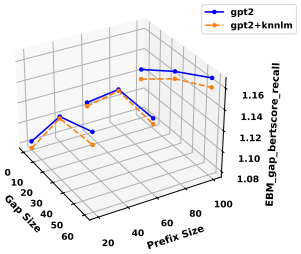
Figure 4: EBM/EBC Ratios for several GPT-2 configurations detailing the disparities in quality, diversity, and consistency (measured via various scoring/divergence functions) between generated text conditioned on model and ground truth prefixes of increasing size. **A ratio farther from 1 indicates more discrepancy**, i.e. exposure bias, **regardless of the exact ratio value**, except for functions demarcated by * (See Appendix D.1). In theory, the kNN-LM variant should see increased exposure bias compared to their standard LM counterparts among all functions. Yet in practice, they typically exhibit equal or lower levels of exposure bias.



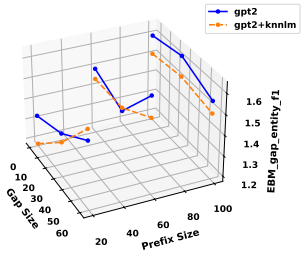
(a) EB-M$_{\text{gap}}(M(l_{\text{gap}})|D_{\text{corrupt}}(l)$, BERTScore-F1)



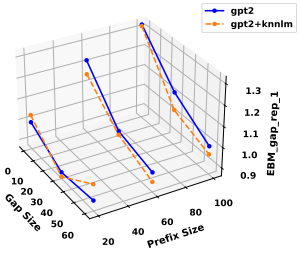
(b) EB-M$_{\text{gap}}(M(l_{\text{gap}})|D_{\text{corrupt}}(l)$, BERTScore-Pr.)



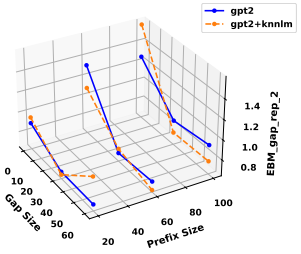
(c) EB-M$_{\text{gap}}(M(l_{\text{gap}})|D_{\text{corrupt}}(l)$, BERTScore-F1)



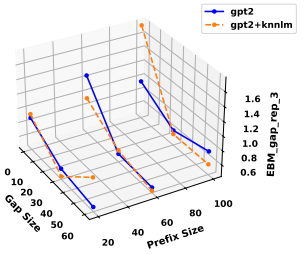
(d) EB-M$_{\text{gap}}(M(l_{\text{gap}})|D_{\text{corrupt}}(l)$, Entity-F1)



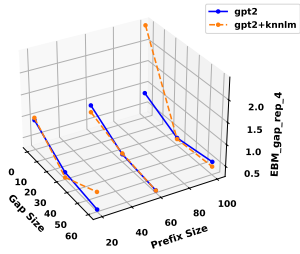
(e) EB-M$_{\text{gapalt}}(M(l_{\text{gap}})|D_{\text{corrupt}}(l)$, SeqRep1*)



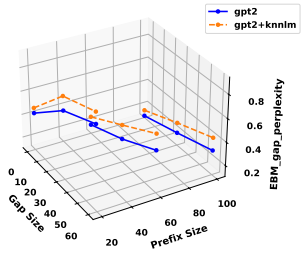
(f) EB-M$_{\text{gapalt}}(M(l_{\text{gap}})|D_{\text{corrupt}}(l)$, SeqRep2*)



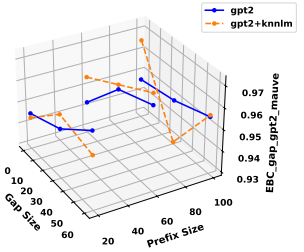
(g) EB-M$_{\text{gapalt}}(M(l_{\text{gap}})|D_{\text{corrupt}}(l)$, SeqRep3*)



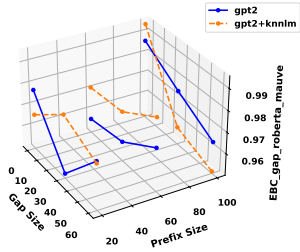
(h) EB-M$_{\text{gapalt}}(M(l_{\text{gap}})|D_{\text{corrupt}}(l)$, SeqRep4*)



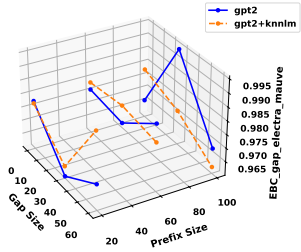
(i) EB-M$_{\text{gapalt}}(M(l_{\text{gap}})|D_{\text{corrupt}}(l)$, GPT-2 Perplexity*)



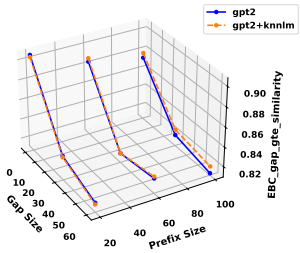
(j) EB-C$_{\text{gap}}(M(l_{\text{gap}})|D_{\text{corrupt}}(l)$, GPT-2 MAUVE)



