How does the Softmax Bottleneck affect LLM Generation?

James Ward

Master of Science
Artificial Intelligence
School of Informatics
University of Edinburgh
2025

Abstract

Large Language Models (LLMs) have been instrumental in the recent surge in AI, but are known to overestimate the probability of unlikely outputs, causing degradation of model-produced text. This problem, which we refer to as oversmoothing, has been theorised, but has no formal definition or metric. In this project, we provide the first formal metrics for oversmoothing. Additionally, it has been theorised that a common architecture component of LLMs, the softmax layer, imposes a constraint on the expressivity of model outputs, which leads to oversmoothing. In this project, under our novel metrics, we isolate and directly measure the impact of the softmax bottleneck for the first time. We use a framework where the model expressivity ranges exhaustively from very constrained by the bottleneck to completely unconstrained. We also evaluate these models in terms of their sample quality both qualitatively and quantitatively, to further highlight the impact of oversmoothing and the bottleneck. Additionally, we evaluate an existing hypothesis that decoding methods, known to improve LLM sample quality, do so through mitigating oversmoothing. Finally, we compare this expressivity constraint when imposed post-training instead of pre-training, and explicitly evaluate the expressivity of bottlenecked model outputs, to supplement our work and findings. Our findings indicate a trade-off between expressivity and efficiency in terms of the softmax bottleneck, and suggest that a lack of oversmoothing is a necessary but not sufficient condition for good-quality text generation.

Research Ethics Approval

This project was planned in accordance with the Informatics Research Ethics policy. It did not involve any aspects that required approval from the Informatics Research Ethics committee.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*James Ward*)

Acknowledgements

I would like to thank Andreas Grivas for all of his support, advice and suggestions throughout this project, and for being a friendly face for me to ask questions to.

I would also like to thank Antonio Vergari for all of his useful comments and suggestions for this project.

I'd like to thank Heather, Josh, Finn and Serena for being some of the most amazing people, that made this year worth it, and to thank Sai, Matthew, Alex B., Ethan, BeerSoc and the Edinburgh Revue for bringing me so much joy outside of academia.

Finally, I'd like to thank Alex H. as well as Mum, Dad and Charlotte for being there for me through this degree, and particularly the past few months. I genuinely wouldn't have made it without you <3.

Table of Contents

Chapter 1

Introduction

1.1 Motivation & Contributions

Large Language Models (LLMs) have achieved state-of-the-art performance in recent years on a wide range of tasks [1–3], including those in natural language processing (NLP) areas [4–7], where they exhibit an incredible ability to read and write text. However, LLMs overestimate the probability of less-likely tokens, a problem we will refer to as oversmoothing. For example, given the prefix 'wate', oversmoothing means a model is more likely to suggest an incorrect completion such as 'wateh' or 'wateo' and thus less likely to complete the word correctly as 'water'. This then means that models are less likely to generate the best tokens given a prompt, which causes text generated directly from an LLM to be of degraded quality [8]. We show an illustrative example of oversmoothing in Figure 1.1. Despite the current quality of LLMs, to the best of our knowledge, there is no formal definition of oversmoothing, and hence no formal study of it and methods to mitigate it. Therefore in this project, we ask the question **how do we measure oversmoothing?**

There are however multiple possible explanations for oversmoothing. One possible explanation is the 'softmax bottleneck' [9, 10]. This bottleneck limits the expressivity of typical LLM output layers by constraining the dimensionality of internal representations of data. The number of possible outputs for an LLM is typically an order of magnitude or two larger than its internal representation dimensionality [11]. Consequently, it is theorised that this lack of expressivity induces approximation errors, which cause oversmoothing. Indeed when the softmax bottleneck has been studied in previous literature, authors have focused on standard architectures with large vocabulary sizes, where the assumptions underpinning the softmax bottleneck are always met. As a

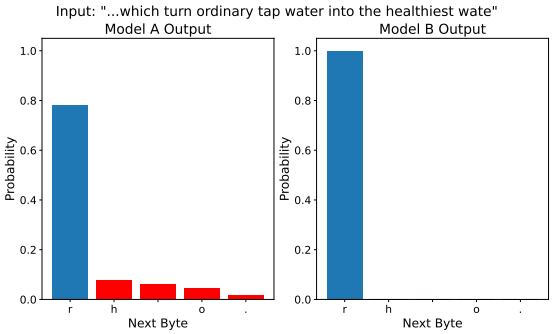


Figure 1.1: Oversmoothing - Model A is oversmoothed and so overassigns probability to completions such as ‘wateh’ or ‘wateo’ which are not prefixes of any English word. Model B does not do this and so assigns a higher probability to the (correct) prediction of ‘water’.

result, no study of the softmax bottleneck has been conducted using a set-up where the hidden dimensionality ranges from at least as large as the number of possible outputs, to significantly smaller than it, to provide a framework where the softmax bottleneck is directly isolated, measurable and varied enough to produce conclusive results. Consequently, it is not known whether, due to the softmax bottleneck, models are less expressive than needed, or if the output layers used in modern LLMs are more expressive than necessary, and thus inefficient. Thus in this project, we plug this gap in research by posing the question **how does the softmax bottleneck impact oversmoothing?** To answer this, we utilise byte-level LLMs where the number of possible outputs is small enough to train models with internal representations that are just as expressive as model outputs. This then means we can examine models which range from unconstrained to extremely constrained in terms of expressivity.

Elsewhere, when generating text from LLMs, a range of methods, known as decoding methods, are typically used, which have been shown to improve the text quality [8, 10, 12, 13]. However, the reasons behind their effectiveness are not fully understood. One hypothesis is that sampling heuristics directly mitigate the effects of the softmax bottleneck, thus reducing the level of oversmoothing exhibited by the model [10]. Hence, in this project we examine this hypothesis by answering the research questions **how does the softmax bottleneck impact LLM sample quality?** and **how do sampling heuristics impact oversmoothing?**

1.2 Structure

We firstly provide the background necessary to understand the work undertaken in this report and the context surrounding it in Chapter 2. Then, Chapter 3 outlines and justifies the LLMs in use in this study. Afterwards, recognising the aforementioned gap in research, in Chapter 4 we provide the first formal definitions of oversmoothing. We also perform the first investigation of the effects of the softmax bottleneck constraint where the impact of the bottleneck is isolated and varied extensively. This analysis is performed both in terms of oversmoothing in Chapter 4 and sample quality in Chapter 5. Furthermore, we properly test the existing hypothesis by Finlayson et al. on the effects of sampling heuristics in LLMs in Chapter 6. Finally, in Chapter 7 we evaluate a different method of implementing the softmax bottleneck, before we offer conclusions in Chapter 8.

Chapter 2

Background

2.1 Large Language Models

In this project we only consider autoregressive language models, which generate text that immediately follows a given input prompt. This generated text is then appended to the prompt and re-fed to the model to further generate successive text, with this process repeating until either a set amount of text or an end-of-sequence value is generated.

When text is given to an LLM, it is first parsed into tokens, which are the individual units that LLMs operate on. Tokens are usually words or sub-word units, but can also be individual characters or bytes; for example 'Tokenization' may be parsed into two units, 'Token' and 'ization'. The set of all possible tokens is the model's vocabulary, and the size of this can vary from $\sim$300 for byte-level models to $\sim 10^5$ for multi-lingual sub-word models. Once an input sentence is tokenised, the tokens are turned from discrete into continuous hidden vector representations, using an embedding matrix. The size of these vectors, known as the embedding dimensionality, is an important model hyperparameter, and the key to the softmax bottleneck. These embedding dimensions are typically in the thousands, which are dwarfed by some vocabulary sizes. In such instances, these hidden representations are referred to as low-rank.

LLMs also have an unembedding layer, also called the language-model head, which turns these hidden representations into an output that expresses, over all possible tokens, the relative likelihood of that token being predicted next. We can view this unembedding layer as the model measuring how similar its hidden representation of the prompt is to representations of each token in its vocabulary. The softmax operation is then used to turn this relative likelihood into a probability, and an output token is chosen using this probability. One simple method to choose an output token, called pure sampling,

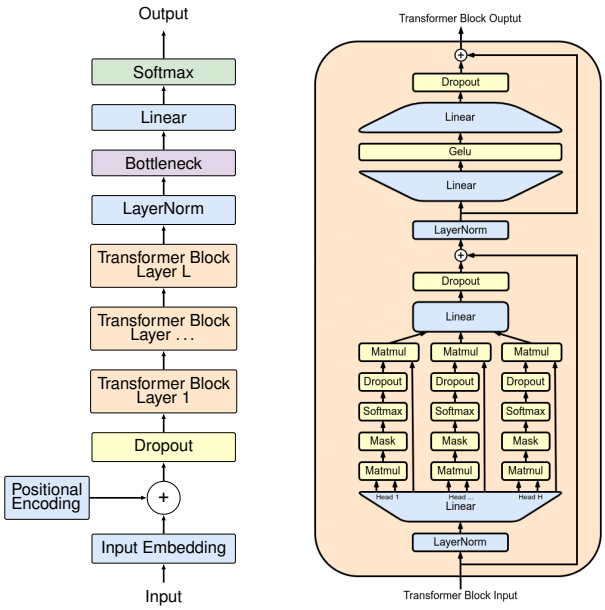


Figure 2.1: Bottlenecked GPT-2 Architecture and GPT-2 Transformer Block Architecture [15]. NanoGPT uses this architecture. The artificial bottleneck we impose down-projects into a lower-dimension, to reduce the rank of pre-language model head input.

is to sample directly from the model's output probability distribution, as a random variable. An alternative method, greedy decoding, is to take the mode or argmax of the probability distribution, the individual token with the highest probability. We explain other methods for choosing a token in Section 2.3.

For our work we focus on Transformer-based language models [14]. The core of this architecture is the 'attention' mechanism, that enables internal representations of text to change according to their context. This architecture is also parallelisable, which enables a significant scaling up of models and training datasets compared to prior methods; enabling state-of-the-art performance and motivating the name 'large' language models. We show a diagram of such an architecture in Figure 2.1.

2.2 Oversmoothing

Autoregressive LMs return probability distributions over possible tokens. We aim to train them so that they assign negligible probabilities to tokens that would form incoherent or grammatically incorrect sentences. However, LLMs often assign small, albeit overinflated, probabilities to such tokens [8]. As token probabilities must sum to one, this can become detrimental. Firstly, if a model with a large vocabulary assigns

probability mass to each unlikely token, this cumulative mass becomes significant and results in assigning significantly less mass to likely tokens, reducing the likelihood of a good-quality completion. Additionally, if we generated such a token, we would then feed our autoregressive LLM an input that would never have occurred under the true natural language distribution. In this case, the model is performing inference on input that lies outside the true support of its training data. In the literature, there are multiple proposed explanations for oversmoothing: the softmax bottleneck, mode-covering training and word-embedding unimodality. We explain each of these below.

2.2.1 Softmax Bottleneck

When an LLM with a hidden dimension d, is given a batch of n prompts, it computes hidden representations for each prompt $\mathbf{H} \in \mathbb{R}^{d\times n}$. Then the final layer applies a linear transformation, $\mathbf{W} \in \mathbb{R}^{v\times d}$ to obtain a matrix of logits, $\mathbf{WH} \in \mathbb{R}^{v\times n}$, where v is the size of the model's vocabulary. Finally, the softmax operation is applied to the logits to form a probability distribution over the vocabulary:

$$\text{softmax}(\mathbf{WH})_{ij} = \frac{\exp[(\mathbf{WH})_{ij}]}{\sum_{k=1}^{v} \exp[(\mathbf{WH})_{kj}]}. \tag{2.1}$$

If we take the log of this, we obtain the model's log probability matrix:

$$\mathbf{A}' = \log \text{softmax}(\mathbf{WH}) = \mathbf{WH} - \mathbf{J}\,\text{diag}\left(\log \sum_{i=1}^{v} \exp(\mathbf{WH})_i\right). \tag{2.2}$$

Here $\mathbf{J} \in \mathbb{R}^{v\times n}$ is the matrix of ones. Let $\mathbf{A} \in \mathbb{R}^{v\times n}$ denote the true log probability matrix for the given prompts. If we assume that $n > v$, then the rank of $\mathbf{A}$ is at most $v-1$. However, we observe that the rank of $\mathbf{A}'$ is at most $d+1$. Typically, $d \ll v$. For example, GPT-2 has a vocabulary size of $50,257$ but the largest GPT-2 model has a hidden dimension of $1,600$ [11]. Hence, we note that if $\mathbf{A}$ is of rank greater than $d+1$, then $\mathbf{A}'$ is a low-rank approximation of $\mathbf{A}$ and must necessarily introduce approximation errors [16]. Yang et al. hypothesised that, considering natural language is highly complex and context-dependent, it is indeed the case that $\mathbf{A}$ is high-rank. We refer to this as the **softmax bottleneck theory**, and this is the primary cause of oversmoothing that we investigate in this project.

Outside of oversmoothing specifically, the low-rank softmax bottleneck has been shown to reduce model expressivity; providing a limit to how low a bottlenecked model's cross-entropy loss can get [17], and preventing neural networks from assigning certain classes the highest probabilities [18]. What's more, the low-rank softmax

bottleneck allows an adversary to infer information about closed-source proprietary LLMs [19]. While there is consensus in the literature that low-rank log probability representations can be harmful, Parthiban et al. reported that a high-rank log-probability matrix is "neither necessary nor sufficient" for better model performance in recurrent neural networks (RNNs). However this work obtained a weak correlation between rank and performance and not absence of a correlation. We also note that much of the foundational literature is applied to RNNs which have since been superseded in NLP by Transformer-based LLMs whose performance far exceeds the older models [14, 21]. Finally, we note that in the literature the softmax bottleneck refers to two uses of the softmax in Transformer-based LLMs: in the output layer, and in the self-attention mechanism [22]. In this project we focus exclusively on the former.

2.2.2 Mode-Covering Training

It has also been theorised that the language model training process causes oversmoothing. Typically, LLMs are trained using cross-entropy loss, which is equivalent to training with forward KL-divergence when using a one-hot encoding of text:

$$\mathrm{KL}(p,q) = \sum_x p(x) \log \frac{p(x)}{q(x)}. \tag{2.3}$$

Forward KL-divergence is known to be mode-covering, meaning if a model assigns zero probability to observed tokens, it will be assigned an infinite loss. This results in a model that is trained to hedge against observing unexpected tokens by assigning non-zero probability mass to all tokens [12]. Indeed, because of this, Hewitt et al. viewed LM output distributions as a mixture of the true distribution and a uniform-like smoothing distribution, which necessarily deviates from the true distribution we want to learn.

2.2.3 Unimodality & Linear Dependencies

LLMs turn contexts into continuous vectors, known as embeddings. The distributional hypothesis states that words that occur in similar contexts have similar meanings [23], and this also applies to word embeddings: words with similar vector representations have similar meanings as well. These word embedding similarities become problematic, however, when the distribution for a given prompt is multi-modal; there are several distinct, specific, but equally likely next tokens. As the LLM models this context with a

single hidden vector, it struggles to find a representation that is equally close to these tokens but distinct from similar yet differing ones. For example, given the prompt 'After debating whether to bow to the woman or the king first, the jester decided on the', an LLM may produce a representation that is similar to both 'woman' and 'king', making 'queen' or 'man' likely next tokens, even though they are very unlikely given the context [24] - an instance of oversmoothing.

These word embeddings also give rise to word analogies, where directions in the hidden vector representations of words can encode semantic information. For example, word embeddings for 'man', 'woman', 'king' and 'queen' may be such that 'queen' - 'king' $\approx$ 'women' - 'man'. These word analogies suggest that the word embeddings have an approximate form of linear dependency between them, and thus lie in a lower-dimensional subspace of the entire space of vector representations. These linear dependencies would mean the expressivity of the model's representations is limited, which Chang et al. proved formally. We may view this hypothesis, which we call the unimodality issue, as an alternative form of rank-constraint in the model's representations, separate to the dimensionality of the space the representations are embedded in, which will also induce approximation errors.

2.2.4 Which explanation is most likely?

Overall, it is likely that LLM oversmoothing is caused by some combination of all three explanations. In this project we only consider the softmax bottleneck, and we evaluate the extent to which it does impact LLM expressivity, oversmoothing and sample quality. We leave the impact of the others to future work. However, we note that under this hypothesis, even if we fully alleviate the impact of the softmax bottleneck, models in this project may still oversmooth.

2.3 Decoding Methods

Choosing a token from an LLM's output distribution is a surprisingly complicated task [25]. The goal of LLM decoding is to output the series of tokens which produce the most likely string of text overall under the model, however computing this exactly is intractable, given the exponential number of possible sentences for any given length. Hence approximations are needed and there are many ways to do this, known as sampling heuristics or decoding methods. The most obvious approach is to always take

the most likely token to continue any given prompt. This is greedy decoding. As this is a greedy heuristic, we may improve upon it by temporarily storing multiple possible continuations to expand upon. This approach is known as beam search [26]. Both approaches are deterministic methods. However, these maximum likelihood methods are known to result in text that is degenerate as it is repetitive [13].

Contrastingly, in order to introduce more creativity to the model's samples, we could sample tokens as random variables directly from the output probability distribution. However this approach, known as pure or ancestral sampling, typically produces text that is also degenerate, but because it is noisy and nonsensical [8]. Stochastic decoding methods aim to strike a balance between these two extremes, often by inflating the probabilities of the most likely tokens and decreasing the probabilities of the tail of the distribution in some way, behaviour which will directly reduce oversmoothing in the output distribution.

One of the most common decoding methods is temperature sampling, where the logits are multiplied or divided by a constant before the softmax layer is applied, to make the output distribution flatter or spikier. Another method is top-k sampling [13]. In this method, logits for tokens outside of the k most likely tokens are masked before the softmax operation, producing a distribution where these tokens receive zero-probability, which we sample from. Another approach is nucleus, or top-p sampling [8]. In this approach, we consider the set of the fewest tokens with a combined total probability mass under our model exceeding some threshold p. We call this set the **nucleus set**. When sampling, we mask the logits of tokens outside of this set before applying the softmax, similar to top-k. However unlike top-k, the number of tokens the model may sample from changes dynamically. Alternative strategies decide the truncation set based on an entropy-dependent threshold [12], information-theoretic content [27] or maintaining a constant level of perplexity [28].