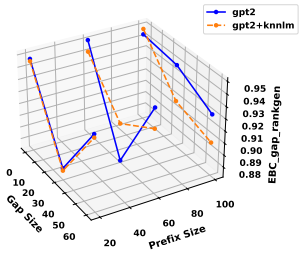
(k) EB-C$_{\text{gap}}(M(l_{\text{gap}})|D_{\text{corrupt}}(l)$, RoBERTa MAUVE)



(l) EB-C$_{\text{gap}}(M(l_{\text{gap}})|D_{\text{corrupt}}(l)$, ELECTRA MAUVE)



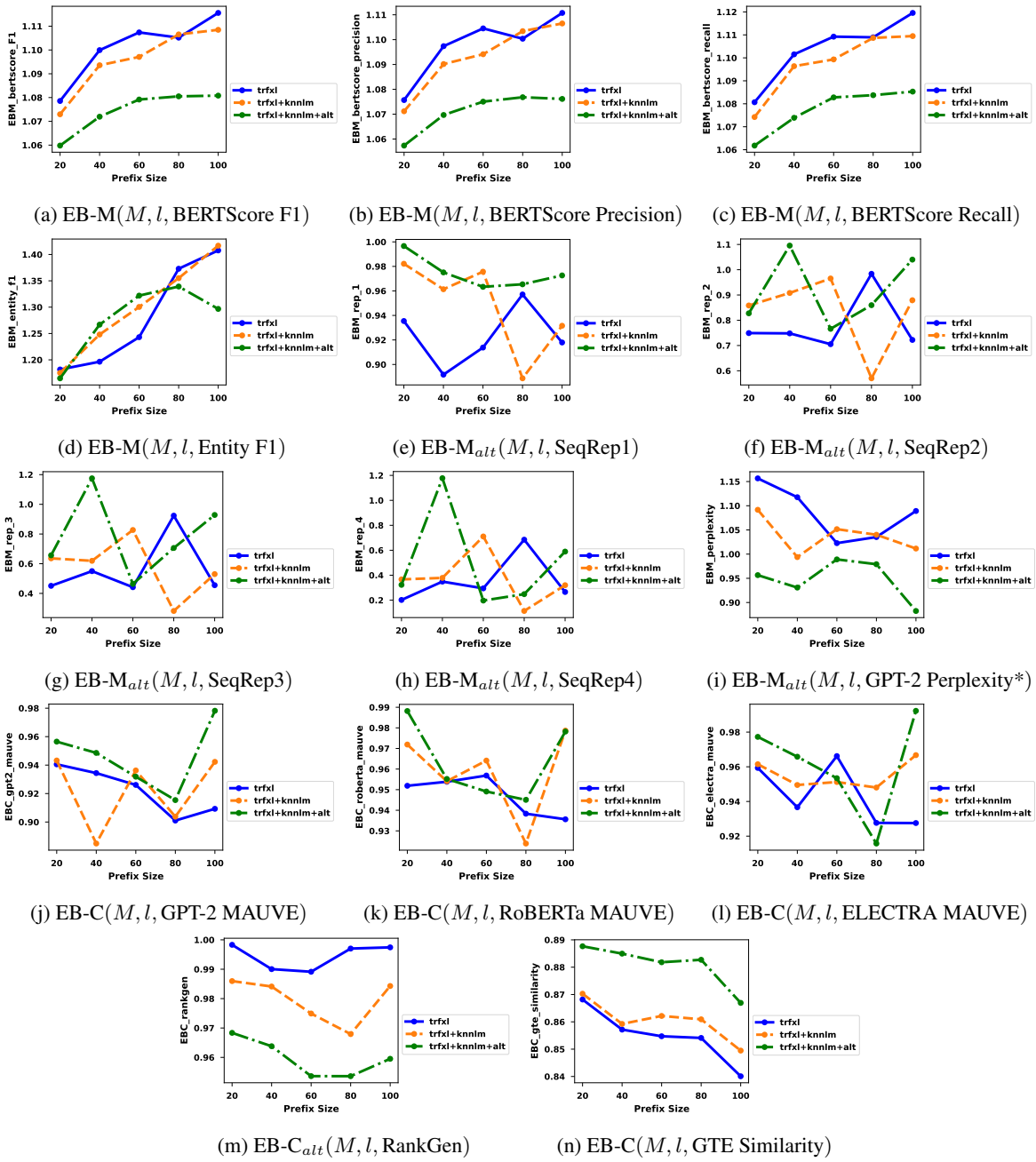
(m) EB-C$_{\text{gap}}(M(l_{\text{gap}})|D_{\text{corrupt}}(l)$, GTE-Similarity)



(n) EB-C$_{\text{gapalt}}(M(l_{\text{gap}})|D_{\text{corrupt}}(l)$, RankGen)

Figure 5: EBC/EBM (GAP) ratios for various GPT-2 configurations which measure disparities in quality, diversity, and consistency (via various scoring and divergence functions) between text generated with a ground truth prefix and its 30% corrupted version. As the model prefix size (gap) increases, the model should in theory self-recover, **by moving the ratio closer to 1**, **regardless of the exact value**, except for functions demarcated by * (see Appendix D.1). However, in practice, no configuration consistently achieves this, and the *k*NN-LM configuration generally performs similarly to the standard configuration.

D.4 Transformer-XL Results



(a) EB-M$(M, l,$ BERTScore F1) (b) EB-M$(M, l,$ BERTScore Precision) (c) EB-M$(M, l,$ BERTScore Recall)

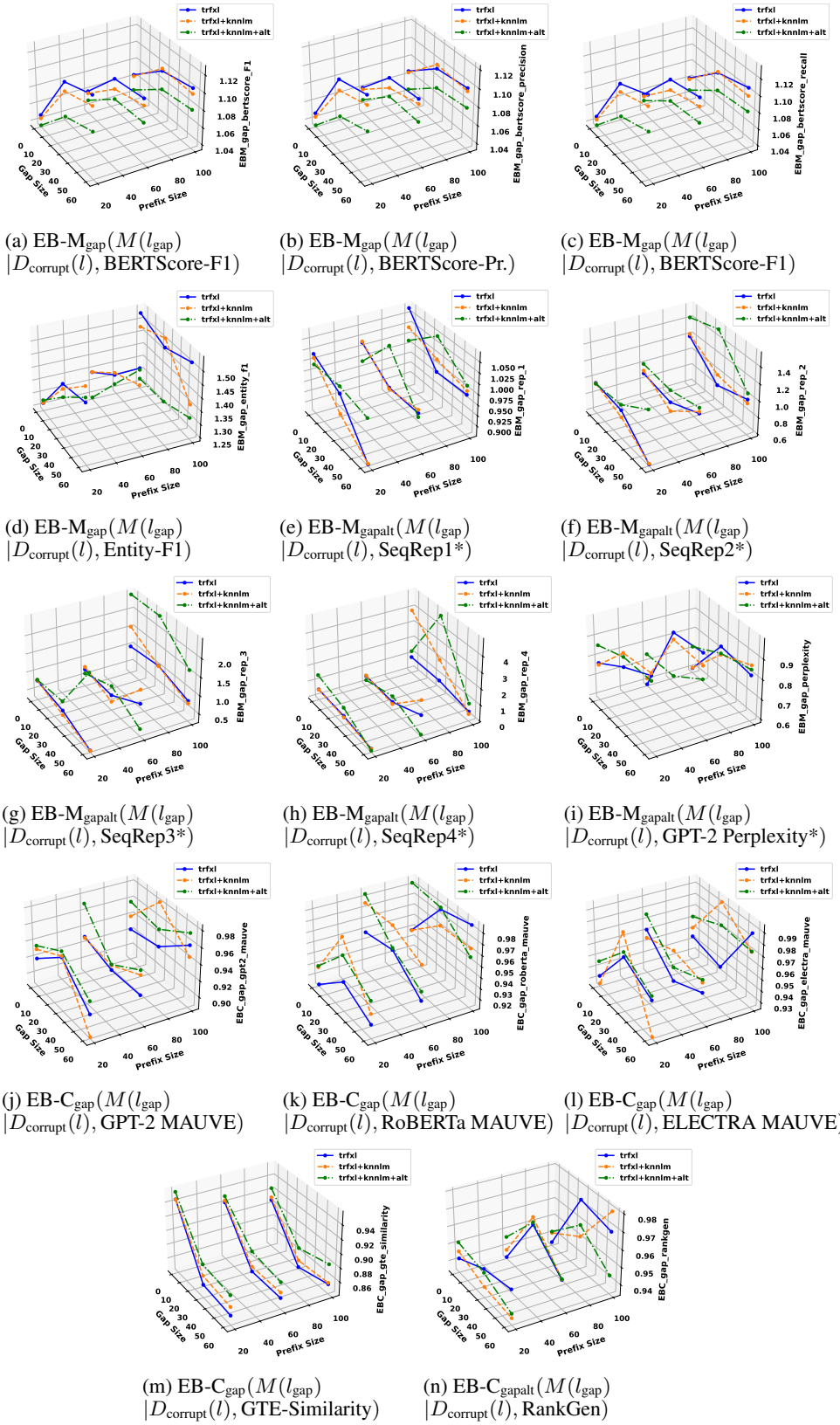
(d) EB-M$(M, l,$ Entity F1) (e) EB-M$_{alt}(M, l,$ SeqRep1) (f) EB-M$_{alt}(M, l,$ SeqRep2)