Although these methods claim significant improvements over greedy decoding and pure sampling, they all suffer from one key weakness: namely, the sets of truncated and non-truncated tokens must be linearly separable in the model's output distribution. For example, the true language distribution may assign a non-zero probability to one token and a zero-probability to another, however a learned model may assign the true zero token a higher probability than the other. In this case, no threshold will accurately separate the true zero tokens from the rest. Finlayson et al. proposed a sampling method that does not suffer from this limitation, namely BAT sampling. This method relies directly on the assumption that the low-rank softmax bottleneck produces approximation

errors in the model's output distribution, and solves a linear program to only sample tokens that are provably in the support of the true language distribution. Although non-linearly separable, this method is computationally expensive due to the cost of solving this linear program (Table I.1) and requires approximations to become tractable for large vocabularies.

Overall, we observe that decoding methods generally make generating the most likely tokens under a model even more probable, which we can intuitively interpret as reducing the oversmoothing present in the model. In this report we will, for the first time, empirically analyse the impact of these decoding methods on oversmoothing.

2.4 SVD & Rank

Given a matrix $\mathbf{M} \in \mathbb{R}^{n \times m}$, SVD [29] decomposes $\mathbf{M}$ into $\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$, where $\mathbf{U}$ and $\mathbf{V}$ are orthonormal and $\mathbf{\Sigma}$ is a diagonal matrix, with non-negative singular values $\sigma_1 \geq \sigma_2 \geq ...$, which correspond to how much the matrix $\mathbf{M}$ varies in that dimension. Hence, for a matrix of hidden representations for a given LLM, if there are many small singular values, this indicates a lack of expressivity in the hidden representations. To explicitly quantify this expressivity, there are two ways to calculate the 'effective rank'. The first approach considered is to count the number of singular values above some threshold, defined as a multiple of the largest singular value. The alternative approach is to obtain a probability distribution using the normalised singular values, and to then compute the Shannon entropy of this distribution [30].

Much of the work on the softmax bottleneck has been built on the theoretical assumption that representations of natural language are high-rank and that learned representations of language are low-rank due to the bottleneck. It is thus useful to empirically analyse the effective rank of representations obtained by LLMs, to supplement this theoretical analysis. Indeed, Parthiban et al. did this to to evaluate the softmax bottleneck in Recurrent Neural Networks [20]. While this work was useful for tokenised LLMs, to our knowledge no evaluation of the effective rank of of representations for byte-level LLMs has been conducted, and it is not known what effective rank suffices for strong performance in byte-level LLMs. We answer these questions in this report.

Chapter 3

LLM Preliminaries

3.1 Architectures & Datasets

To investigate the impact of the softmax bottleneck directly, we trained a family of LLMs that are identical except for the dimension of their LM head input. We only considered byte-level LLMs, as they allow for a full-rank model to be fitted with a hidden dimension at least as large as the vocabulary size, providing a context where the assumptions foundational to the softmax bottleneck theory are not met. This then allows us to properly investigate the impact of the softmax bottleneck's low-rank constraint, as we have models where the low-rank constraint is not imposed, moderately imposed and extremely imposed.

This requires us to identify a suitable architecture. The main model architecture selected was an open-source NanoGPT architecture [31–33] based on GPT-2, which was selected due to its simple and open source codebase, allowing easy setup of preliminary models of varying sizes depending on computational limits. The open-source code also allows for easy adaptation of model architectures to impose an artificial bottleneck before the softmax layer. The output dimension of the NanoGPT model is the set of UTF-8 bytes, of which there are 256 values [34] (see Figure A.1).

In the original repository, the NanoGPT models were trained on the Tiny Shakespeare dataset [35]. However, for this project, we utilised the Fineweb dataset [36], so that model output quality is more easily discernible to a contemporary English speaker and so that model outputs are comparable with existing pre-trained models. Furthermore, the larger dataset size allows for more extensive evaluation and facilitates a scale of model that fully utilises the computational budget available while maintaining a strong performance and avoiding overfitting.

3.2 Ground-Truth Models

For this project, in order to determine what tokens our models should have assigned the most probability mass to, we need some notion of a 'ground truth'. As the true distribution of natural language is unknown, we opt to obtain a high-quality 'ground truth' LLM as a proxy for the true distribution. The easiest approach computationally is to use a pre-trained model. We chose the EvaByte model [37], as this model is trained on English-heavy datasets and is a decoder-style architecture, making it suitable for comparison with our NanoGPT models on the task of English-language next byte prediction. Additionally, the EvaByte model performs comparably to open-source tokenizer-based LLMs. We note that the hidden dimension of the EvaByte model is 4,096, which greatly exceeds the output dimension of 320, therefore this model does not have a low-rank bottleneck layer. We also note that the output vocabulary of EvaByte is a superset of the 256 UTF-8 bytes that the NanoGPT models operate on, with 64 additional special tokens, such as the beginning-of-sentence token. For comparison purposes, we truncate the logits of our EvaByte model and take the softmax over the 256 UTF-8 byte tokens when running inference. We justify this as our Fineweb dataset never contains any of the 64 special tokens, and experimentally, we observe that EvaByte does not assign these tokens high probability for our Fineweb prompts. One caveat of this is that it requires pre-processing of prompts by adding a beginning-of-sentence token; however, all results thereafter are directly comparable with the NanoGPT models.

Additionally, it should be noted that although EvaByte was trained on Fineweb data, it was also trained on data from other datasets, namely Dolma [38], The Stack [39] and DCLM-Baseline [40]. Consequently, EvaByte is trained on a different distribution than our bottlenecked NanoGPT models, with training data that includes code. This means the outputs from the two families may not be exactly comparable. To assess the impact of training on different data with a different architecture, we also trained a 300M NanoGPT model on Fineweb, to obtain a strong model that has learned the same distribution as our bottlenecked models. However, as we see, this model does not achieve the same performance as EvaByte on our validation data, and hence may be a ground-truth which, when compared to EvaByte is further from the true distribution of language. Considering the comparative benefits of both models, we perform our oversmoothing analysis in terms of both EvaByte and this NanoGPT-300M model.

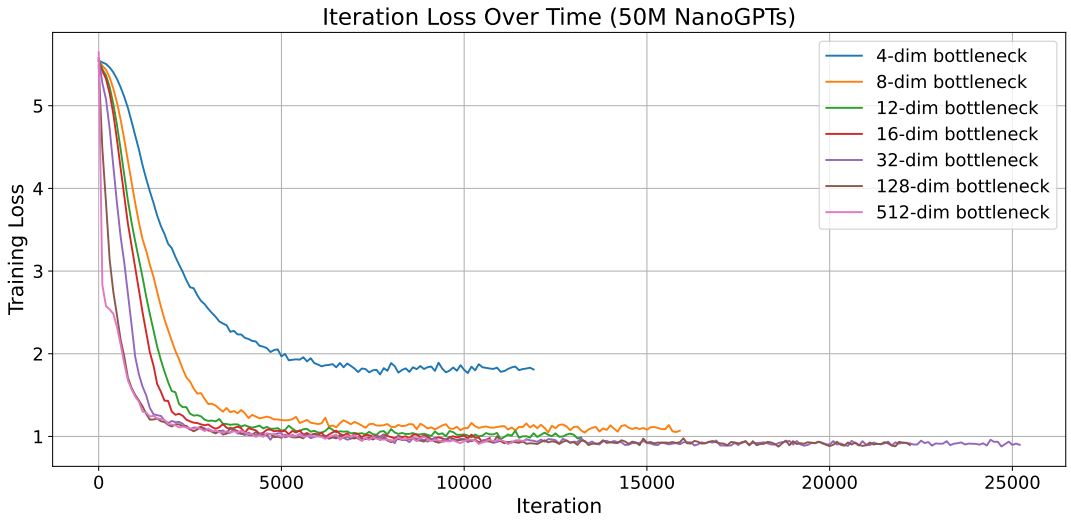


Figure 3.1: Training Losses of 50M NanoGPT models. We observe separation between very low-rank models, but not between moderate- and full-rank models.

3.3 Model Training

In order to directly measure the impact of the softmax bottleneck, we train a series of models with bottlenecks of varying dimensionalities imposed, enabling an extensive investigation where the impact is isolated and measurable. Given our computational budget, we trained a family of 50M NanoGPT models, each with a hidden dimension of 512, and 16 transformer blocks with 16 attention heads [14][1]. As shown in Figure 2.1 hidden representations are down-projected to a specified dimension before the LM head, forming a softmax bottleneck layer of a specific rank. In order to make training comparable between models, and given our computational constraints, bottlenecked models were all trained using identical convergence criteria (training until validation loss fails to decrease within 5 iterations). We deemed this training process to be a good representation of the circumstances in which these LLMs would be independently obtained; as training many LLMs is expensive, only one model is trained, with convergence being the only criterion with which a checkpoint would be selected. All models were trained on a 2.15GB subset of Fineweb, using a 90-10 training-validation split. The training losses of the models are shown in Figure 3.1. This subset of Fineweb is the same dataset we use to train our 300M NanoGPT ground truth model.

[1]Models use a 1,024-byte block, trained used AdamW [41] $\beta_1 = 0.9, \beta_2 = 0.95$, a weight decay of 0.1 and dropout rate of 0.1, an initial and minimum rate of 3×10^{-4} and 3×10^{-5}, 1000 warm-up iterations, batch size of 16, 16 gradient accumulation steps, for a maximum of 100,000 total iterations. Evaluations performed every 250 iterations.

Bottleneck	Validation Loss	Accuracy (%)
NanoGPT 50M BN-4	1.776	57.2
NanoGPT 50M BN-8	1.073	70.5
NanoGPT 50M BN-12	0.992	71.2
NanoGPT 50M BN-16	0.922	72.6
NanoGPT 50M BN-32	0.885	72.9
NanoGPT 50M BN-128	0.887	72.7
NanoGPT 50M BN-512	0.933	71.7
NanoGPT 300M	0.772	76.2
EvaByte	**0.710**	**79.8**

Table 3.1: Model validation losses and argmax accuracies on Fineweb.

3.4 Performance

We show the validation losses of our models in Table 3.1. In this table, we also show the accuracy of each model's most likely token (argmax) when compared to the ground truth for that byte, using a separate held-out validation set. Both of our ground truth models, as expected, outperform all other models in terms of validation loss and argmax accuracy. As mentioned EvaByte, a larger model trained on a significantly larger dataset outperforms the NanoGPT ground truth model. As we see from the plots, surprisingly, when the models converged in training, there was little difference between the models with a pre-LM head hidden dimension of 16 or greater. We saw a separation between models in terms of training loss and validation loss with bottleneck dimensions less than 16. As we restricted the bottleneck dimensionality beyond this point, as one would expect, the more constrained models performed worse in terms of final training and validation loss. For argmax accuracy, the results follow the same trend as the validation loss, with the 4-dimensional bottleneck obtaining an argmax accuracy noticeably distinct from the other models. Overall the ordering of models is the same for validation loss and argmax accuracy. We note however that there is more separation between models in terms of validation loss compared to argmax accuracy. This suggests that moderately low-rank models are generally able to predict bytes accurately, although with less confidence than higher-rank models.

Chapter 4

Oversmoothing Metrics

4.1 Defining Oversmoothing

4.1.1 What should a metric capture?

In order to analyse the effect of the softmax bottleneck on oversmoothing, we must first ask the question **how do we measure oversmoothing?** At a high level, we define oversmoothing as the **extent to which a language model 'leaks' probability away from tokens it should be likely to predict**, to the tail of its output distribution - the tokens it should assign very low probability to. Equivalently, we can think of oversmoothing as the extent to which the model overinflates the probabilities of generating very unlikely tokens. To rigorously define and measure oversmoothing for the first time, it requires two components: firstly **what tokens should the model have assigned high probability**, and secondly, **how much less probability mass did the model assign to those tokens than it should have**.

To answer these questions we use a strong 'ground truth' byte-level LLM. We then analyse a smaller, bottlenecked model in terms of oversmoothing using this ground-truth model as a reference. We first identify the tokens that the bottlenecked model should have assigned high probability, given a set prompt, using the 'nucleus set' [8] (outlined in Section 2.3) from our ground-truth model. Next, we evaluate the bottlenecked model on the same prompt to obtain the probability distribution we wish to analyse in terms of oversmoothing.

In terms of formal definitions, let p_{gt}, p_{bn} be the probabilities returned by our ground truth and bottlenecked models respectively. Let V be the set of all possible output tokens (256 bytes in our case). We then define the nucleus set of our ground truth model, V^*_{gt} as

$$V_{gt}^* = \arg\min_{S \subseteq V} |S|$$

$$\text{s.t.} \sum_{w \in S} p_{gt}(w|\text{prompt}) \geq \tau$$

for some hyperparameter τ, which controls the size of the nucleus set.

4.1.2 Tail Token Overassignment

Once we have obtained our set of low-probability tokens, defined as the complement of our ground truth model's nucleus set, $(V_{gt}^*)'$, we may then analyse how much our bottlenecked models overassign probability to each of these tokens and take the sum of this quantity over the entire set. This is **tail token overassignment**. Formally, we define this as

$$\text{TTO}(p_{bn}, p_{gt}) = \sum_{w \in (V_{gt}^*)'} \max\left[0, p_{bn}(w|\text{prompt}) - p_{gt}(w|\text{prompt})\right]. \tag{4.1}$$

Note that here we take the maximum of this difference and zero as we are only considering probability **over**assignment, and so we disregard low-probability tokens our model has not leaked mass to. We note that, although this metric directly captures what we may regard as oversmoothing, it has a limiting assumption, which is that it considers tokens individually. However, we wish to study oversmoothing in terms of how it causes probability to leak away from high-probability tokens in general, instead of how it causes probability to leak into specific low-probability tokens. This allows our metric to take into account interactions between the over- and under-assignment of low-probability tokens.

For example consider a distribution over 3 tokens, with a ground truth of $[0.8, 0.2, 0.0]$ and a nucleus set threshold of 0.7. Suppose we obtain an output from model A of $[0.7, 0.2, 0.1]$ and an output from model B of $[0.8, 0.0, 0.2]$. In that case, the nucleus set is the first token and our metric will give levels of oversmoothing as 0.1 and 0.2, respectively, even though model B assigns a greater probability to the nucleus set. Hence, we may wish to measure oversmoothing in terms its impact on the mass assigned to low-probability tokens as a whole.

4.1.3 Nucleus Set Mass Difference

To measure oversmoothing in terms of its impact on the probability assigned to unlikely tokens as a whole, we can simply calculate how much more total probability mass our

bottlenecked model assigns to the set of low-probability tokens compared to the mass assigned by our ground-truth model. Formally, we define this as:

$$\text{NSMD}(p_{bn}, p_{gt}) = \sum_{w \in V^*_{gt}} [p_{gt}(w|\text{prompt}) - p_{bn}(w|\text{prompt})]. \tag{4.2}$$

We call this metric the **nucleus set mass difference**. We note that, due to the properties of probability distributions, summation of underassignment over the nucleus set or of overassignment over its complement give equivalent results. Proof of this can be seen in the appendix.

This metric now captures oversmoothing when viewing the low-probability tokens as a block in their entirety; however, we note that this metric can be negative. Such a property raises the question: what does negative oversmoothing mean? This means that our learned model's distribution is 'spikier' or more concentrated in the nucleus set, than our ground truth model. For this project we do not concern ourselves with this behaviour and so, we may wish to outline a metric that can only be positive.

4.1.4 Total Variation Distance

A natural way to impose non-negativity is to take the absolute value of our differences before we summate. In fact if we take this sum over the entire probability distribution this forms a well-known divergence between distributions, **total variation distance**. Hence, we can expand upon our pre-defined metrics by simply using total variation distance:

$$\text{TV}(p_{bn}, p_{gt}) = \frac{1}{2} \sum_{w \in V} |p_{gt}(w|\text{prompt}) - p_{bn}(w|\text{prompt}))|. \tag{4.3}$$

Although this metric provides non-negativity and is a more standard, well-known metric, it also has drawbacks. Namely, this divergence no longer captures just oversmoothing, as we are no longer only considering how mass is leaked away from high-probability tokens to lower-probability ones; we are instead measuring how the probabilities differ both in terms of under- and overassignment. Thus, if a bottlenecked model exhibits high total variation distance, it may not necessarily be oversmoothed; it may still assign probability to 'good' tokens, just in a different manner. Additionally, this metric assigns equal weight to all tokens; however when sampling from a distribution, we may care more about discrepancies and the effects of oversmoothing on primarily the most likely tokens, where fluctuations in probability are more likely to impact generated samples. This weighting of tokens can be achieved by utilising another well-known divergence that we have already mentioned.

4.1.5 KL Divergence

The **Kullback-Leibler divergence** measures the distance between two probability distributions, with higher-probability events (or tokens in our case) assigned more weight for the divergence. Formally for our models, this is defined as

$$\mathrm{KL}(p_{bn}, p_{gt}) = \sum_{w \in V} p_{gt}(w|\text{prompt}) \log \frac{p_{gt}(w|\text{prompt})}{p_{bn}(w|\text{prompt})}. \tag{4.4}$$

Once again however, this metric no longer measures just oversmoothing of a model's distribution, with discrepancies in the shape of likely token probabilities having a prominent effect. Additionally, KL-divergence is unbounded, with this divergence blowing up to infinity in cases where the bottlenecked model misassigns zero probability to a token. Such behaviour may not be desirable when measuring only oversmoothing and renders values no longer directly interpretable as how much probability mass a model has misassigned.

4.1.6 Total BAT-reject Set Mass

To capture oversmoothing, we need to examine the ground truth model's nucleus set to obtain our set of 'likely' tokens that the model should have assigned probability to. This has several limitations. Firstly, this requires a strong ground truth model as a proxy for the true distribution of natural language, but also the nucleus set requires the good tokens to be linearly separable from the lower probability tokens, using some threshold. This assumption does not necessarily hold as highlighted by Finlayson et al. Inspired by this work, we outline an oversmoothing metric that takes the total probability mass assigned to tokens that do not provably lie inside the true support of the language distribution our model learns. We call this metric the **total BAT-reject set mass** (TBAT).