(g) EB-M$_{alt}(M, l,$ SeqRep3) (h) EB-M$_{alt}(M, l,$ SeqRep4) (i) EB-M$_{alt}(M, l,$ GPT-2 Perplexity*)

(j) EB-C$(M, l,$ GPT-2 MAUVE) (k) EB-C$(M, l,$ RoBERTa MAUVE) (l) EB-C$(M, l,$ ELECTRA MAUVE)

(m) EB-C$_{alt}(M, l,$ RankGen) (n) EB-C$(M, l,$ GTE Similarity)

Figure 6: EBM/EBC Ratios for several Transformer-XL configurations detailing the disparities in quality, diversity, and consistency (measured via various scoring/divergence functions) between generated text conditioned on model and ground truth prefixes of increasing size. This includes a configuration with **alt**ernate hyperparameters and an **ext**ended datastore. **A ratio farther from 1 indicates more discrepancy**, i.e. exposure bias, **regardless of the exact ratio value**, except for functions demarcated by * (See Appendix D.1). In theory, all of the kNN-LM variants should see increased exposure bias compared to their standard LM counterparts among all functions. Yet in practice, they typically exhibit equal or lower levels of exposure bias.



(a) EB-M$_{\text{gap}}(M(l_{\text{gap}})|D_{\text{corrupt}}(l)$, BERTScore-F1)

(b) EB-M$_{\text{gap}}(M(l_{\text{gap}})|D_{\text{corrupt}}(l)$, BERTScore-Pr.)

(c) EB-M$_{\text{gap}}(M(l_{\text{gap}})|D_{\text{corrupt}}(l)$, BERTScore-F1)

(d) EB-M$_{\text{gap}}(M(l_{\text{gap}})|D_{\text{corrupt}}(l)$, Entity-F1)

(e) EB-M$_{\text{gapalt}}(M(l_{\text{gap}})|D_{\text{corrupt}}(l)$, SeqRep1*)

(f) EB-M$_{\text{gapalt}}(M(l_{\text{gap}})|D_{\text{corrupt}}(l)$, SeqRep2*)

(g) EB-M$_{\text{gapalt}}(M(l_{\text{gap}})|D_{\text{corrupt}}(l)$, SeqRep3*)

(h) EB-M$_{\text{gapalt}}(M(l_{\text{gap}})|D_{\text{corrupt}}(l)$, SeqRep4*)

(i) EB-M$_{\text{gapalt}}(M(l_{\text{gap}})|D_{\text{corrupt}}(l)$, GPT-2 Perplexity*)

(j) EB-C$_{\text{gap}}(M(l_{\text{gap}})|D_{\text{corrupt}}(l)$, GPT-2 MAUVE)

(k) EB-C$_{\text{gap}}(M(l_{\text{gap}})|D_{\text{corrupt}}(l)$, RoBERTa MAUVE)

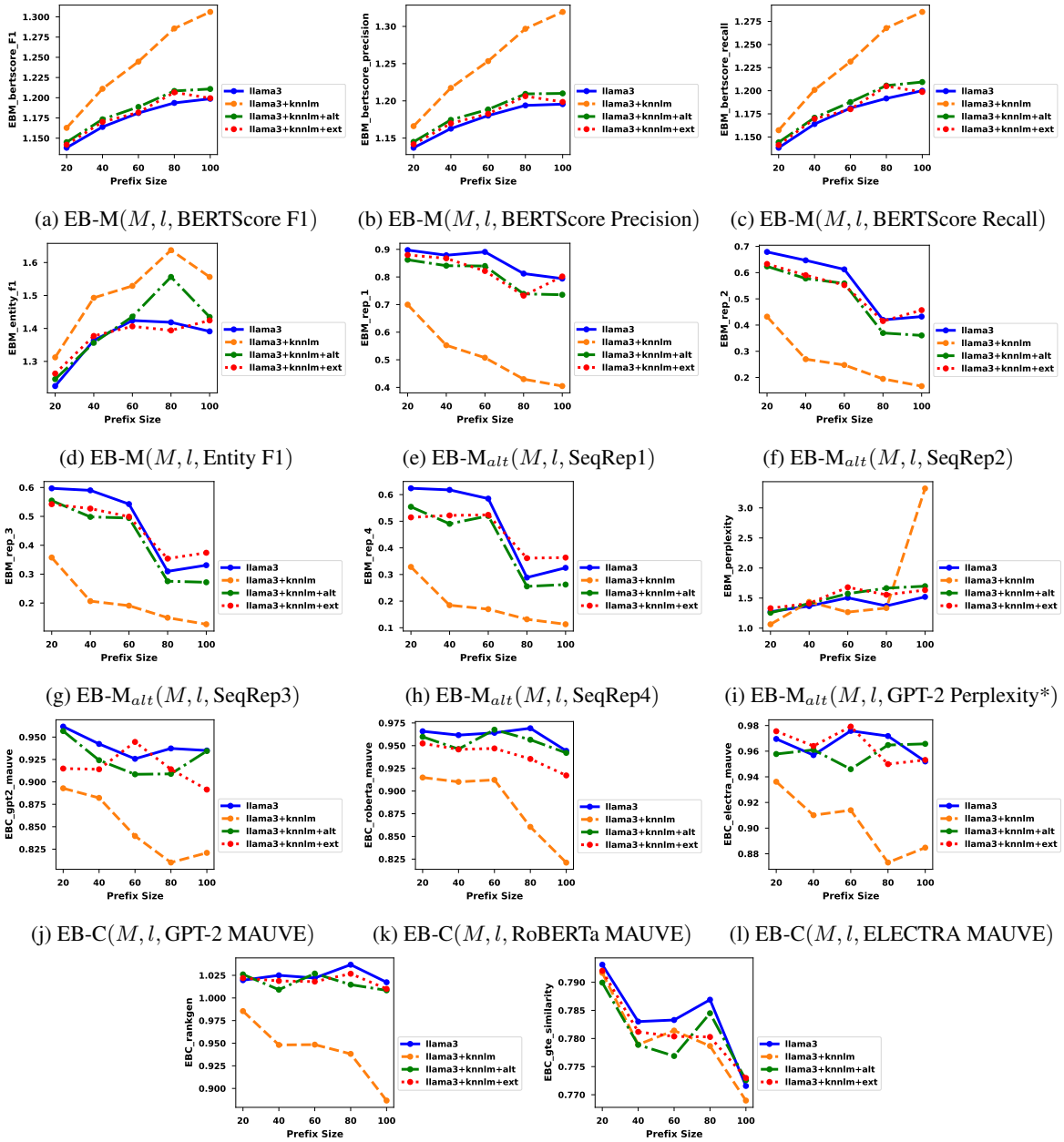
(l) EB-C$_{\text{gap}}(M(l_{\text{gap}})|D_{\text{corrupt}}(l)$, ELECTRA MAUVE)