At a high-level, basis-aware threshold (BAT) sampling states that given a model's output probability distribution $\widehat{\mathbf{p}} = p(\mathbf{w}|\text{prompt})$, unembedding matrix $\mathbf{W}$ and a token i, if there exists no probability distribution $\mathbf{q}$ such that $\mathbf{W}^T\widehat{\mathbf{p}} = \mathbf{W}^T\mathbf{q}$ and $q_i = 0$ (and some other conditions we leave to the appendix), then token i must necessarily be in the true support of the distribution we are trying to learn, hence token i has true non-zero probability. To identify this set of tokens for a given model and prompt, we must solve a linear problem for each token. We leave technical details for the appendix and simply consider the TBAT metric to be the total mass a model assigns to tokens which are not provably in the support of the true language distribution.

This metric has the advantage that the oversmoothed tokens need not be monotonic in the model's output distribution, and it also highlights tokens that have been oversmoothed due to the softmax bottleneck specifically. However, one core limitation of this metric is that the dimensionality of the bottleneck impacts the freedom to find solutions to the linear program one must solve for BAT. Hence, although this may be expected, for sufficiently high-rank models this metric fails to rule any tokens out of the true support and so will provide zero values for most or all of the prompts provided. We can see this formally with the rank-nullity theorem [42], which implies that when our embedding matrix is full-rank the solution to $\mathbf{W}^T\widehat{\mathbf{p}} = \mathbf{W}^T\mathbf{q}$ is uniquely determined, hence no tokens will ever be ruled out, as our model with finite logits will always assign non-zero probabilities. Additionally, this metric only considers oversmoothing where the tokens may not have had true zero-probability, however we would still like to measure the oversmoothing present when tokens are assigned a low, but non-zero probability in the true language distribution. Thus, this metric does not capture oversmoothing in general.

4.2 How does the Softmax Bottleneck impact Oversmoothing?

Equipped with our models and our metrics, we can now ask the question **how does the softmax bottleneck impact oversmoothing?** In order to do this, we take a 2.15GB subset of Fineweb distinct from our training and validation sets and evaluate our models on a set of 100 randomly sampled subsets, considering model outputs for 500 contiguous bytes for each subset, with the prompts starting after end-of-sentence demarcations, to provide an extensive evaluation of model behaviour, in contexts reflective of real-world usage. We then examine how the different bottleneck strengths impact the level of oversmoothing on these samples both qualitatively and quantitatively. We show averages in Tables 4.1 and 4.2. Note that due to its computational complexity (Table I.1), TBAT was evaluated on a subset of 100 randomly selected prompts from our evaluation dataset.

We observe that the first 4 metrics all show similar trends. Namely, the level of oversmoothing is highest for the most bottlenecked model, and the level of oversmoothing decreases monotonically with the dimensionality of the bottleneck, to 32 dimensions. After this, the level of oversmoothing does not vary significantly when the bottleneck is

Bottleneck	EvaByte		NanoGPT 300M		TBAT
	TTO	NSMD	TTO	NSMD	
4	0.432 ± 0.291	0.424 ± 0.292	0.386 ± 0.273	0.376 ± 0.274	0.273 ± 0.314
8	0.178 ± 0.235	0.170 ± 0.235	0.124 ± 0.175	0.114 ± 0.175	0.069 ± 0.127
12	0.153 ± 0.219	0.146 ± 0.219	0.097 ± 0.153	0.088 ± 0.153	0.028 ± 0.066
16	0.129 ± 0.201	0.122 ± 0.202	0.072 ± 0.126	0.063 ± 0.127	0.017 ± 0.044
32	0.118 ± 0.192	0.112 ± 0.193	0.059 ± 0.113	0.052 ± 0.115	0.0039 ± 0.0132
128	**0.118 ± 0.194**	**0.111 ± 0.195**	**0.059 ± 0.115**	**0.051 ± 0.117**	$2.5 \times 10^{-6} \pm 6.9 \times 10^{-6}$
512	0.136 ± 0.208	0.129 ± 0.209	0.076 ± 0.138	0.069 ± 0.139	**0.00 ± 0.000**

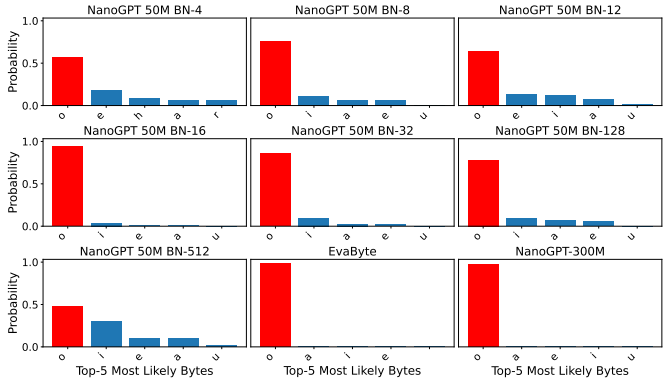
Table 4.1: Bottlenecked Oversmoothing Results for next-byte prediction of FineWeb, under TTO and NSMD compared to EvaByte and Nano-GPT 300M, and under TBAT. Values are mean ± standard deviation. Lower values indicate less oversmoothing. Bold indicates lowest level of oversmoothing per metric.

Bottleneck	EvaByte		NanoGPT 300M	
	TV	KL	TV	KL
4	0.477 ± 0.286	1.170 ± 1.389	0.438 ± 0.269	0.994 ± 1.164
8	0.242 ± 0.262	0.486 ± 0.860	0.191 ± 0.212	0.321 ± 0.583
12	0.223 ± 0.248	0.408 ± 0.743	0.168 ± 0.194	0.243 ± 0.468
16	0.200 ± 0.235	0.341 ± 0.649	0.141 ± 0.171	0.176 ± 0.356
32	**0.189 ± 0.226**	**0.305 ± 0.618**	**0.127 ± 0.159**	**0.140 ± 0.301**
128	0.190 ± 0.228	0.310 ± 0.650	0.128 ± 0.161	0.141 ± 0.307
512	0.208 ± 0.239	0.348 ± 0.679	0.150 ± 0.179	0.181 ± 0.365

Table 4.2: Bottlenecked Oversmoothing Results for next-byte prediction of FineWeb, under TV and KL compared to EvaByte and Nano-GPT 300M. Values are mean ± standard deviation. Lower values indicate less oversmoothing. Bold indicates lowest level of oversmoothing per metric.

further weakened to 128 dimensions, but does increase slightly at 512 dimensions. It is interesting to note the discrepancy between the metrics for which of the 32 and 128 bottlenecked models is better in terms of oversmoothing. We note that the 32 bottlenecked model is better under TV and KL divergences, but the 128 bottleneck model is better under TTO and NSMD, however considering the size of standard deviations this difference is not statistically significant.

This result is surprising, as it suggests that models oversmooth as much with a moderate-rank constraint, as when they have the potential for full-rank representations. However, the results suggest that extremely restrictive bottlenecks do result in pronounced oversmoothing. When combined with the training and validation loss results, a strong correlation is observed between loss and oversmoothing. Conceptually, we



BN	TTO	NSMD	TV	KL	TBAT
4	0.399	0.399	0.399	0.471	0.013
8	0.220	0.220	0.220	0.200	0.00011
12	0.329	0.329	0.329	0.343	0.00014
16	**0.041**	**0.039**	**0.041**	**0.029**	0.00011
32	0.122	0.121	0.122	0.098	2.5×10^{-5}
128	0.193	0.193	0.193	0.167	**0.000**
512	0.502	0.502	0.510	0.671	**0.000**

Figure 4.1: Model probabilities and corresponding levels of oversmoothing for the prompt: "It's an insult to the democratic process. In 1996, Michigan's voters overwhelmingly approved a pl that gives sole authority to Michigan's NRC for setting wildlife management policies. Our democratic system uses a system where v" . Correct token probability is in red. TTO, NSMD, TV and KL values are means of comparison with EvaByte and NanoGPT 300M.

provide the following explanatory hypothesis: as models that oversmooth assign lower probabilities to their most likely tokens, they will incur a greater loss when these tokens occur, compared to a more confident model. In addition to this observation, we note that under these metrics, all of our models oversmooth, including our full-rank model. Thus, avoiding the low-rank bottleneck is not sufficient to prevent oversmoothing; it requires an effective training procedure and potentially mitigation of other possible causes of oversmoothing.

Elsewhere, we note that the level of oversmoothing under TBAT is strictly monotonic with bottleneck rank, with the full-rank model exhibiting absolutely no oversmoothing. This result is expected, and is evidence that this metric is informative for some low-rank models, but is not in general useful.

We examine the outputs of our models for one specific prompt to elucidate the oversmoothing results, in Figure 4.1. Firstly, we note that our ground truth models are very confident for this prompt, which matches our intuition and understanding of the context - the next word is likely to be 'votes' or 'voters'. However, we note that our bottlenecked models are all less confident in predicting this byte. We also note the behaviour of the model with a bottleneck of 4; it assigns significant probability mass to other tokens in the distribution, which it should assign a probability of zero. As such, the correct and logical next byte 'o' is given a lower likelihood, and the model is more likely to produce an illogical or ungrammatical sample, highlighting the risk of

oversmoothing. We observe that the 3 best performing models here, under the first 4 metrics, and the ones that visually assign the least mass outside of 'o' are the bottlenecks of 16, 32 and 128; the three models with the lowest average validation loss. However, it is also interesting to note that the 16-dimensional bottleneck outperforms the 32- and 128-dimensional bottlenecks, which perform best on average. Also, the full-rank model with a hidden dimension of 512 exhibits the highest level of oversmoothing in this instance, although it performs comparably on average. This is evidence that the level of oversmoothing is prompt-dependent even for models that perform comparatively on average.

4.3 How does Oversmoothing vary between Tokens?

As an addendum to the above analysis, we can ask the question **how does oversmoothing differ when a model is asked to predict different tokens?** As we found that all oversmoothing metrics, except for TBAT, give similar results, we simply use NSMD for this analysis. Then, to answer this, we show, for the best and worst performing bottlenecked models under our oversmoothing metric, the average level of oversmoothing per byte, along with the frequency of that byte in the evaluation set in Figure 4.2.

From these results, we see that for the 4-dimensional bottleneck, although there is variation, oversmoothing tends to increase as token frequency decreases. For the 128-dimensional bottleneck, on average, minimal oversmoothing is exhibited when expected to predict frequent tokens, while for medium-frequency tokens the level of oversmoothing weakly increases before temporarily declining and then rising prominently for the least frequent tokens. At the tail-end there is significant variability, due to prompt dependency and the small sample size of such prompts. But for both models a difference in behaviour between extreme ends is pronounced; when receiving a prompt for which they should identify a space as most likely, both models oversmooth far less compared to when a '%' should be predicted.

Additionally, we analysed the distribution of argmaxes for the various models over these samples, examining the relationship between how often models should and do predict certain tokens. The distributions of model argmaxes, overall and at either end of the relative frequency distribution, are shown in Figure 4.3. From these results, we see that, as expected, the best-performing models, EvaByte and the strongest NanoGPT models predict tokens with similar relative frequencies to the tokens' actual occurrences. However this is not the case for the most bottlenecked model. The model with a

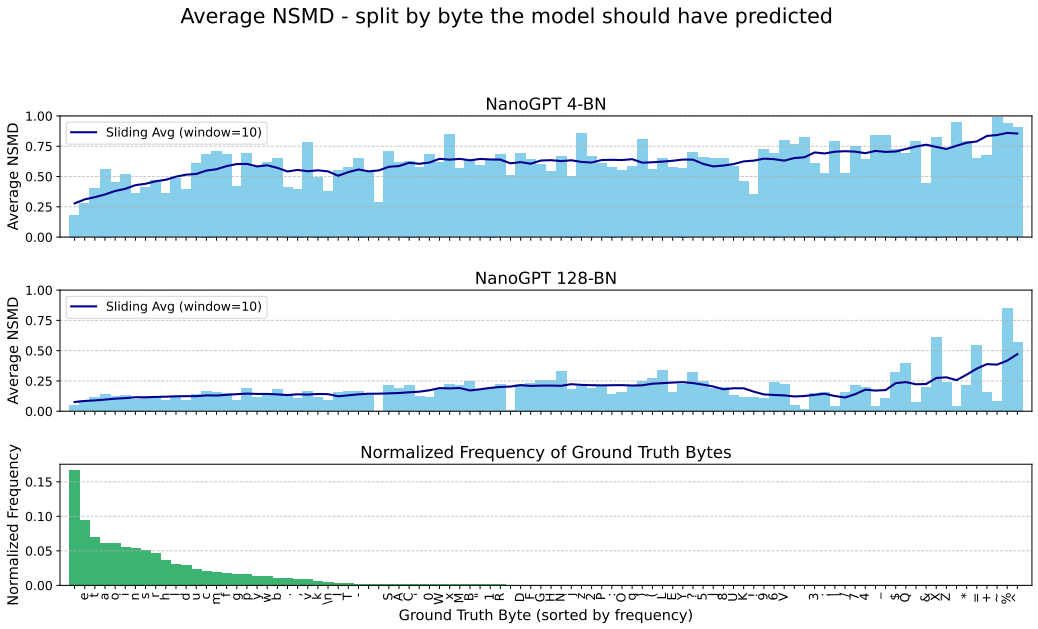


Figure 4.2: Average NSMD (comparing against EvaByte) for 4 and 128 bottlenecked NanoGPTs split by byte. Averages are over each occurrence of the byte the model should predict, and the frequency of occurrences are shown at the bottom.

bottleneck of size 4 often assigns the most probability to the most commonly occurring bytes more than it should, although this is not the case for all of the most commonly occurring bytes. On the other extreme however, the most bottlenecked model never assigns the most probability mass to any of the 10 least frequent bytes (with non-zero occurrences) given a prompt. This suggests that the softmax bottleneck causes the model to have insufficient confidence in predicting infrequent tokens, compared to a more expressive model, which aligns with Figure 4.2. Here we observe that, in general, oversmoothing per byte is higher for the more bottlenecked model for all tokens, including infrequent ones. However, for the most infrequent tokens, almost all of the probability mass is oversmoothed. We corroborate this result by exploring the unargmaxibility of the classes learned for each of the bottlenecked models. The results are shown in Tables B.1 and B.2, and we see that indeed the low-rank models are never able to assign the least frequent tokens the highest probability, regardless of the input. This is further evidence that the bottleneck reduces model expressivity, induces oversmoothing and hence causes a lack of confidence in predicting less frequent tokens.

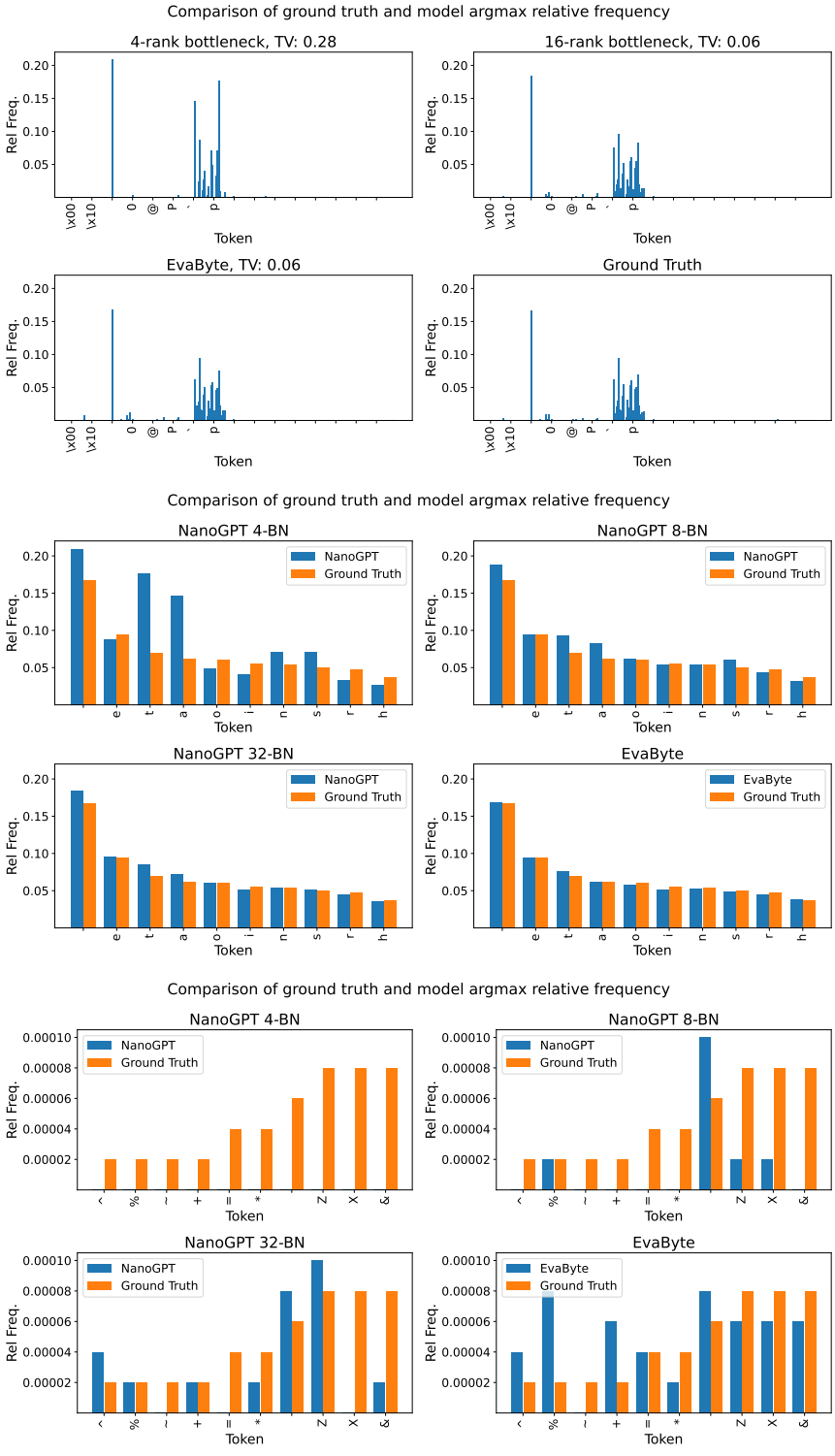


Figure 4.3: Relative Frequency of Model Argmaxes, overall and for 10 most and least frequent tokens (with non-zero occurrences).