(m) EB-C$_{\text{gap}}(M(l_{\text{gap}})|D_{\text{corrupt}}(l)$, GTE-Similarity)

(n) EB-C$_{\text{gapalt}}(M(l_{\text{gap}})|D_{\text{corrupt}}(l)$, RankGen)

Figure 7: EBC/EBM (GAP) ratios for various Transformer-XL configurations which measure disparities in quality, diversity, and consistency (via various scoring and divergence functions) between text generated with a ground truth prefix and its 30% corrupted version. As the model prefix size (gap) increases, the model should in theory self-recover, **by moving the ratio closer to 1**, **regardless of the exact value**, except for functions demarcated by * (see Appendix D.1). However, in practice, no configuration consistently achieves this, and the *k*NN-LM configurations generally perform similarly to the standard configuration.

D.5 Llama 3 Results



(a) EB-M$(M, l,$ BERTScore F1) (b) EB-M$(M, l,$ BERTScore Precision) (c) EB-M$(M, l,$ BERTScore Recall)

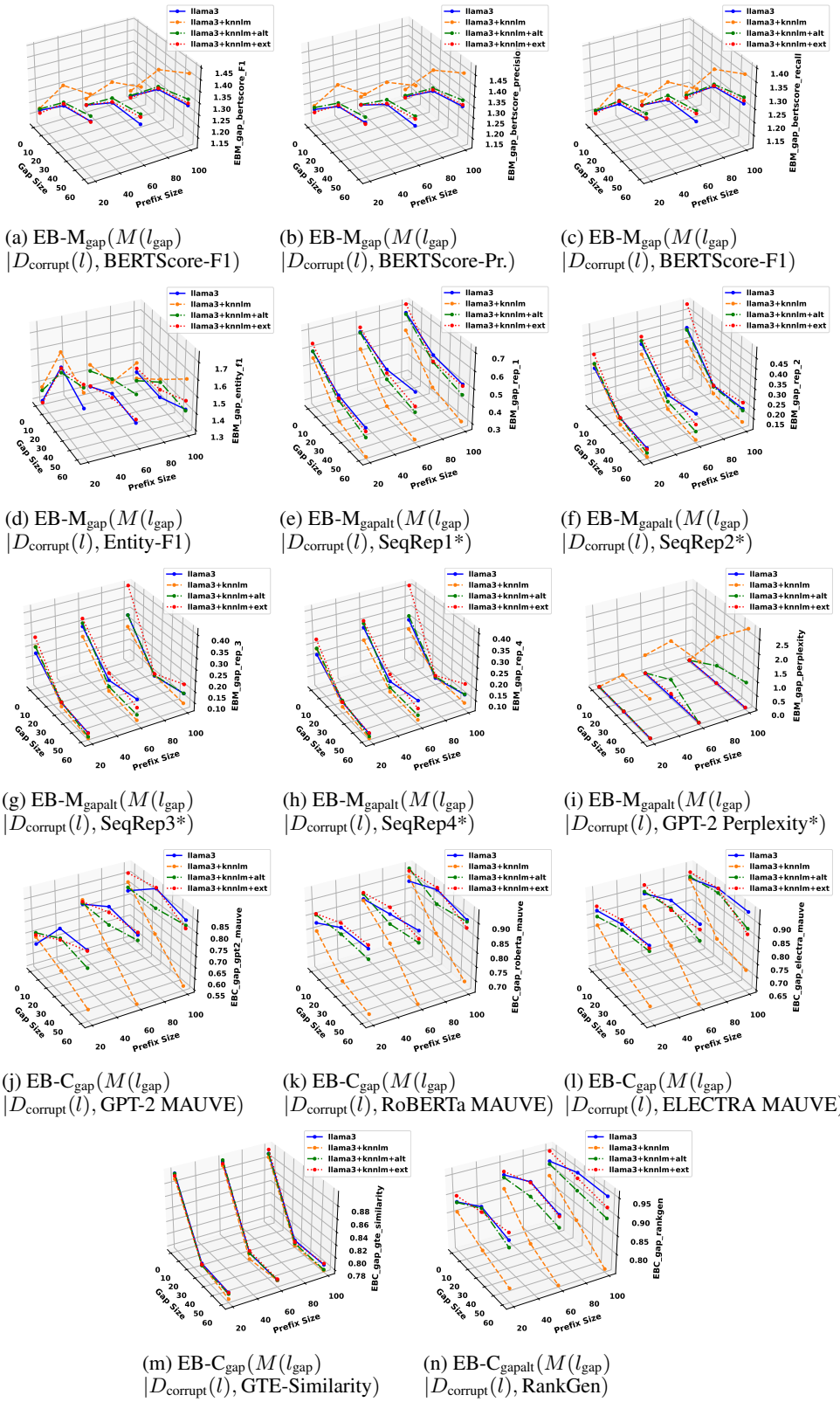
(d) EB-M$(M, l,$ Entity F1) (e) EB-M$_{alt}(M, l,$ SeqRep1) (f) EB-M$_{alt}(M, l,$ SeqRep2)

(g) EB-M$_{alt}(M, l,$ SeqRep3) (h) EB-M$_{alt}(M, l,$ SeqRep4) (i) EB-M$_{alt}(M, l,$ GPT-2 Perplexity*)

(j) EB-C$(M, l,$ GPT-2 MAUVE) (k) EB-C$(M, l,$ RoBERTa MAUVE) (l) EB-C$(M, l,$ ELECTRA MAUVE)

(m) EB-C$_{alt}(M, l,$ RankGen) (n) EB-C$(M, l,$ GTE Similarity)