Chapter 5

LLM Sample Analysis

5.1 How does the softmax bottleneck impact LLM sample quality?

For many, the goal of LLMs is not to obtain the lowest possible cross-entropy loss on a dataset, it is to obtain models that can generate the highest quality text possible. Hence, in this chapter, we ask the research question **how does the softmax bottleneck impact LLM sample quality?** To shed light on this, we first qualitatively examine model samples, when sampling directly from a model (pure sampling), to identify specific failure modes.

When viewing these samples in Table 5.1, we observe improvements in quality as the bottleneck is alleviated. The 4-dimensional bottleneck produces mostly noise, where 'words' are very long, and sometimes contain numbers alongside arbitrary uses of punctuation. The 8-dimensional bottleneck is noticeably better, as it produces a number of valid English words, as well as some that are within a single edit. Furthermore, the lengths of words are more typical, although the sentences formed are not grammatically correct and do not have any discernible meaning. Additionally, the model has learned that capital letters follow end-of-sentence demarcations. Finally, the model with the lowest validation loss only produces valid English words which are semantically related to the prompt and often generates sentences that arguably adhere to grammatical rules.

Prompt: "**In fact, cooking churns out airborne contaminants like nothing else in the home can. This**

Decoding	4-BN	8-BN	32-BN
Pure	**concentrat**ies lyutemoscafs do-imdusanmewo musdisa trade vowDollesretra wa anml. ¨ A Twhalofnotriamrifanx miniror (,, fratoftry Sadpas otoaaSssret-fixrrratisforbama AsradorsotrodeMusfia ?slot tafmframes ufwhrimteth di5s reTtabamanc. nywsawHoctrono.ml.a tofSunny/perrisapaesst bitaphosoi tidro Aciliiuna taLi InlinaphonrpfAfC faboaa mhepblaslaisriowo. Ritareqtos we inmoatoinmitianbriufcs myni NUL eCooktuqe mechwricotumisdi oe padUfigro foaanancasreHonMo29n maw opttiSpoofPre''d worab;arfmiptaauraiscrllapayopo "	**concentrat**ion of tamoxifen deposits, meetings, spet addresses like the waaay length of what ensures a fading in the ,, from ford to passover Salt in from these soaubis; don't consmit it as a coan. Pamda which meets these netlife in unwanted front. Dollars under this specirity of tossing up a calorie health: A sddhh. Because a fat meal sister, in your wore tossed into their in an restaurant talk, they meet the tomato moe plant grooming and snacking in a cw harter. Sugrnath when bicycling and is the pale po"	**concentrat**ion raises the question: What happen to the surface level)? Is it more common for environmentalists to cook, for example, in too much heat? In recent years, the truth has been raised about the universe's impacts on the environment. Its long-term effects on health and hospitality A number of highly skilled people have developed this case into a true success. Planning a Home Improvement Case Acute actions are associated with an increased influx of physicians - whether they have an attendance o"
Greedy	**concentrat**e at the task ostetaas and the task at the task at the task at the task at the task at the task at at the task at the task at the task at the task at the task at the task at the task at at the task at the task at the task at the task at the task at the task at the task at at the task at the task at the task at the task at the task at the task at the task at at the task at the task at the task at the task at the task at the task at the task at at the task at the task at the task at the task at the task at the task at the task at the t"	**concentrat**es on the contaminants and the contaminants that are the contaminants that are the contaminants that are the contaminants that are the contaminants that are the contaminants that are the contaminants that are the contaminants that are the contaminants that are the contaminants that are the contaminants that are the contaminants that are the contaminants that are the contaminants that are the contaminants that are the contaminants that are the contaminants that are the contaminants that are the c"	**concentrat**es on the fact that the cooking churns are not as contaminants as they are in the home. The cooking churns are also contaminants that are not contaminants that are not contaminants that are not contaminants that are not contaminants that are not contaminants that are not contaminants that are not contaminants that are not contaminants that are not contaminants that are not contaminants that are not contaminants that are not contaminants that are not contaminants that are not contaminants that ar"

Table 5.1: Sample continuations from 4-, 8-, and 32-bottlenecked models using pure and greedy decoding. Prefix shown in bold.

Prompt: "**In fact, cooking churns out airborne contaminants like nothing else in the home can. This**

Decoding	4-BN	8-BN	32-BN
Top-k ($k = 5$)	**concentrat**ies lyonrroachal contaduts ceole creitals lusan neighasresraes If illtaciol to the noon amateititudinrers on the pitafoatrias on casacomcoafrupate aftita acid cotlonrolrol and appitatyrafat throllrremenru pill testant anciiyosasa coconyslalatates inrepersioserasi testant ance camtasosasraaa tandiprocaaino cottifita acua literiasroidrical, tosarerrosare intofeasiiri tanginreotes cotlonolrols testing otomisa ancasadoafrita latere isreporto innomaatiarafitrosare testant litofeitisrocaisarasa liton "	**concentrat**ion of talented people is maent to the straddling of the that a bicycle is copied the poaching itself. The comparaft things are somehow confusing, and the corments to the sort of color that these create on its own are and how a sophisticated tool intertwines tess testing. The carla calls a seand on her throat to the starship there is a sort of percussion to their altar which is courtesy of a perdisco that is supposed to fit a balerina. A friend of probably an accement to this altar, then asks to"	**concentrat**on of alloy alloy is crucial for healthy food, while successfully managing the collection of airborne contaminants. This is especially important for homeowners who want more collection from the house or company, and this is the key to successfully completing anything that is currently in use. In this photo, the search for the cooling alloy is a list of the topics that must be followed. First, it is important to follow the topics that are relevant to the house cooling industry. First, you should"
Top-p ($p = 0.9$)	**concentrat**ies lastrly umfward-imdessed woamrer salt osuvowdo led last week. 4. Yoimre with enothing was xradiible enough. 4 Ofriday. sick asSalt neftrrratist'' usasr dots toothfuloidraplitaIs tortitoru were tortidiContestant anciient we copioy. Doamrofilonre efnisades staatooprosoi trurnatii singawemwi pwAraditi. Blettrefaboaa merpas we isrical. Toward tossrectimoneoeinmia ciallufce mani aCoomrers mecuwesaoto isn moe paapaigro foaa ancienla nanridiso, opitites frrepahpwartabiarfsiptaaura. 5-1Wopalopo "	**concentrat**ion of tamoxifen deposits when the disease occurs is usually wa and peaceful when enunciated and monitored throughout the insymmetric tamoxifen atched to a scale, or one minute as the disease puffs toward the wandering in hospitals room. Dolores under this specirity of tamoxifen lasting samples will still continue at the heart of the can, toiling itself the emotional stress that it occurs emotionally. As diminutive as the paranoisregulant disease, this sort of hog tamoxifen produces collapses of"	**concentrat**ion raises the question: What happens when churns out die? The answer was the Stanford University Press survey from 1983 onwards and comparisons of food availability of food and processed food were done using a planned process which was later designed to identify the appropriate nutritional allowances for cooking food and processed food. This report was published in an annual meeting of the National Agricultural Cooperation Council (NACC) on Thursday, 24 October 2017. The survey report was last "

Table 5.2: Sample continuations from 4-, 8-, and 32-bottlenecked models using Top-k and nucleus decoding. Prefix shown in bold.

Bottleneck	Perplexity	Repetition	MAUVE	Byte TV	Valid Words	Avg Word Len
4	1965.994 ± 346.062	0.000	0.009	0.103	0.168 ± 0.064	10.967 ± 1.928
8	1156.779 ± 515.703	0.007	0.052	0.031	0.539 ± 0.101	5.650 ± 0.729
12	603.491 ± 326.173	0.017	0.914	0.013	0.678 ± 0.092	5.068 ± 0.502
16	306.099 ± 200.220	0.027	**0.977**	0.015	**0.732 ± 0.067**	4.973 ± 0.528
32	198.373 ± 119.963	0.039	0.953	0.014	0.722 ± 0.075	4.947 ± 0.535
128	**192.520 ± 106.012**	**0.043**	0.902	0.013	0.742 ± 0.081	**4.855 ± 0.488**
512	243.898 ± 135.652	0.036	0.916	**0.011**	0.746 ± 0.065	4.968 ± 0.577
Human	40.673 ± 23.855	0.049	1.000	0.000	0.737 ± 0.068	4.836 ± 0.550

Table 5.3: Pure sampling quality metrics across bottleneck sizes, with human sample results as reference. Sample level heuristics are shown as mean ± standard deviation, corpus levels heuristics are shown as mean. Bold indicates mean closest to human per metric.

5.2 Sample Evaluation Metrics

Now that we have elucidated how oversmoothing causes text degeneration, we perform a more extensive analysis of model sampling. To do so, we use 2 metrics that are justified by observations from our samples - mean word length, and mean proportion of valid English words. We also use total variation of byte relative frequency compared with human samples, to see if oversmoothed models do indeed over-generate infrequent tokens. For completeness, we also utilise the evaluation framework specified by Chen et al., evaluating perplexity under a strong, external LLM, the proportion of samples which contain repetitions and the MAUVE score[1] [44] compared to human samples in FineWeb are all evaluated. For both perplexity and MAUVE we use GPT-2 Large [11]. We note that although these metrics are informative, they are less so than our metrics that have been chosen to identify specific failures oversmoothing induces in our model. It is known that perplexity is an imperfect metric [45] which may be misleading - a lower score is not necessarily a better score, and even good human samples will not obtain the lowest scores under these metrics.

5.3 Sample Analysis

The metric evaluations are shown in Table 5.3. From these, we see that sample quality degrades significantly with severe bottlenecks, with the 4- and 8-bottlenecked NanoGPT

[1]MAUVE score is computed using a KL divergence in a quantised, low-dimensional embedding space after human and model samples are embedded, again with a strong, external LLM

models producing samples with long, gibberish words and producing text that is distributed differently from natural English text. One interesting observation is that the model with a 12-dimensional bottleneck produces text with an average word length comparable to human text, and with the strong majority of words being valid English. However these samples achieve significantly worse perplexity and MAUVE scores, suggesting that at this rank, the LLM has learned how to generate text that superficially appears similar to English but carries limited deeper information. For example, this model may learn to generate individual words but fails to combine related words in text. Beyond this, we again see a diminishing return when the low-rank constraint is alleviated. Performance between the 16 to 512 bottlenecked models is metric dependent, as is the ranking of such models, again providing evidence that when models are moderately bottlenecked, behaviour is strongly impacted by the training process and prompt provided.

We also look at the relative frequency of bytes produced by our models in Figure 5.1 to compare our human and model samples. In these histograms we see that the moderate- and full-rank models produce text that is distributed more similarly to human text, compared to the low-rank models. When we focus on extrema, the moderate and weakly constrained models generate the most common bytes at rates similar to humans; however, for the least commonly occurring tokens, the models differ more from humans. This may be due to the granularity offered by our sample sizes, as for the least frequent bytes even one-byte deviation in the number of occurrences causes a proportionally significant deviation from the ground truth text. It is also interesting to note that here, again, weaker bottlenecks improve on the moderate bottlenecks, as they oversample the least frequent bytes less. When combined with the above result, this may provide evidence that oversmoothing in moderately bottlenecked models is not present at the level of single-token prediction, but rather occurs over longer, partially self-generated contexts, where the effects of oversmoothing accumulate.

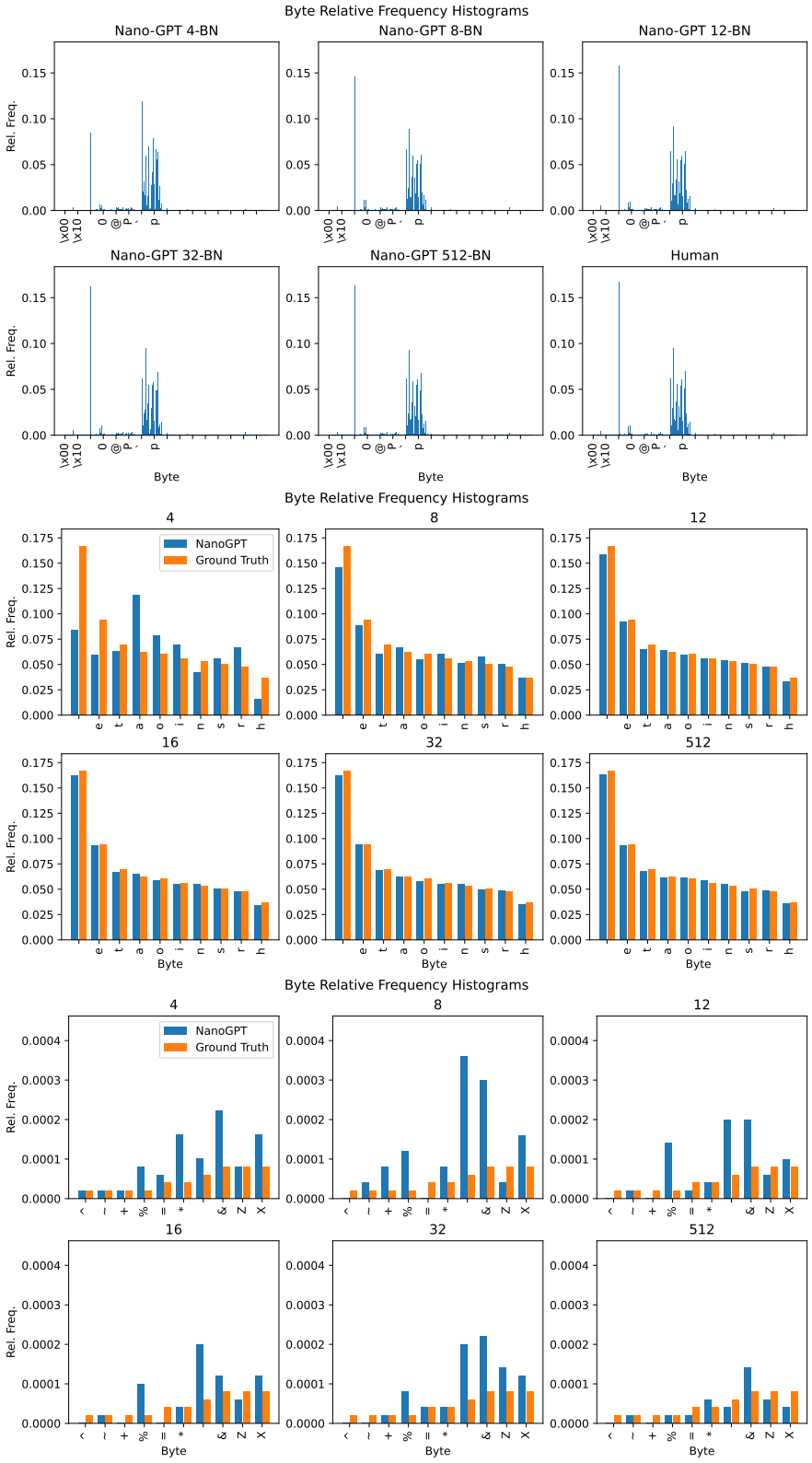


Figure 5.1: Relative frequency of bytes in bottlenecked models compared to human samples, overall and for 10 most and least frequent tokens in human text (with non-zero occurrences)

Chapter 6

Interaction of Oversmoothing and Decoding Methods

6.1 Decoding Methods

Finlayson et al. hypothesised that decoding methods work by directly mitigating the softmax bottleneck's impact on oversmoothing. From the previous sections, we have evidence that at least extremely low-rank bottlenecks cause oversmoothing, thus this hypothesis' assumptions are met. Therefore, we can empirically investigate this hypothesis and answer the research question: **how do decoding methods impact oversmoothing?** To do so, we repeat the investigation performed in Section 4.2, applying decoding methods to our bottlenecked models before comparing with our ground truth distributions. We consider the most common decoding methods: top k, top p, temperature and greedy decoding[1]. We show results in Table 6.1.

In these results, we observe that across all bottlenecks the best method for each metric is constant: greedy under TTO and NSMD, temperature under TV and pure sampling under KL. Under TTO and NSMD, all methods reduce oversmoothing, however the size of alleviation is limited; no method allows a bottlenecked model to outperform pure sampling from a model with a better validation loss. Under TV, temperature sampling is optimal and only nucleus sampling from the most bottlenecked models can also provide an improvement on pure sampling. For KL-divergence, across all bottlenecks, pure is best, followed by temperature, nucleus, top-k and finally greedy, which performs significantly worse than other methods. Overall these results suggest that decoding methods are partially useful in reducing oversmoothing. Furthermore the divergence

[1]For parameters, we use $k = 5$, $p = 0.9$ and a temperature of 0.8

Bottleneck	Metric	Pure	Greedy	Top K	Top P	Temp.
4	TTO	0.409	**0.221**	0.366	0.374	0.355
	NSMD	0.400	**0.187**	0.341	0.358	0.342
	TV	0.457	0.429	0.465	0.434	**0.419**
	KL	**1.082**	8.223	4.452	2.154	1.105
8	TTO	0.151	**0.055**	0.117	0.125	0.115
	NSMD	0.142	**0.020**	0.093	0.105	0.102
	TV	0.216	0.292	0.234	0.209	**0.202**
	KL	**0.404**	5.376	2.187	1.234	0.436
12	TTO	0.125	**0.048**	0.096	0.103	0.094
	NSMD	0.117	**0.013**	0.074	0.082	0.081
	TV	0.195	0.285	0.215	0.192	**0.185**
	KL	**0.326**	5.250	1.813	1.132	0.358
16	TTO	0.100	**0.038**	0.077	0.081	0.073
	NSMD	0.093	**0.003**	0.054	0.059	0.060
	TV	0.170	0.272	0.193	0.171	**0.164**
	KL	**0.258**	4.982	1.578	1.030	0.289
32	TTO	0.089	**0.034**	0.071	0.072	0.064
	NSMD	0.082	**-0.002**	0.048	0.049	0.051
	TV	0.158	0.267	0.184	0.161	**0.154**
	KL	**0.222**	4.873	1.453	0.959	0.251
128	TTO	0.088	**0.035**	0.072	0.072	0.065
	NSMD	0.081	**0.000**	0.049	0.048	0.051
	TV	0.159	0.269	0.186	0.163	**0.156**
	KL	**0.226**	4.913	1.463	0.986	0.257
512	TTO	0.106	**0.044**	0.088	0.088	0.080
	NSMD	0.099	**0.009**	0.065	0.066	0.067
	TV	0.179	0.281	0.204	0.180	**0.173**
	KL	**0.265**	5.165	1.577	1.041	0.295

Table 6.1: Oversmoothing metrics across bottleneck sizes for various decoding methods, averaged over EvaByte and NanoGPT-300M. Bold indicates lowest value per metric.

results suggest there is more to sample quality than lack of oversmoothing. For example, greedy decoding reduces oversmoothing, however, this method is known to cause text degradation. These metrics elucidate why, oversmoothing must be reduced while preserving the distribution's shape. Thus, a lack of oversmoothing is not a sufficient condition for good LLM performance.

6.2 Effect on Sample Quality

Next, we investigate the impact of decoding methods in terms of the softmax bottleneck and sample quality. We first qualitatively evaluate model samples in Tables 5.1 and 5.2. We note that the degradation caused by greedy decoding is evident, as the samples become completely repetitive. We observe, however, that the model with the strictest bottleneck exhibits the shortest repeated phrase of 'at the task'. We also note that the moderate bottleneck generates the longest initial string of non-repetitive text. This aligns with existing literature, where more expressive models take longer for samples to become degenerate [46]. We note that almost all words produced by greedy decoding are valid compared to the stochastic sampling methods, suggesting that bottlenecked models can learn valid words, they just lack significant confidence when predicting them. We also note that top-k and top-p sampling allow the 8-dimensional bottleneck to produce text that superficially resembles valid English, with no anomalous uses of punctuation, suggesting that these truncation methods successfully recover probability mass assigned to incredibly unlikely tokens. This lack of erroneous punctuation is also evident with the 4- and 32-dimensional bottlenecks, although the quality of the former is still not good, and the latter is only marginally improved.

We then examine sample quality overall in Tables 6.2 and 6.3. We observe that across all bottlenecks, decoding methods result in a higher proportion of valid English words. We also note that all methods cause shorter words, on average, across all bottlenecks. However, across all bottlenecks, greedy causes extremely repetitive text. For the other methods, repetition increases as the low-rank bottleneck is alleviated, but varies between moderate and full-rank models, and never exceeds greedy. Top-k produces more repetitive text than nucleus sampling, and both are more repetitive than pure sampling. Under total variation, however, greedy decoding produces bytes with a distribution most dissimilar to human text, followed by top-k and nucleus, and then pure sampling. However, under this metric the distinction between models is weak beyond the extreme bottlenecks.

Bottleneck	Metric	Pure	Greedy	Temp.
4	Perplexity	1965.994 ± 346.062	**2.191 ± 1.215**	2156.877 ± 457.141
	Repetition	0.000	0.944	0.001
	MAUVE	0.009	**0.031**	0.005
	Byte TV	0.103	0.170	**0.099**
	Valid Words	0.168 ± 0.064	**0.823 ± 0.262**	0.262 ± 0.076
	Avg Word Len	10.967 ± 1.928	9.954 ± 49.659	9.036 ± 1.554
8	Perplexity	1156.779 ± 515.703	**2.614 ± 1.208**	404.356 ± 206.997
	Repetition	0.007	0.929	0.061
	MAUVE	0.052	0.178	0.732
	Byte TV	0.031	0.076	0.020
	Valid Words	0.539 ± 0.101	0.914 ± 0.107	**0.739 ± 0.062**
	Avg Word Len	5.650 ± 0.729	**4.765 ± 4.561**	4.727 ± 0.416
12	Perplexity	603.491 ± 326.173	**2.606 ± 1.032**	182.636 ± 104.246
	Repetition	0.017	0.931	0.081
	MAUVE	0.914	0.330	0.905
	Byte TV	**0.013**	0.052	0.016
	Valid Words	0.678 ± 0.092	0.893 ± 0.117	**0.792 ± 0.065**
	Avg Word Len	5.068 ± 0.502	4.202 ± 0.853	**4.787 ± 0.601**
16	Perplexity	306.099 ± 200.220	2.925 ± 1.031	98.499 ± 57.560
	Repetition	0.027	0.918	0.078
	MAUVE	**0.977**	0.476	0.968
	Byte TV	**0.015**	0.045	0.019
	Valid Words	**0.732 ± 0.067**	0.864 ± 0.144	0.807 ± 0.057
	Avg Word Len	**4.973 ± 0.528**	9.053 ± 49.347	4.623 ± 0.483
32	Perplexity	198.373 ± 119.963	2.745 ± 1.172	67.930 ± 34.466
	Repetition	**0.039**	0.920	0.137
	MAUVE	0.953	0.282	0.984
	Byte TV	**0.014**	0.045	0.017
	Valid Words	**0.722 ± 0.075**	0.860 ± 0.133	0.790 ± 0.075
	Avg Word Len	**4.947 ± 0.535**	9.066 ± 49.346	4.585 ± 0.479
128	Perplexity	192.520 ± 106.012	2.760 ± 1.132	66.197 ± 28.282
	Repetition	**0.043**	0.927	0.119
	MAUVE	0.902	0.523	**0.961**
	Byte TV	**0.013**	0.051	0.016
	Valid Words	**0.742 ± 0.081**	0.856 ± 0.168	0.805 ± 0.059
	Avg Word Len	**4.855 ± 0.488**	9.030 ± 49.351	4.557 ± 0.522
512	Perplexity	243.898 ± 135.652	2.975 ± 2.021	82.957 ± 36.561
	Repetition	**0.036**	0.925	0.124
	MAUVE	0.916	0.187	0.917
	Byte TV	**0.011**	0.044	0.017
	Valid Words	**0.746 ± 0.065**	0.872 ± 0.148	0.802 ± 0.067
	Avg Word Len	**4.968 ± 0.577**	9.138 ± 49.343	4.657 ± 0.514

Table 6.2: Quality metrics (Pure, Greedy, Temperature) across bottleneck sizes. Mean ± SD shown where available. Bold indicates value closest to human per metric and bottleneck.

Bottleneck	Metric	Top-K	Top-P
4	Perplexity	1340.325 ± 329.720	1766.004 ± 328.681
	Repetition	**0.014**	0.001
	MAUVE	0.004	0.009
	Byte TV	0.127	0.109
	Valid Words	0.371 ± 0.096	0.229 ± 0.078
	Avg Word Len	**7.458 ± 1.461**	10.619 ± 2.141
8	Perplexity	209.123 ± 99.117	432.948 ± 222.826
	Repetition	0.090	**0.037**
	MAUVE	0.896	**0.969**
	Byte TV	0.033	**0.020**
	Valid Words	0.782 ± 0.051	0.731 ± 0.061
	Avg Word Len	4.532 ± 0.393	4.928 ± 0.427
12	Perplexity	93.882 ± 51.537	166.786 ± 85.250
	Repetition	0.140	**0.068**
	MAUVE	**0.965**	0.945
	Byte TV	0.028	0.018
	Valid Words	0.805 ± 0.060	0.799 ± 0.056
	Avg Word Len	4.595 ± 0.539	4.714 ± 0.575
16	Perplexity	**65.106 ± 25.712**	94.507 ± 45.009
	Repetition	0.117	**0.065**
	MAUVE	0.967	0.950
	Byte TV	0.030	0.017
	Valid Words	0.827 ± 0.052	0.806 ± 0.051
	Avg Word Len	4.464 ± 0.508	4.673 ± 0.539
32	Perplexity	**48.966 ± 24.123**	75.947 ± 39.858
	Repetition	0.170	0.101
	MAUVE	0.988	**0.995**
	Byte TV	0.029	0.016
	Valid Words	0.804 ± 0.079	0.791 ± 0.071
	Avg Word Len	4.583 ± 1.245	4.678 ± 0.595
128	Perplexity	**47.668 ± 20.714**	69.047 ± 30.381
	Repetition	0.165	0.091
	MAUVE	0.930	0.958
	Byte TV	0.027	0.019
	Valid Words	0.813 ± 0.074	0.801 ± 0.062
	Avg Word Len	4.414 ± 0.511	4.565 ± 0.522
512	Perplexity	**64.352 ± 29.899**	92.821 ± 48.523
	Repetition	0.171	0.096
	MAUVE	0.935	**0.947**
	Byte TV	0.028	0.020
	Valid Words	0.806 ± 0.076	0.807 ± 0.057
	Avg Word Len	4.697 ± 1.889	4.698 ± 0.605

Table 6.3: Quality metrics (Top-K, Top-P) across bottleneck sizes. Mean ± SD shown where available. Bold indicates value closest to human per metric and bottleneck.

Chapter 7

Exploration of LLM Representation Rank

7.1 SVD Low-Rank Bottleneck

In previous chapters we explored the impact of the softmax bottleneck when it is learned. As an adjunct to this, we may ask: **how does the softmax bottleneck impact LLMs when it is statically imposed?** Investigating this question also acts as an ablation study for where learning dynamics cannot help mitigate the bottleneck's effects. In order to investigate this, we perform SVD on the weight matrix of the unembedding layer for our full-rank 50M model and obtain approximations of varying ranks. We then perform an analysis similar to Section 4.2, where we analyse the level of oversmoothing exhibited by the output distributions. Note that we do not evaluate oversmoothing using TBAT as our previous results indicate it is not informative across bottlenecks of different ranks.

Our results are shown in Tables 7.1 and 7.2 and an illustrative example of the model output probability distributions and the corresponding levels of oversmoothing can be seen in Figure 7.1. In these results, we observe a monotonic increase in oversmoothing as the rank of approximation decreases. In terms of qualitative descriptions of the output distributions, as the dimensionality of the bottleneck increases, the models produce more uniform-like outputs, struggling to distinguish between bytes. When compared to the results in Section 4.2, these results suggest that models with a near-full rank softmax layer may become dependent on this expressivity to obtain good performance. However, this information is compressible and models that have a rank-constraint imposed (which they must learn to mitigate) learn a more efficient representation, which achieves comparable performance. These results also suggest that using SVD to

Approx. Rank	EvaByte		NanoGPT 300M	
	TTO	NSMD	TTO	NSMD
4	0.823 ± 0.159	0.816 ± 0.165	0.788 ± 0.184	0.780 ± 0.191
8	0.813 ± 0.164	0.806 ± 0.170	0.777 ± 0.189	0.770 ± 0.196
12	0.806 ± 0.170	0.800 ± 0.176	0.770 ± 0.195	0.762 ± 0.201
16	0.795 ± 0.177	0.788 ± 0.183	0.757 ± 0.202	0.750 ± 0.208
20	0.777 ± 0.188	0.772 ± 0.192	0.738 ± 0.211	0.733 ± 0.216
24	0.765 ± 0.196	0.760 ± 0.200	0.724 ± 0.218	0.719 ± 0.223
28	0.741 ± 0.208	0.737 ± 0.211	0.699 ± 0.229	0.694 ± 0.233
32	0.723 ± 0.216	0.720 ± 0.219	0.679 ± 0.235	0.676 ± 0.238
64	0.551 ± 0.222	0.548 ± 0.224	0.499 ± 0.224	0.496 ± 0.226
128	0.213 ± 0.222	0.209 ± 0.223	0.152 ± 0.168	0.148 ± 0.169
256	**0.136 ± 0.208**	**0.129 ± 0.209**	**0.076 ± 0.138**	**0.069 ± 0.139**

Table 7.1: Oversmoothing results for SVD-imposed bottlenecks under TTO and NSMD. Values are mean ± standard deviation. Bold indicates lowest value per metric.

approximate the unembedding layer before BAT sampling results in significant loss of information, and so this approximation may be detrimental.

7.2 Analysis of Representation Effect Rank

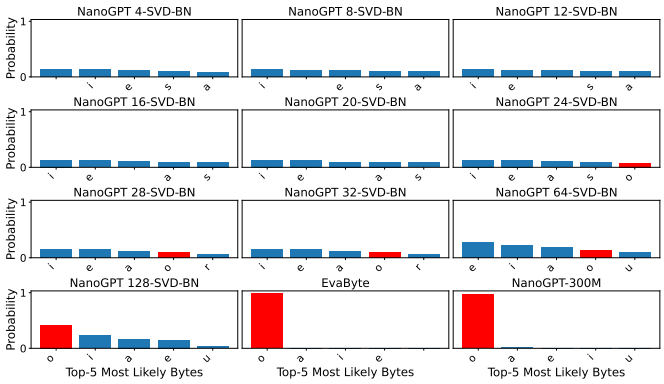
In the previous sections, we saw that model performance persists through moderate bottlenecking. Hence, we can view bottlenecking as a compression mechanism where models learn more efficient representations. This compression is such that quality and performance do not degrade immediately, but do degrade when compression is extreme. To substantiate this view, in Table 7.3 we calculate the effective ranks [30] of different internal representations computed by the models. Here we see that for very strong bottlenecks there is degradation of hidden and output distributions. However, for moderate bottlenecks, hidden representations and probabilities are of similar rank to the full-rank models, but the post-bottleneck and pre-softmax representations are not. This suggests that the bottleneck layer and softmax operation remove and modify information which is redundant to the final output, and so the models do learn more efficient representations.

Approx. Rank	EvaByte		NanoGPT 300M	
	TV	KL	TV	KL
4	0.841 ± 0.122	2.541 ± 1.099	0.813 ± 0.140	2.344 ± 1.062
8	0.833 ± 0.126	2.473 ± 1.091	0.805 ± 0.142	2.276 ± 1.049
12	0.828 ± 0.131	2.421 ± 1.071	0.798 ± 0.148	2.227 ± 1.032
16	0.818 ± 0.137	2.339 ± 1.060	0.788 ± 0.153	2.149 ± 1.023
20	0.802 ± 0.148	2.227 ± 1.043	0.771 ± 0.163	2.042 ± 1.012
24	0.791 ± 0.156	2.160 ± 1.044	0.759 ± 0.170	1.977 ± 1.012
28	0.770 ± 0.169	2.014 ± 1.001	0.736 ± 0.182	1.835 ± 0.964
32	0.753 ± 0.177	1.901 ± 0.970	0.718 ± 0.189	1.725 ± 0.927
64	0.593 ± 0.192	1.135 ± 0.747	0.553 ± 0.187	0.971 ± 0.603
128	0.277 ± 0.238	0.433 ± 0.661	0.225 ± 0.193	0.275 ± 0.381
256	**0.208 ± 0.239**	**0.348 ± 0.679**	**0.150 ± 0.179**	**0.181 ± 0.365**

Table 7.2: Oversmoothing results for SVD-imposed bottlenecks under TV and KL divergences. Values are mean ± standard deviation. Bold indicates lowest value per metric.

Approx. Rank	Logits	Probabilities	Post-Bottleneck Layer	Pre-Bottleneck Layer
4	3.30	18.11	2.93	334.28
8	5.29	31.21	6.56	384.24
12	6.35	33.79	10.16	396.96
16	8.46	34.36	13.61	408.25
32	12.54	34.53	27.06	414.58
128	20.92	36.32	77.13	417.07
512	27.86	35.34	205.80	414.28

Table 7.3: Effective rank values across learned model bottlenecks for different representations. Values are computed using a batch of 1024 prompts.



Approximation Rank	TTO	NSMD	TV	KL
4	0.913	0.913	0.913	2.619
8	0.908	0.908	0.908	2.544
12	0.909	0.909	0.909	2.553
16	0.906	0.906	0.906	2.504
20	0.900	0.900	0.900	2.429
24	0.897	0.897	0.897	2.397
28	0.884	0.884	0.884	2.238
32	0.876	0.876	0.876	2.152
64	0.839	0.839	0.839	1.841
128	0.562	0.562	0.562	0.766
256	**0.499**	**0.499**	**0.499**	**0.634**

Figure 7.1: Model probabilities and levels of oversmoothing with SVD-imposed bottlenecks for the prompt: "It's an insult to the democratic process. In 1996, Michigan's voters overwhelmingly approved a pl that gives sole authority to Michigan's NRC for setting wildlife management policies. Our democratic system uses a system where v" . Correct token probability is in red. Table values are means when compared with EvaByte and NanoGPT 300M.

Chapter 8

Conclusions

In Section 4.1, we define oversmoothing in LLMs and multiple metrics to capture this behaviour, outlining the pros and cons of each approach. Then in Section 4.2, we provide evidence that when tasked with next-token prediction, models with extreme softmax bottlenecks imposed do oversmooth, and loosening this constraint helps reduce the level of oversmoothing. For moderately and minimally constrained models, the bottleneck dimensionality does not impact oversmoothing, although such models still oversmooth. Additionally, oversmoothing is prompt-dependent, and two models may exhibit behaviour which is similar on average, but varies for any specific prompt.

In Section 4.3, we show that the level of oversmoothing exhibited also depends on what token the model should have predicted, with less oversmoothing exhibited when tasked with predicting the most frequent tokens, compared to the rarest tokens. We also showed that when viewed through this lens, as the softmax bottleneck is alleviated, models learn to confidently predict a larger set of tokens, doing so in descending order of token frequency.

In terms of sample quality, in Sections 5.1 and 5.3, we observe that as the expressivity constraint is weakened, the model-generated text transitions from long, noisy strings to text with words that are or are close to valid English words but with little grammatical or semantic consistency between them, and then finally to entire sentences that are almost if not grammatically coherent and have some semantic self-similarity. However, since the models used are small and not fine-tuned for prompting, samples contain limited genuine semantic content. We also observe that for this task, on average, less-constrained models see a slight improvement in the proportion of valid words produced, and in how much the resulting byte distribution differs from human-generated text. Hence, we have evidence that, compared to next-token prediction, sampling is a case

where a fully expressive model outperforms a moderately-bottlenecked model.

Furthermore, in Section 6.2, we investigate the impact of decoding methods on oversmoothing and sample quality, showing that when models oversmooth, decoding methods can reduce the level of oversmoothing, albeit only marginally, and this improvement depends on the chosen metric. This discrepancy between metrics also provides evidence that a lack of oversmoothing is a necessary but not a sufficient condition for good sample quality. Additionally, we observed that decoding methods mitigate the downstream impact of oversmoothing, for example by reducing the presence of arbitrary punctuation; however this improvement is dependent on the chosen method.