Figure 8: EBM/EBC Ratios for several Llama 3 configurations detailing the disparities in quality, diversity, and consistency (measured via various scoring/divergence functions) between generated text conditioned on model and ground truth prefixes of increasing size. This includes a configuration with **alt**ernate hyperparameters and an **ext**ended datastore. **A ratio farther from 1 indicates more discrepancy**, i.e. exposure bias, **regardless of the exact ratio value**, except for functions demarcated by * (See Appendix D.1). In theory, all of the kNN-LM variants should see increased exposure bias compared to their standard LM counterparts among all functions. Yet in practice, they generally exhibit equal or lower levels of exposure bias, with the key exception of the configuration of llama3+knnlm.



(a) EB-M$_{\text{gap}}(M(l_{\text{gap}})|D_{\text{corrupt}}(l)$, BERTScore-F1)

(b) EB-M$_{\text{gap}}(M(l_{\text{gap}})|D_{\text{corrupt}}(l)$, BERTScore-Pr.)

(c) EB-M$_{\text{gap}}(M(l_{\text{gap}})|D_{\text{corrupt}}(l)$, BERTScore-F1)

(d) EB-M$_{\text{gap}}(M(l_{\text{gap}})|D_{\text{corrupt}}(l)$, Entity-F1)

(e) EB-M$_{\text{gapalt}}(M(l_{\text{gap}})|D_{\text{corrupt}}(l)$, SeqRep1*)

(f) EB-M$_{\text{gapalt}}(M(l_{\text{gap}})|D_{\text{corrupt}}(l)$, SeqRep2*)

(g) EB-M$_{\text{gapalt}}(M(l_{\text{gap}})|D_{\text{corrupt}}(l)$, SeqRep3*)

(h) EB-M$_{\text{gapalt}}(M(l_{\text{gap}})|D_{\text{corrupt}}(l)$, SeqRep4*)

(i) EB-M$_{\text{gapalt}}(M(l_{\text{gap}})|D_{\text{corrupt}}(l)$, GPT-2 Perplexity*)

(j) EB-C$_{\text{gap}}(M(l_{\text{gap}})|D_{\text{corrupt}}(l)$, GPT-2 MAUVE)