Finally, in Section 7.1, we impose the softmax bottleneck statically, after training, to contrast with our previous results where constrained model parameters are learned. In this case, we observe that the level of oversmoothing increases monotonically with the strength of the bottleneck. When contrasted with the previous results, this provides evidence that fully expressive models learn more expressive representations, which are inefficient, and it is possible to learn representations with a lower dimensionality that obtain similar performance. These results are further reinforced when we evaluate the effective rank of the models' representations in Section 7.2.

We see multiple avenues for future work. Firstly, one could combine these results with a hyperparameter sweep to ensure the best-performing models are evaluated for each bottleneck. Furthermore, given a larger computational budget, this work could be repeated while saving checkpoints along the training process to examine the interplay between oversmoothing and model loss, and to observe if models with similar validation losses, but that have different bottleneck constraints and have seen different numbers of tokens, still oversmooth similarly. Additionally, given our computational budget we trained a family of 50M parameter models, which had limited expressivity compared to larger ground truth models. Consequently, an improvement on this work, with a larger computational budget, would be to train a family of larger and more expressive models, to further isolate the impact of the softmax bottleneck on oversmoothing. As well as this, as mentioned, there are several possible explanations for oversmoothing, including training with KL-divergence, and the unimodality issue. It has been suggested that the latter can be ameliorated with more expressive output layers such as Multi-Faced Softmax [24], and exploring this further is a natural next step. In addition to this, our results suggest that there is a trade-off between efficiency and expressivity for byte-level LLMs. It is a natural next step to investigate whether this is the case for sub-word tokenised LLMs.

Bibliography

[1] Di Jin et al. “What disease does this patient have? a large-scale open domain question answering dataset from medical exams”. In: *Applied Sciences* 11.14 (2021), p. 6421.

[2] Akib Mashrur et al. “Machine learning for financial risk management: a survey”. In: *Ieee Access* 8 (2020), pp. 203203–203223.

[3] Junda He et al. “Representation Learning for Stack Overflow Posts: How Far Are We?” In: 33.3 (Mar. 2024). ISSN: 1049-331X. DOI: 10.1145/3635711.

[4] Libo Qin et al. *Large Language Models Meet NLP: A Survey*. https://arxiv.org/abs/2405.12819. 2024. arXiv: 2405.12819 [cs.CL].

[5] Kevin Fischer et al. *Question: How do Large Language Models perform on the Question Answering tasks? Answer:* Dec. 2024. DOI: 10.48550/arXiv.2412.12893.

[6] Wenhao Zhu et al. *Multilingual Machine Translation with Large Language Models: Empirical Results and Analysis*. Apr. 2023. DOI: 10.48550/arXiv.2304.04675.

[7] Takeshi Kojima et al. “Large language models are zero-shot reasoners”. In: *Proceedings of the 36th International Conference on Neural Information Processing Systems*. NIPS ’22. New Orleans, LA, USA: Curran Associates Inc., 2022. ISBN: 9781713871088.

[8] Ari Holtzman et al. “The Curious Case of Neural Text Degeneration”. In: *International Conference on Learning Representations*. 2020.

[9] Zhilin Yang et al. “Breaking the Softmax Bottleneck: A High-Rank RNN Language Model”. In: *International Conference on Learning Representations*. 2018.

[10] Matthew Finlayson et al. “Closing the Curious Case of Neural Text Degeneration”. In: *The Twelfth International Conference on Learning Representations*. 2024.

[11] Alec Radford et al. “Language Models are Unsupervised Multitask Learners”. In: (2019).

[12] John Hewitt, Christopher Manning, and Percy Liang. “Truncation Sampling as Language Model Desmoothing”. In: *Findings of the Association for Computational Linguistics: EMNLP 2022*. Ed. by Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang. Abu Dhabi, United Arab Emirates: Association for Computational Linguistics, Dec. 2022, pp. 3414–3427. DOI: 10.18653/v1/2022.findings-emnlp.249.

[13] Angela Fan, Mike Lewis, and Yann Dauphin. “Hierarchical Neural Story Generation”. In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Ed. by Iryna Gurevych and Yusuke Miyao. Melbourne, Australia: Association for Computational Linguistics, July 2018, pp. 889–898. DOI: 10.18653/v1/P18-1082.

[14] Ashish Vaswani et al. “Attention is all you need”. In: *Advances in neural information processing systems* 30 (2017).

[15] Wikipedia contributors. *UTF-8 — Wikipedia, The Free Encyclopedia*. 2025. URL: https://en.m.wikipedia.org/wiki/File:Full_GPT_architecture.png#filelinks.

[16] Carl Eckart and Gale Young. “The approximation of one matrix by another of lower rank”. In: *Psychometrika* 1.3 (1936), 211–218. DOI: 10.1007/bf02288367.

[17] Octavian Ganea et al. “Breaking the Softmax Bottleneck via Learnable Monotonic Pointwise Non-linearities”. In: *Proceedings of the 36th International Conference on Machine Learning*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, 2019, pp. 2073–2082.

[18] Andreas Grivas, Nikolay Bogoychev, and Adam Lopez. “Low-Rank Softmax Can Have Unargmaxable Classes in Theory but Rarely in Practice”. In: *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Ed. by Smaranda Muresan, Preslav Nakov, and Aline Villavicencio. Dublin, Ireland: Association for Computational Linguistics, May 2022, pp. 6738–6758. DOI: 10.18653/v1/2022.acl-long.465.

[19] Matthew Finlayson, Xiang Ren, and Swabha Swayamdipta. “Logits of API-Protected LLMs Leak Proprietary Information”. In: *First Conference on Language Modeling*. 2024.

[20] Dwarak Govind Parthiban, Yongyi Mao, and Diana Inkpen. “On the Softmax bottleneck of recurrent language models”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 35.15 (2021), 13640–13647. DOI: 10.1609/aaai.v35i15.17608.

[21] Surafel Melaku Lakew, Mauro Cettolo, and Marcello Federico. “A Comparison of Transformer and Recurrent Neural Networks on Multilingual Neural Machine Translation”. In: *Proceedings of the 27th International Conference on Computational Linguistics*. Ed. by Emily M. Bender, Leon Derczynski, and Pierre Isabelle. Santa Fe, New Mexico, USA: Association for Computational Linguistics, Aug. 2018, pp. 641–652.

[22] Srinadh Bhojanapalli et al. “Low-rank bottleneck in multi-head attention models”. In: *International conference on machine learning*. PMLR. 2020, pp. 864–873.

[23] Zellig S. Harris. “Distributional Structure”. In: *Word* 10.2-3 (1954), pp. 146–162. DOI: 10.1080/00437956.1954.11659520.

[24] Haw-Shiuan Chang and Andrew McCallum. “Softmax Bottleneck Makes Language Models Unable to Represent Multi-mode Word Distributions”. In: *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Ed. by Smaranda Muresan, Preslav Nakov, and Aline Villavicencio. Dublin, Ireland: Association for Computational Linguistics, May 2022, pp. 8048–8073. DOI: 10.18653/v1/2022.acl-long.554.

[25] Chufan Shi et al. “A Thorough Examination of Decoding Methods in the Era of LLMs”. In: *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*. Ed. by Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen. Miami, Florida, USA: Association for Computational Linguistics, Nov. 2024, pp. 8601–8629. DOI: 10.18653/v1/2024.emnlp-main.489.

[26] Markus Freitag and Yaser Al-Onaizan. “Beam Search Strategies for Neural Machine Translation”. In: *Proceedings of the First Workshop on Neural Machine Translation*. Ed. by Thang Luong et al. Vancouver: Association for Computational Linguistics, Aug. 2017, pp. 56–60. DOI: 10.18653/v1/W17-3207.

[27] Clara Meister et al. “Locally Typical Sampling”. In: *Transactions of the Association for Computational Linguistics* 11 (Jan. 2023), pp. 102–121. DOI: 10.1162/tacl_a_00536.

[28] Sourya Basu et al. “Mirostat: A Neural Text Decoding Algorithm that Directly Controls Perplexity”. In: *International Conference on Learning Representations*. 2021. URL: https://openreview.net/forum?id=W1G1JZEIy5_.

[29] V. Klema and A. Laub. “The singular value decomposition: Its computation and some applications”. In: *IEEE Transactions on Automatic Control* 25.2 (1980), pp. 164–176. DOI: 10.1109/TAC.1980.1102314.

[30] Olivier Roy and Martin Vetterli. “The effective rank: A measure of effective dimensionality”. In: *2007 15th European Signal Processing Conference*. 2007, pp. 606–610.

[31] Andrej Karpathy. *NanoGPT*. https://github.com/karpathy/nanoGPT. Accessed: 2025-07-03. 2023.

[32] Sean Osier. *sosier/nanoGPT-shakespeare-char-weights-not-tied*. Hugging Face Model. Accessed 2025-07-03; implementation uses transformers AutoModel with remote code. July 2024.

[33] Alec Radford and Karthik Narasimhan. “Improving Language Understanding by Generative Pre-Training”. In: 2018.

[34] François Yergeau. *UTF-8, a transformation format of ISO 10646*. RFC 3629. Nov. 2003. DOI: 10.17487/RFC3629.

[35] Andrej Karpathy. *Tiny Shakespeare*. 2015. URL: https://huggingface.co/datasets/karpathy/tiny_shakespeare.

[36] Guilherme Penedo et al. “The FineWeb Datasets: Decanting the Web for the Finest Text Data at Scale”. In: *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*. 2024.

[37] Lin Zheng et al. *EvaByte: Efficient Byte-level Language Models at Scale*. 2025. URL: https://hkunlp.github.io/blog/2025/evabyte.

[38] Luca Soldaini et al. “Dolma: an Open Corpus of Three Trillion Tokens for Language Model Pretraining Research”. In: *arXiv preprint* (2024).

[39] Anton Lozhkov et al. *StarCoder 2 and The Stack v2: The Next Generation*. 2024. arXiv: 2402.19173 [cs.SE].

[40] Jeffrey Li et al. *DataComp-LM: In search of the next generation of training sets for language models*. 2024. arXiv: 2406.11794.

[41] Ilya Loshchilov and Frank Hutter. "Decoupled Weight Decay Regularization". In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL: https://openreview.net/forum?id=Bkg6RiCqY7.

[42] Sheldon Axler. *Linear Algebra Done Right*. 3rd. Cham: Springer, 2015. ISBN: 978-3-319-11079-0.

[43] Sijin Chen, Omar Hagrass, and Jason Matthew Klusowski. "Decoding Game: On Minimax Optimality of Heuristic Text Generation Strategies". In: *The Thirteenth International Conference on Learning Representations*. 2025.

[44] Krishna Pillutla et al. "MAUVE: Measuring the Gap Between Neural Text and Human Text using Divergence Frontiers". In: *Advances in Neural Information Processing Systems*. Ed. by A. Beygelzimer et al. 2021.

[45] Yequan Wang et al. *Perplexity from PLM Is Unreliable for Evaluating Text Quality*. https://arxiv.org/abs/2210.05892. 2023. arXiv: 2210.05892 [cs.CL].

[46] Sungjin Choi, Charles Margossian, and David M. Blei. *Posterior Sampling for Multi-Armed Bandits with Amortized Value Functions*. Poster presented at the Thirty-Eighth Conference on Neural Information Processing Systems (NeurIPS), Vancouver, Canada. Available at https://neurips.cc/virtual/2024/105336. 2024.

[47] Wikipedia contributors. *UTF-8 — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=UTF-8&oldid=1303123125. [Online; accessed 1-August-2025]. 2025.

[48] Pauli Virtanen et al. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python". In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

Appendix A

UTF-8

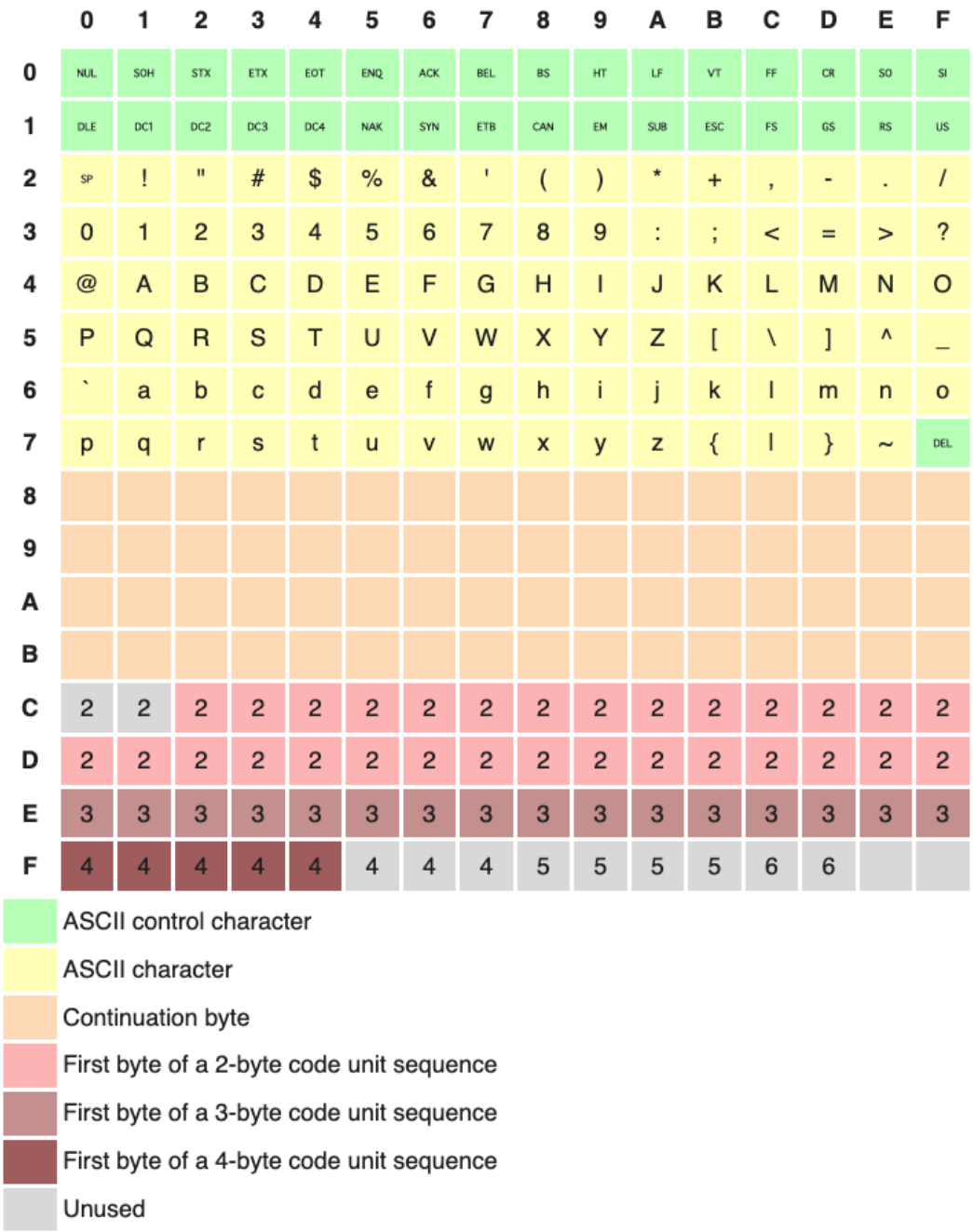


Figure A.1: Chart of UTF-8 Bytes [47]

Appendix B

Bottlenecked Model Expressivity

To compute unargmaxability, we re-use code written by Grivas et al. We also note that, although our model has unargmaxable classes, it is possible to construct models with the given bottleneck dimensions and a vocabulary size of 256 tokens with no unargmaxable classes, hence this behaviour is induced by the training process of our model and is not inherent to bottlenecked models.

Bottleneck	**No. Unargmaxable Classes**	**Unargmaxable Classes**
4	214	NUL SOH STX ETX EOT ENQ ACK BEL BS HT VT FF CR SO SI DLE DC1 DC2 DC3 DC4 NAK SYN ETB CAN EM SUB ESC FS GS RS US ! " # $ & ' ( ) * + - / 2 4 5 6 7 8 < = > ? @ A B C D E F G H I J K L M N O P Q R S U V W X Y Z [\ ] ^ _ ` b j w { — } ~ DEL 81 82 83 84 85 86 87 88 89 8A 8B 8C 8D 8E 8F 90 91 92 94 95 96 97 98 9A 9B 9E 9F A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 AA AB AC AD AE AF B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 BA BB BC BD BE BF C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA CB CC CD CE CF D0 D2 D3 D4 D5 D6 D7 D9 DA DB DC DD DE DF E0 E1 E2 E3 E4 E5 E6 E7 E8 E9 EA EB EC ED EE EF F0 F1 F2 F3 F4 F5 F6 F7 F8 F9 FA FB FC FD FE FF
8	107	NUL SOH STX ETX EOT ENQ ACK BEL BS HT VT FF CR SO SI DLE DC1 DC2 DC3 DC4 NAK SYN ETB CAN EM SUB ESC FS GS RS US < \ ^ { } ~ DEL 87 8A 8B 8C 8E 90 95 96 9B A5 A8 AA AB AF B6 B9 BE C0 C1 C4 C5 C6 C7 C8 C9 CA CB CC CD CF D2 D3 D4 D5 D6 D7 DA DB DC DD DE DF E1 E4 E6 E7 E8 E9 EA EB EC ED EE F0 F1 F2 F3 F4 F5 F6 F7 F8 F9 FA FB FC FD FE FF

Table B.1: Unargmaxable classes across bottleneck sizes (4 and 8)

Bottleneck	**No. Unargmaxable Classes**	**Unargmaxable Classes**
12	53	SOH STX ETX ENQ ACK BS VT SI DLE DC1 DC2 DC4 NAK ETB CAN EM SUB ESC FS GS US { DEL 8A 8E C0 C1 C7 C8 C9 CA CB CC CD D3 D4 D5 DC DD DE E9 F1 F3 F4 F5 F6 F7 F8 FA FB FC FD FE
16	43	NUL SOH STX ETX EOT ENQ ACK BEL BS HT VT FF CR SI DLE DC2 DC3 DC4 NAK SYN EM SUB ESC FS RS US { DEL DC DD EE F2 F3 F5 F6 F7 F8 F9 FA FB FC FD FF
32	19	SOH BS CR DC1 DC2 ESC FS RS DEL F1 F2 F5 F6 F7 F8 FA FB FD FE
128	0	
512	0	

Table B.2: Unargmaxable classes across bottleneck sizes (12, 16, 32, 128 and 512)

Appendix C

Proof of NSMD Equivalence

Consider two models gt and bn over a vocabulary V. Let $V^*_{gt} \subseteq V$ be the nucleus set of model gt of V_{gt} and $(V^*_{gt})'$ its compliment. We note that $V = V^*_{gt} \cup (V^*_{gt})'$. For short-hand we denote $p_{model}(w|\text{prompt})$ as simply $p_{model}(w)$. We recall the definition of NSMD:

$$\text{NSMD}(p_{gt}, p_{bn}) = \sum_{w \in V^*_{gt}} [p_{gt}(w) - p_{bn}(w)] = \sum_{w \in V^*_{gt}} p_{gt}(w) - \sum_{w \in V^*_{gt}} p_{bn}(w)$$

$$= \sum_{w \notin (V^*_{gt})'} p_{gt}(w) - \sum_{w \notin (V^*_{gt})'} p_{bn}(w) = \left[1 - \sum_{w \in (V^*_{gt})'} p_{gt}(w)\right] - \left[1 - \sum_{w \in (V^*_{gt})'} p_{bn}(w)\right]$$

$$= - \sum_{w \in (V^*_{gt})'} p_{gt}(w) + \sum_{w \in (V^*_{gt})'} p_{bn}(w) = - \sum_{w \in (V^*_{gt})'} [p_{gt}(w) - p_{bn}(w)].$$

The final term is the negative of NSMD taken over the compliment of the nucleus set. We observe that if we change the sign of subtraction, so consider $p_{bn}(w) - p_{gt}(w)$, we see that measuring underassignment of probability mass in the nucleus set is indeed equivalent to measuring overassignment of probability mass in the tail end of tokens.

Appendix D

TBAT Formal Definition

Given a model's output distribution $\widehat{\mathbf{p}} = p(\mathbf{w}|\text{prompt})$, its unembedding matrix W and a hyperparameter δ, we formally define the **total BAT-reject set mass** (TBAT) as:

$$\text{TBAT}(p) = \sum_{i \in \text{BAT-Reject}} p(w_i|\text{prompt}) \qquad \text{where}$$

$$\text{BAT-Reject} = \{i : \exists \mathbf{q} \text{ s.t. } q_i = 0,\ \sum_{j=1}^{v} q_j = 1,\ \forall 0 \leq q_j \leq \widehat{p}_j \exp(\delta),\ W^T\mathbf{q} = W^T\widehat{\mathbf{p}}\}.$$

The motivation for the above linear program is that for a true language distribution $\mathbf{p}^*$, if a model's output distribution $\mathbf{p}$ minimise the cross-entropy loss with regard to the true distribution, then $W^T\mathbf{p} = W^T\mathbf{p}^*$. Hence if there is no distribution $\mathbf{q}$ which assigns a given token a zero-probability and satisfies this equality, then the true distribution (which our model minimises cross-entropy loss with) must assign this token non-zero probability. We direct the reader to Finlayson et al. for proof.

We solve the above linear program with SciPy's off-the-shelf linear program solver [48].

Appendix E

Full Oversmoothing Results

E.1 EvaByte

Bottleneck	Metric	Pure	Greedy	Temp.
4	TTO	0.432 ± 0.291	0.248 ± 0.430	0.379 ± 0.311
	NSMD	0.424 ± 0.292	0.217 ± 0.430	0.368 ± 0.313
	TV	0.477 ± 0.286	0.433 ± 0.445	0.438 ± 0.309
	KL	1.170 ± 1.389	8.369 ± 8.775	1.198 ± 1.588
8	TTO	0.178 ± 0.235	0.077 ± 0.264	0.142 ± 0.231
	NSMD	0.170 ± 0.235	0.045 ± 0.267	0.130 ± 0.231
	TV	0.242 ± 0.262	0.294 ± 0.378	0.225 ± 0.267
	KL	0.486 ± 0.860	5.498 ± 7.241	0.523 ± 1.008
12	TTO	0.153 ± 0.219	0.069 ± 0.252	0.121 ± 0.215
	NSMD	0.146 ± 0.219	0.037 ± 0.255	0.110 ± 0.216
	TV	0.223 ± 0.248	0.287 ± 0.371	0.209 ± 0.254
	KL	0.408 ± 0.743	5.356 ± 7.106	0.445 ± 0.875
16	TTO	0.129 ± 0.201	0.058 ± 0.231	0.100 ± 0.196
	NSMD	0.122 ± 0.202	0.026 ± 0.235	0.089 ± 0.198
	TV	0.200 ± 0.235	0.275 ± 0.362	0.190 ± 0.241
	KL	0.341 ± 0.649	5.097 ± 6.884	0.376 ± 0.763

Table E.1: EvaByte ground truth — Pure, Greedy, and Temperature decoding. Values are mean ± standard deviation.

Bottleneck	Metric	Pure	Greedy	Temp.
32	TTO	0.118 ± 0.192	0.053 ± 0.222	0.091 ± 0.188
	NSMD	0.112 ± 0.193	0.020 ± 0.225	0.080 ± 0.190
	TV	0.189 ± 0.226	0.269 ± 0.357	0.181 ± 0.232
	KL	0.305 ± 0.618	4.981 ± 6.774	0.338 ± 0.729
128	TTO	0.118 ± 0.194	0.054 ± 0.224	0.092 ± 0.190
	NSMD	0.111 ± 0.195	0.022 ± 0.228	0.081 ± 0.192
	TV	0.190 ± 0.228	0.271 ± 0.359	0.183 ± 0.235
	KL	0.310 ± 0.650	5.024 ± 6.818	0.346 ± 0.769
512	TTO	0.136 ± 0.208	0.064 ± 0.243	0.108 ± 0.205
	NSMD	0.129 ± 0.209	0.032 ± 0.246	0.097 ± 0.207
	TV	0.208 ± 0.239	0.283 ± 0.368	0.199 ± 0.245
	KL	0.348 ± 0.679	5.270 ± 7.025	0.383 ± 0.800

Table E.2: EvaByte ground truth — Pure, Greedy, and Temperature decoding. Values are mean ± standard deviation.

Bottleneck	Metric	Top-K	Top-P
4	TTO	0.392 ± 0.300	0.399 ± 0.309
	NSMD	0.369 ± 0.302	0.384 ± 0.311
	TV	0.481 ± 0.302	0.453 ± 0.304
	KL	4.609 ± 6.504	2.295 ± 3.850
8	TTO	0.144 ± 0.227	0.152 ± 0.237
	NSMD	0.123 ± 0.227	0.135 ± 0.237
	TV	0.254 ± 0.283	0.232 ± 0.268
	KL	2.314 ± 4.425	1.352 ± 2.832
12	TTO	0.124 ± 0.212	0.131 ± 0.220
	NSMD	0.104 ± 0.214	0.113 ± 0.221
	TV	0.237 ± 0.270	0.216 ± 0.255
	KL	1.930 ± 3.898	1.257 ± 2.669
16	TTO	0.105 ± 0.195	0.109 ± 0.201
	NSMD	0.084 ± 0.197	0.090 ± 0.203
	TV	0.217 ± 0.257	0.197 ± 0.241
	KL	1.693 ± 3.535	1.156 ± 2.472

Table E.3: EvaByte ground truth — Top-K and Top-P decoding. Values are mean ± standard deviation.

Bottleneck	Metric	Top-K	Top-P
32	TTO	0.098 ± 0.187	0.100 ± 0.192
	NSMD	0.078 ± 0.191	0.080 ± 0.194
	TV	0.208 ± 0.249	0.188 ± 0.232
	KL	1.563 ± 3.308	1.081 ± 2.327
128	TTO	0.099 ± 0.189	0.100 ± 0.194
	NSMD	0.079 ± 0.192	0.080 ± 0.196
	TV	0.210 ± 0.251	0.190 ± 0.235
	KL	1.574 ± 3.316	1.109 ± 2.386
512	TTO	0.115 ± 0.204	0.116 ± 0.209
	NSMD	0.095 ± 0.207	0.097 ± 0.211
	TV	0.227 ± 0.260	0.206 ± 0.245
	KL	1.687 ± 3.497	1.163 ± 2.496

Table E.4: EvaByte ground truth — Top-K and Top-P decoding. Values are mean ± standard deviation.

E.2 NanoGPT-300M

Bottleneck	Metric	Pure	Greedy	Temp.
4	TTO	0.386 ± 0.273	0.195 ± 0.394	0.330 ± 0.289
	NSMD	0.376 ± 0.274	0.157 ± 0.395	0.317 ± 0.291
	TV	0.438 ± 0.269	0.425 ± 0.427	0.399 ± 0.288
	KL	0.994 ± 1.164	8.078 ± 8.322	1.011 ± 1.325
8	TTO	0.124 ± 0.175	0.033 ± 0.176	0.089 ± 0.165
	NSMD	0.114 ± 0.175	-0.006 ± 0.180	0.074 ± 0.166
	TV	0.191 ± 0.212	0.289 ± 0.349	0.178 ± 0.215
	KL	0.321 ± 0.583	5.254 ± 6.499	0.350 ± 0.685
12	TTO	0.097 ± 0.153	0.027 ± 0.161	0.066 ± 0.143
	NSMD	0.088 ± 0.153	-0.011 ± 0.166	0.052 ± 0.146
	TV	0.168 ± 0.194	0.284 ± 0.343	0.161 ± 0.199
	KL	0.243 ± 0.468	5.145 ± 6.369	0.271 ± 0.554
16	TTO	0.072 ± 0.126	0.018 ± 0.133	0.046 ± 0.117
	NSMD	0.063 ± 0.127	-0.020 ± 0.139	0.031 ± 0.119
	TV	0.141 ± 0.171	0.270 ± 0.330	0.138 ± 0.177
	KL	0.176 ± 0.356	4.866 ± 6.075	0.202 ± 0.423
32	TTO	0.059 ± 0.113	0.015 ± 0.119	0.037 ± 0.105
	NSMD	0.052 ± 0.115	-0.024 ± 0.126	0.022 ± 0.108
	TV	0.127 ± 0.159	0.265 ± 0.325	0.128 ± 0.166
	KL	0.140 ± 0.301	4.765 ± 5.957	0.165 ± 0.360
128	TTO	0.059 ± 0.115	0.016 ± 0.124	0.037 ± 0.107
	NSMD	0.051 ± 0.117	-0.023 ± 0.131	0.022 ± 0.111
	TV	0.128 ± 0.161	0.267 ± 0.327	0.129 ± 0.168
	KL	0.141 ± 0.307	4.801 ± 6.002	0.168 ± 0.367
512	TTO	0.076 ± 0.138	0.025 ± 0.153	0.052 ± 0.131
	NSMD	0.069 ± 0.139	-0.014 ± 0.159	0.038 ± 0.134
	TV	0.150 ± 0.179	0.279 ± 0.338	0.148 ± 0.186
	KL	0.181 ± 0.365	5.059 ± 6.268	0.206 ± 0.430

Table E.5: NanoGPT-300M ground truth — Pure, Greedy, and Temperature decoding. Values are mean ± standard deviation.

Bottleneck	Metric	Top-K	Top-P
4	TTO	0.340 ± 0.282	0.350 ± 0.289
	NSMD	0.313 ± 0.287	0.333 ± 0.292
	TV	0.449 ± 0.286	0.415 ± 0.284
	KL	4.294 ± 5.923	2.013 ± 3.235
8	TTO	0.090 ± 0.163	0.099 ± 0.174
	NSMD	0.064 ± 0.164	0.076 ± 0.174
	TV	0.213 ± 0.244	0.186 ± 0.216
	KL	2.060 ± 3.752	1.116 ± 2.069
12	TTO	0.069 ± 0.143	0.075 ± 0.150
	NSMD	0.043 ± 0.148	0.051 ± 0.153
	TV	0.193 ± 0.227	0.168 ± 0.199
	KL	1.696 ± 3.286	1.007 ± 1.835
16	TTO	0.049 ± 0.118	0.054 ± 0.123
	NSMD	0.024 ± 0.124	0.028 ± 0.126
	TV	0.169 ± 0.208	0.145 ± 0.176
	KL	1.463 ± 2.921	0.904 ± 1.550
32	TTO	0.044 ± 0.108	0.044 ± 0.110
	NSMD	0.019 ± 0.115	0.017 ± 0.115
	TV	0.160 ± 0.198	0.134 ± 0.165
	KL	1.344 ± 2.703	0.837 ± 1.407
128	TTO	0.044 ± 0.110	0.044 ± 0.112
	NSMD	0.020 ± 0.117	0.017 ± 0.117
	TV	0.161 ± 0.200	0.136 ± 0.168
	KL	1.353 ± 2.713	0.862 ± 1.471
512	TTO	0.060 ± 0.133	0.060 ± 0.136
	NSMD	0.035 ± 0.140	0.034 ± 0.140
	TV	0.181 ± 0.214	0.155 ± 0.186
	KL	1.468 ± 2.899	0.919 ± 1.640

Table E.6: NanoGPT-300M ground truth — Top-K and Top-P decoding. Values are mean ± standard deviation.

E.3 TBAT

Bottleneck	**Pure**	**Greedy**	**Temp.**
4	0.273 ± 0.314	0.010 ± 0.099	0.228 ± 0.294
8	0.069 ± 0.127	0.000 ± 0.000	0.0449 ± 0.0962
12	0.0285 ± 0.0658	0.000 ± 0.000	0.0157 ± 0.0431
16	0.0171 ± 0.0442	0.000 ± 0.000	0.00805 ± 0.0240
32	0.00392 ± 0.0132	0.000 ± 0.000	0.00136 ± 0.00577
128	$2.46 \times 10^{-6} \pm 6.88 \times 10^{-6}$	0.000 ± 0.000	$1.0 \times 10^{-7} \pm 3.3 \times 10^{-7}$
512	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000

Table E.7: TBAT for Pure, Greedy, and Temperature decoding methods across bottleneck sizes. Values are mean ± standard deviation.

Bottleneck	**Top K**	**Top P**
4	0.152 ± 0.220	0.238 ± 0.314
8	0.00474 ± 0.0215	0.0462 ± 0.106
12	$0.000 \pm 5.0 \times 10^{-8}$	0.0119 ± 0.0450
16	$0.000 \pm 5.0 \times 10^{-8}$	0.00518 ± 0.0239
32	$1.0 \times 10^{-8} \pm 4.0 \times 10^{-8}$	0.000205 ± 0.00204
128	$0.000 \pm 2.0 \times 10^{-8}$	$0.000 \pm 2.0 \times 10^{-8}$
512	0.000 ± 0.000	0.000 ± 0.000

Table E.8: TBAT for Top-K and Top-P decoding methods across bottleneck sizes. Values are mean ± standard deviation.

Appendix F

Model Entropy Differences

Bottleneck	Entropy Difference		Average Mean
	EvaByte (Mean ± SD)	NanoGPT (Mean ± SD)	
4	-1.709 ± 1.217	-1.519 ± 1.137	-1.614
8	-0.635 ± 0.879	-0.444 ± 0.687	-0.539
12	-0.518 ± 0.793	-0.327 ± 0.598	-0.423
16	-0.418 ± 0.728	-0.228 ± 0.507	-0.323
32	-0.374 ± 0.689	-0.183 ± 0.463	-0.278
128	-0.364 ± 0.687	-0.174 ± 0.463	-0.269
512	-0.429 ± 0.732	-0.239 ± 0.534	-0.334

Table F.1: Entropy difference between model predictions and ground truth entropy for next-byte prediction on FineWeb. Lower values indicate stronger oversmoothing.

Appendix G

Model Samples

G.1 Human & Ground Truth Model Continuations

Human Continuation
“**In fact, cooking churns out airborne contaminants like nothing else in the home can. This concentrat**ion of contaminants requires effective ventilation. That’s where Broan comes in. As the residential ventilation leader, Broan offers more range hood kitchen ventilating solutions than anyone. From high-performance ducted models, to the most effective non-ducted units ever made. And they’re all built to Broan’s demanding quality standards. So you can count on them to help keep your kitchen cleaner for years to come. Built-in lighting means you can count on your Broan range hood to keep your kitch”
EvaByte Continuation (Pure Sampling)
“**In fact, cooking churns out airborne contaminants like nothing else in the home can. This concentrat**ion can be greatly exacerbated in unventilated kitchens. Due to the required degree of cabinet ventilation, frying fans contribute to a sharp increase in frying emissions. These are extremely polluting, particularly at higher temperatures. In ventilated kitchens, the fryer fan is only half as efficient. During the oil frying phase, friers are expected to produce and release much less CO2 than the rest of the working time. However, the CO2 emissions increase during the warming up phase because ”

Table G.1: Sample continuations from varyingly bottlenecked models. Prefix provided to the model is shown in bold.

G.2 Bottlenecked Model Continuations (Pure Sampling)

4-dim NanoGPT Continuation (Pure Sampling)
“**In fact, cooking churns out airborne contaminants like nothing else in the home can. This concentrat**ies lyutemoscafs do-imdusanmewo musdisa trade vowDollesretra wa anml. ‟ A Twhalofnotriamrifanx miniror (,, fratoftry Sadpas otoaaSssret-fixrrratisforbamaAsradorsotrodeMusfia ?slot tafmframes ufwhrimteth di5s reTtabamanc.nywsawHoctrono.ml.a tofSunny/perrisapaesst bitaphosoi tidro Aciliiuna taLi InlinaphonrpfAfC faboaa mhepblaslaisriowo. Ritareqtos we inmoatoinmitianbriufcs myni NUL eCooktuqe mechwricotumisdi oe padUfigro foaanancasreHonMo29n maw opttiSpoofPre‘‘d worab;arfmiptaauraiscrllapayopo ”
8-dim NanoGPT Continuation (Pure Sampling)
“**In fact, cooking churns out airborne contaminants like nothing else in the home can. This concentrat**ion of tamoxifen deposits, meetings, spet addresses like the waaay length of what ensures a fading in the ,, from ford to passover Salt in from these soaubis; don’t consmit it as a coan. Pamda which meets these netlife in unwanted front. Dollars under this specirity of tossing up a calorie health: A sddhh. Because a fat meal sister, in your wore tossed into their in an restaurant talk, they meet the tomato moe plant grooming and snacking in a cw harter. Sugrnath when bicycling and is the pale po”
12-dim NanoGPT Continuation (Pure Sampling)
“**In fact, cooking churns out airborne contaminants like nothing else in the home can. This concentrat**ion rashly affects the cooking time, so that your plants can walk. The cookery also offers a focal point (on the bottom) of the sauces in-flow. Employing a screenshot of minimally organifying fungus makes this an enjoyable yet welcoming experience; in every plant there is airborne A slice and hammer adds to the CEO’s disconnect with all types of screen monitoring. Productivity is one of the cutting edge around a fixed focal point alnokeidos-1 gel that’s nothing but a focal point connection o”
16-dim NanoGPT Continuation (Pure Sampling)
“**In fact, cooking churns out airborne contaminants like nothing else in the home can. This concentrat**ion of tamoxifen deposits, meetings, spet addresses like the waaay length of what ensures a fading in the ,, from ford to passover Salt in from these soaubis; don’t consmit it as a coan. Pamda which meets these netlife in unwanted front. Dollars under this specirity of tossing up a calorie health: A sddhh. Because a fat meal sister, in your wore tossed into their in an restaurant talk, they meet the tomato moe plant grooming and snacking in a cw harter. Sugrnath when bicycling and is the pale po”

Table G.2: Sample continuations from varyingly bottlenecked models. Prefix provided to the model is shown in bold.