(k) EB-C$_{\text{gap}}(M(l_{\text{gap}})|D_{\text{corrupt}}(l)$, RoBERTa MAUVE)

(l) EB-C$_{\text{gap}}(M(l_{\text{gap}})|D_{\text{corrupt}}(l)$, ELECTRA MAUVE)

(m) EB-C$_{\text{gap}}(M(l_{\text{gap}})|D_{\text{corrupt}}(l)$, GTE-Similarity)

(n) EB-C$_{\text{gapalt}}(M(l_{\text{gap}})|D_{\text{corrupt}}(l)$, RankGen)

Figure 9: EBC/EBM (GAP) ratios for various Llama 3 configurations which measure disparities in quality, diversity, and consistency (via various scoring and divergence functions) between text generated with a ground truth prefix and its 30% corrupted version. As the model prefix size (gap) increases, the model should in theory self-recover, **by moving the ratio closer to 1**, **regardless of the exact value**, except for functions demarcated by * (see Appendix D.1). However, in practice, no configuration consistently achieves this, and the *k*NN-LM configurations generally perform similarly to the standard configuration among all functions except the configuration of llama3+knnlm.

E Implementation Details

This section covers the implementation details concerning text generation, the datastore/dataset, and scoring/divergence functions. At the highest level, this project uses Alon et al. (2022)'s implementation of kNN-LM[8]. This also includes the fine-tuned variants of the 117M parameter GPT-2 LM[9] (Radford et al., 2019; Alon et al., 2022) used in our evaluations, and its corresponding datastore[10]. Similarly, for Llama 3 (Dubey et al., 2024), we use the extended datastore[11] available from Shi et al. (2022); Geng et al. (2024). Our primary difference is that rather using the `faiss-gpu` library, we instead use the `faiss-cpu==1.8.0` library for evaluation. This is due to all variants of MAUVE (Pillutla et al., 2021; Liu et al., 2021; Pillutla et al., 2023) requiring the use of `faiss-cpu`. The primary libraries used by the implementation are `transformers==4.44.2` `accelerate==0.30.1`, `evaluate==0.4.1`, `datasets==2.19.0`. Additionally, we set the `probe` parameter to 1, for evaluation speed (at the expense of some accuracy in kNN search). However, it should be noted that `Meta-Llama-3-8B` requires `sentencepiece==0.2.0` for its tokenizer, and similarly, `transfo-xl/transfo-xl-wt103` requires the `sacremoses==0.1.1` dependency for its tokenizer. For many of the dependencies we did not list, we do so under specific scoring/divergence functions in this section. We additionally note that every model, metric, and dataset is primarily built for English language modeling.

E.1 Dataset Information

Wikitext-103 (Merity et al., 2016) is a language modeling dataset released by Salesforce containing "Good and Certified" Wikipedia articles in the English language. It contains approximately 100 million tokens. Note that we use the raw token level `wikitext-103-raw-v1` version of the Wikitext-103 dataset, following Alon et al. (2022)'s implementation.

[8] https://github.com/neulab/knn-transformers

[9] https://huggingface.co/neulab/gpt2-finetuned-wikitext103

[10] We use all the files under the "gpt-2" directory in https://knn-transformers.s3.amazonaws.com/index.html

[11] https://huggingface.co/datasets/wentingzhao/knn-prompt-datastore

E.2 Text Generation Information

As text generation was the foundational building block of our evaluation, we list its related settings here. In particular, we use huggingface transformer's `model.generate()` function to generate text, where `model` is one of our evaluated LMs (e.g. GPT-2) loaded via `AutoModelForCausalLM.from_pretrained()`. Our parameters common to all decoding methods are as follows:

- `max_new_tokens = num_generations`
- `min_new_tokens = num_generations`
- `pad_token_id = 50256`

This allows us to have an exact amount of generations, in this work, we use 20. 50256 is the <|endoftext|> for GPT-2, and we use this largely as a dummy value for padding (which is somewhat redundant as all input values are the same length and hence this only exists to dismiss a warning regarding a missing pad token). We use `do_sample = True` for nucleus sampling (Holtzman et al., 2019) top-k sampling(Fan et al., 2018; Holtzman et al., 2018; Radford et al., 2019), and ancestral sampling to stochastically sample. We set this parameter to false for greedy decoding. Generally, then the hyperparameters would be straightforward: `top-k` is set to 40 following Wang et al. (2023) and ancestral sampling requires no extra hyperparameters. The exception is nucleus sampling as by default, top-k is utilized with nucleus sampling. Hence, we explicitly call the following parameters to invoke nucleus sampling without top-k sampling: `top_p = 0.8, top_k = 0`. We follow this according to the official huggingface transformers generation guide[12]. Note that this is not an issue for greedy decoding, as it does not invoke stochastic sampling.