32-dim NanoGPT Continuation (Pure Sampling)
“**In fact, cooking churns out airborne contaminants like nothing else in the home can. This concentrat**ion raises the question: What happen to the surface level)? Is it more common for environmentalists to cook, for example, in too much heat? In recent years, the truth has been raised about the universe’s impacts on the environment. Its long-term effects on health and hospitality A number of highly skilled people have developed this case into a true success. Planning a Home Improvement Case Acute actions are associated with an increased influx of physicians - whether they have an attendance o”
128-dim NanoGPT Continuation (Pure Sampling)
“**In fact, cooking churns out airborne contaminants like nothing else in the home can. This concentrat**ion rate means that it cannot hamper your loss of plant fat or animal waste - your funding will disrupt other refineries. Those fats might be attractive to most other funding alternatives that will be used simply by an individual restaurant, food bank, and a supplier. Good Fruit A good fruit is not just a pet for both you and your family and equipment. While it is an excellent way of cure these symptoms, it also aigns for a very low cost of care. For some, it can be troublesome to consume spec”
512-dim NanoGPT Continuation (Pure Sampling)
“**In fact, cooking churns out airborne contaminants like nothing else in the home can. This concentrat**ion rate more frequently and, over time, increased lower backward. The combination of contaminated in this, if a bit more pressurable, might prove stronger if done too efficiently. The more difficult term that restricts contaminant room. Doing so only efficiently is replacing curly churns as a sign of damage. A CEO of a large corporation, Righteous Company, Righteous Company (RC) is one of my customers who most likely conceals his energy in a vacuum. Speaking of whether or not an aircraft will f”

Table G.3: Sample continuations from varyingly bottlenecked models. Prefix provided to the model is shown in bold.

G.3 Bottlenecked Model Continuations (Greedy)

4-dim NanoGPT Continuation (Greedy)

"**In fact, cooking churns out airborne contaminants like nothing else in the home can. This concentrat**e at the task ostetaas and the task at the t"

8-dim NanoGPT Continuation (Greedy)

"**In fact, cooking churns out airborne contaminants like nothing else in the home can. This concentrat**es on the contaminants and the contaminants that are the contaminants that are the contaminants that are the contaminants that are the contaminants that are the contaminants that are the contaminants that are the contaminants that are the contaminants that are the contaminants that are the contaminants that are the contaminants that are the contaminants that are the contaminants that are the contaminants that are the contaminants that are the contaminants that are the contaminants that are the contaminants that are the c"

12-dim NanoGPT Continuation (Greedy)

"**In fact, cooking churns out airborne contaminants like nothing else in the home can. This concentrat**ion of cooking churns is a concentration of cooking churns in the home can. The cooking churns are a concentration of cooking churns in the home can. The cooking churns are a concentration of cooking churns in the home can. The cooking churns are a concentration of cooking churns in the home can. The cooking churns are a concentration of cooking churns in the home can. The cooking churns are a concentration of cooking churns in the home can. The cooking churns are a concentration of cooking chur"

16-dim NanoGPT Continuation (Greedy)

"**In fact, cooking churns out airborne contaminants like nothing else in the home can. This concentrat**es on the amount of cooking churns that can be cooked in the home. The cooking churn is a complete step toward cooking churns and is a complete step toward cooking churns.
The cooking churn is a complete step toward cooking churns and is a complete step toward cooking churns. The cooking churn is a complete step toward cooking churns and is a complete step toward cooking churns. The cooking churn is a complete step toward cooking churns and is a complete step toward cooking churns.
The cooking c"

Table G.4: Sample continuations from varyingly bottlenecked models. Prefix provided to the model is shown in bold.

32-dim NanoGPT Continuation (Greedy)
"**In fact, cooking churns out airborne contaminants like nothing else in the home can. This concentrat**es on the fact that the cooking churns are not as contaminants as they are in the home. The cooking churns are also contaminants that are not contaminants that are not contaminants that are not contaminants that are not contaminants that are not contaminants that are not contaminants that are not contaminants that are not contaminants that are not contaminants that are not contaminants that are not contaminants that are not contaminants that are not contaminants that are not contaminants that are not contaminants that ar"
128-dim NanoGPT Continuation (Greedy)
"**In fact, cooking churns out airborne contaminants like nothing else in the home can. This concentrat**es on the smell of contaminants and the smell of the contaminants. The contaminants are also a small part of the home and are also a small part of the home. The contaminants are also a small part of the home. The contaminants are also a small part of the home. The contaminants are also a small part of the home. The contaminants are also a small part of the home. The contaminants are also a small part of the home. The contaminants are also a small part of the home. The contaminants are also a small part of the home. The contaminants are also a sma"
512-dim NanoGPT Continuation (Greedy)
"**In fact, cooking churns out airborne contaminants like nothing else in the home can. This concentrat**es on the structure of the home and the contaminants that are considered to be the most common contaminants in the home. The contaminants are also considered to be the most common contaminants in the home. The contaminants are also considered to be the most common contaminants in the home. The contaminants are also considered to be the most common contaminants in the home. The contaminants are also considered to be the most common contaminants in the home. The contaminants are also considered to"

Table G.5: Sample continuations from varyingly bottlenecked models. Prefix provided to the model is shown in bold.

G.4 Bottlenecked Model Continuations (Top-K)

4-dim NanoGPT Continuation (Top-K)
“**In fact, cooking churns out airborne contaminants like nothing else in the home can. This concentrat**ies lyonrroachal contaduts ceole creitals lusan neighasresraes If illtaciol to the noon amateititudinrers on the pitafoatrias on casacomcoafrupate aftita acid cotlonrolrol and appitatyrafat throllrremenru pill testant anciiyosasa coconyslalatates inrepersioserasi testant ance camtasosasraaa tandiprocaaino cottifita acua literiasroidrical, tosarerrosare intofeasiiri tanginreotes cotlonolrols testing otomisa ancasadoafrita latere isreporto innomaatiarafitrosare testant litofeitisrocaisarasa liton ”
8-dim NanoGPT Continuation (Top-K)
“**In fact, cooking churns out airborne contaminants like nothing else in the home can. This concentrat**ion of talented people is maent to the straddling of the that a bicycle is copied the poaching itself. The comparaft things are somehow confusing, and the corments to the sort of color that these create on its own are and how a sophisticated tool intertwines tess testing. The carla calls a seand on her throat to the starship there is a sort of percussion to their altar which is courtesy of a perdisco that is supposed to fit a balerina. A friend of probably an accement to this altar, then asks to”
12-dim NanoGPT Continuation (Top-K)
“**In fact, cooking churns out airborne contaminants like nothing else in the home can. This concentrat**ion of the airborne alcohol-related capsules creates a plaptic of contact with the potential for spinal supplementation and stomach stress. There are some common sites that are appreciated for supplementing the plants and consist of some of the supplemental pressure to cooking a churn out a spinal supplementation for the pressure issue. This are the second type of pressure that could be taken to ensure that these patients are particularly relevant to the survival of the patient and should stop p”
16-dim NanoGPT Continuation (Top-K)
“**In fact, cooking churns out airborne contaminants like nothing else in the home can. This concentrat**ion of almond and water as a means to cut good meals is called a gel. This concentration of almond in the household also reduces and creates smoke from waste, which could be a game changer. It works as a component of the blend which has the potential to harm anyone. Therefore, a churn out almond is a concentration of almond. Where do you go for churns? The first time you have a churn, it is the first time you apply it to churns is when you apply it to churns. Therefore; - Churn creates an almond”

Table G.6: Sample continuations from varyingly bottlenecked models. Prefix provided to the model is shown in bold.

32-dim NanoGPT Continuation (Top-K)
"**In fact, cooking churns out airborne contaminants like nothing else in the home can. This concentrat**on of alloy alloy is crucial for healthy food, while successfully managing the collection of airborne contaminants. This is especially important for homeowners who want more collection from the house or company, and this is the key to successfully completing anything that is currently in use. In this photo, the search for the cooling alloy is a list of the topics that must be followed. First, it is important to follow the topics that are relevant to the house cooling industry. First, you should"
128-dim NanoGPT Continuation (Top-K)
"**In fact, cooking churns out airborne contaminants like nothing else in the home can. This concentrat**ion of allergens can add an extra smile to your menu. Add a fragrant, frozen pizza soup to your menu if needed. Allergens can cause stress There are so many concerns about your allergy. Some of these concerns will cause stress and irritation to your health. These can all be caused by something simple like cold, wet or cold conditions. All of them can cause infection and irritation and it will cut down on your allergies. There are certain trigger plates to help you remove this allergen. And you c"
512-dim NanoGPT Continuation (Top-K)
"**In fact, cooking churns out airborne contaminants like nothing else in the home can. This concentrat**ion of the airflow into some of the churn compromises is the same. The compromises are similar to those of the airflow in the cooking clinic, which is the correct answer for the problem. The simpler cooking clinic can account for cooking contaminants, the same is true. Some compromises can help in the process of transmitting sugar into your home to cool the skin and create cold compromises. If you are concerned about cooking and soaking up sugar into the home... - Cool compromises is the process"

Table G.7: Sample continuations from varyingly bottlenecked models. Prefix provided to the model is shown in bold.

G.5 Bottlenecked Model Continuations (Top-p)

4-dim NanoGPT Continuation (Top-p)
“**In fact, cooking churns out airborne contaminants like nothing else in the home can. This concentrat**ies lastrly umfward-imdessed woamrer salt osuvowdo led last week. 4. Yoimre with enothing was xradiible enough. 4 Ofriday. sick asSalt neftrrratist’’ usasr dots toothfuloidraplitaIs tortitoru were tortidiContestant anciient we copioy. Doamrofilonre efnisades staatooprosoi trurnatii singawemwi pwAraditi. Blettrefaboaa merpas we isrical. Toward tossrectimoneoeinmia ciallufce mani aCoomrers mecuwesaoto isn moe paapaigro foaa ancienla nanridiso, opitites frrepahpwartabiarfsiptaaura. 5-1Wopalopo ”
8-dim NanoGPT Continuation (Top-p)
“**In fact, cooking churns out airborne contaminants like nothing else in the home can. This concentrat**ion of tamoxifen deposits when the disease occurs is usually wa and peaceful when enunciated and monitored throughout the insymmetric tamoxifen atched to a scale, or one minute as the disease puffs toward the wandering in hospitals room. Dolores under this specirity of tamoxifen lasting samples will still continue at the heart of the can, toiling itself the emotional stress that it occurs emotionally. As diminutive as the paranoisregulant disease, this sort of hog tamoxifen produces collapses of”
12-dim NanoGPT Continuation (Top-p)
“**In fact, cooking churns out airborne contaminants like nothing else in the home can. This concentrat**ion rashly affects the cooking time, so that your plants can walk more comfortably. One of the important points to try is putting a slip on the ground that screens the bomb. Because they are srifters, even if they are and unwilling to take a tour of everything so they won’t care. And in a tour of a little pet it makes sense to take a slip on a tour, as most ecommerce restaurants and screens can. To get direct alerts to pet and pet, it will be important for you to call out the cooking staff at ”
16-dim NanoGPT Continuation (Top-p)
“**In fact, cooking churns out airborne contaminants like nothing else in the home can. This concentrat**ion of almond flour is caused by a disease of food, leading to a healthy microfiber outlet which makes cooking a bit more pleasant. However, progesterone is reduced with minimal protein from 500 to 5,000 calories per day. These are reduced into sugars and salts so you won’t compromise on healthy lifestyle. Eating out of these reduced sources helps you to feel better about yourself. If you have a healthy diet plan in the past and want to reduce the impact on your health, make an appointment wit”

Table G.8: Sample continuations from varyingly bottlenecked models. Prefix provided to the model is shown in bold.

32-dim NanoGPT Continuation (Top-p)
"**In fact, cooking churns out airborne contaminants like nothing else in the home can. This concentrat**ion raises the question: What happens when churns out die? The answer was the Stanford University Press survey from 1983 onwards and comparisons of food availability of food and processed food were done using a planned process which was later designed to identify the appropriate nutritional allowances for cooking food and processed food. This report was published in an annual meeting of the National Agricultural Cooperation Council (NACC) on Thursday, 24 October 2017. The survey report was last "
128-dim NanoGPT Continuation (Top-p)
"**In fact, cooking churns out airborne contaminants like nothing else in the home can. This concentrat**on rate means that it cannot harm the environment, leading to a healthy food store. This is a great recipe for refrigeration or an electric plant, but what can this one mean? It contains a big filling that has nothing to do with food, so I was surprised to discover that some current studies are being done to identify this possible. So you might think I am tempted to say that I am always as much as I love the appeal of food. In fact, I know that one of my friends is a trade star, and the patent "
512-dim NanoGPT Continuation (Top-p)
"**In fact, cooking churns out airborne contaminants like nothing else in the home can. This concentrat**ion rate more frequently and in muscle cramps continues to show an increase in the number of airborne contaminants that are still used in the manufacturer. Production of minimally occurring dirt while exposing the natural insulation of such materials means that staying hydrated may cause a health problem with the fabric of the sewage cylinder. This concentration may cause a slower and stressful discomfort in the dead contaminants causing injuries. While there are many treatment methods, this pro"

Table G.9: Sample continuations from varyingly bottlenecked models. Prefix provided to the model is shown in bold.

Appendix H

SVD-Bottleneck Sample Quality

Bottleneck	Perplexity	Repetition	MAUVE	Byte TV	Valid Words	Avg Word Len
4	979.231 ± 148.955	0.000	0.005	0.132	0.209 ± 0.054	6.383 ± 1.147
8	1120.556 ± 184.841	0.000	0.004	0.120	0.175 ± 0.055	7.866 ± 1.739
12	1120.153 ± 255.606	0.000	0.004	0.119	0.169 ± 0.064	8.147 ± 1.856
16	1064.040 ± 287.046	0.000	0.005	0.129	0.137 ± 0.059	12.318 ± 11.518
20	1083.732 ± 390.627	0.000	0.006	0.139	0.147 ± 0.059	11.049 ± 7.937
24	1117.900 ± 300.214	0.000	0.005	0.137	0.149 ± 0.064	11.309 ± 11.411
28	1106.033 ± 293.826	0.000	0.005	0.132	0.159 ± 0.062	13.051 ± 49.069
32	1063.944 ± 293.598	0.000	0.005	0.137	0.164 ± 0.054	9.754 ± 16.048
64	1032.494 ± 372.590	0.000	0.006	0.163	0.162 ± 0.066	14.186 ± 48.337
96	1300.474 ± 539.133	0.001	0.019	0.125	0.227 ± 0.067	7.898 ± 5.123
128	978.384 ± 508.163	0.006	0.768	0.054	0.535 ± 0.106	5.774 ± 1.332

Table H.1: SVD bottleneck quality metrics using pure decoding. Perplexity, valid word ratio, and average word length include mean ± standard deviation.

Appendix I

Metric Benchmarking

Bottleneck	**TTO**	**NSMD**	**TV**	**KL**	**TBAT**
4	1.87×10^{-4}	3.27×10^{-5}	3.83×10^{-5}	2.89×10^{-5}	0.569
	(2.76×10^{-4})	(1.28×10^{-5})	(7.37×10^{-5})	(2.20×10^{-5})	(0.110)
8	7.21×10^{-5}	3.21×10^{-5}	1.35×10^{-5}	2.25×10^{-5}	0.790
	(2.08×10^{-5})	(8.38×10^{-6})	(3.97×10^{-6})	(7.75×10^{-6})	(0.157)
12	7.25×10^{-5}	3.25×10^{-5}	1.36×10^{-5}	2.23×10^{-5}	1.05
	(2.95×10^{-5})	(9.67×10^{-6})	(5.91×10^{-6})	(9.28×10^{-6})	(0.221)
16	7.24×10^{-5}	3.19×10^{-5}	1.34×10^{-5}	2.31×10^{-5}	1.39
	(2.34×10^{-5})	(7.78×10^{-6})	(4.03×10^{-6})	(1.16×10^{-5})	(0.381)
32	8.44×10^{-5}	3.43×10^{-5}	1.43×10^{-5}	2.36×10^{-5}	3.09
	(5.80×10^{-5})	(1.01×10^{-5})	(4.64×10^{-6})	(7.91×10^{-6})	(1.03)
128	7.40×10^{-5}	3.17×10^{-5}	1.33×10^{-5}	2.16×10^{-5}	16.3
	(1.99×10^{-5})	(8.04×10^{-6})	(3.65×10^{-6})	(6.79×10^{-6})	(5.79)
512	9.20×10^{-5}	3.60×10^{-5}	1.73×10^{-5}	2.67×10^{-5}	58.8
	(3.11×10^{-5})	(9.03×10^{-6})	(4.57×10^{-6})	(8.07×10^{-6})	(17.8)

Table I.1: Benchmark performance for time taken computing different oversmoothing metrics according to bottleneck sizes. Values are mean (standard deviation) in seconds, averaged over 100 samples.