E.3 Scoring and Divergence Functions

In this section, we list extra implementation details regarding scoring and divergence functions. Primarily, we do not reintroduce information already found in 4.4 such as model sizes. Additionally, any settings that are set to their defaults are not mentioned here, as well as details on batching. In general, most metrics here were batched, and we take the mean of said batches. While we believe there will be some variation as truncated sampling

[12] https://huggingface.co/blog/how-to-generate#top-p-nucleus-sampling

methods like nucleus sampling are stochastic, we do not foresee major differences in our results.

GPT-2 Perplexity We use huggingface's `evaluate==0.4.1` to score perplexity[13], reporting the mean perplexities of the samples evaluated. We use the 774M `gpt2-large` parameter GPT-2 Large to calculate the mean perplexities.

SeqRep1-4 We calculate the SeqRep1-4(Welleck et al., 2019; Fu et al., 2021; Li et al., 2023a) from Li et al. (2023a)'s implementation[14]. Prior to doing so we modify it such that `ngramlist` also returns the unigram repetition, or otherwise, SepReq1. SeqRep1-4 requires the dependency `nltk==3.8.1`.

BERTScore We utilize the huggingface evaluate version of BERTScore (Zhang et al., 2019)[15], which requires the dependency `bert-score==0.3.13` and `evaluate==0.4.1`.

EntityF1 We implemented EntityF1 (Derczynski, 2016; Lee et al., 2022) by utilizing the `spacy==3.7.4` library in conjunction with the 125M parameter `en_core_web_trf`[16] model for Named Entity Recognition. As we specifically design EntityF1 to ignore repetition and consider the entity overlaps between all of the predictions(i.e. LM generated text) and labels, we utilize Python sets and intersections to calculate the true positives, false positives, and false negatives. For example, the number of true positives would be considered the `len` of the `list` of elements in the `intersection` between the `set` of entities in the prediction text and `set` of entities in the label text.

RankGen To our knowledge Wang et al. (2023)'s work has no public implementation of how they utilized RankGen (Krishna et al., 2022). Hence, we adapted a version as to what we thought was the most accurate implementation for use in scoring. We utilized the RankGEN repository[17] such that the `RankGenEncoder` and `RankGenGenerator` classes are used to encode and score generations based on their similarity to their prefixes. Additionally, because RankGen utilizes the dot product, we take the diagonal of the results to avoid scoring similarities between predictions and labels not within their respective pairs. To do so, we follow similarly in the official tutorial [18] up until the `generator` variable is defined. Aftward, we manually call the `RankGenGenerator.rankgenscorer` function, and take the `.diagonal()` of the first returned value, which we consider to be the RankGen score.

GTE Similarity We follow the instructions from the official model card of the 434M `Alibaba-NLP/gte-large-en-v1.5` (Li et al., 2023b) [19] embedding model to encode tokens. Our main difference is how we extract the embeddings: rather than computing the cosine similarity score between all labels and generations, we compute their corresponding cosine similarity scores individually and then average them in aggregate. We do so to remove unnecessary pairs, similar to the process followed in RankGen. To calculate the cosine similarity between generation-label pairs, we utilize Pytorch's `CosineSimilarity` class[20]. We note the primary addition with this class is a ϵ, which is a small value meant to prevent division by 0. We use the default value of `1e-8`.

MAUVE & Variants We primarily utilize a fork of MAUVE (Pillutla et al., 2021; Liu et al., 2021; Pillutla et al., 2023)[21], as the original forbids any models that do not have `gpt` or `bert` in their title[22] to be used. This rule makes it impossible to calculate ELECTRA-MAUVE (Clark et al., 2020; He et al., 2023). We used the aforementioned fork as it removed this limitation. Regarding hyperparameters, we keep every parameter at their defaults except for `kmeans_num_redo`, which we lower from 5 to 1. This is because MAUVE took the longest to calculate out of any divergence score, as it utilizes k-means clustering in its calculation, and we used this as a means for quicker evaluation. We generally saw the difference in scores was negligible and opted for this difference in hyperparameters in return for a quicker evaluation. MAUVE requires the dependency `scikit_learn==1.4.2` for its k-means clustering calculations.

[13] https://huggingface.co/spaces/evaluate-metric/perplexity
[14] https://github.com/gmftbyGMFTBY/Rep-Dropout/blob/main/repetition_dropout/utils/evaluation.py#L53
[15] https://huggingface.co/spaces/evaluate-metric/bertscore
[16] https://huggingface.co/spacy/en_core_web_trf
[17] https://github.com/martiansideofthemoon/rankgen
[18] https://github.com/martiansideofthemoon/rankgen?tab=readme-ov-file#using-rankgen
[19] Alibaba-NLP/gte-large-en-v1.5
[20] https://pytorch.org/docs/stable/generated/torch.nn.CosineSimilarity.html
[21] https://github.com/jackjyzhang/mauve
[22] https://github.com/krishnap25/mauve/blob/main/src/mauve/utils.py#L27