Small Character Models Match Large Word Models for Autocomplete Under Memory Constraints

Ganesh Jawahar[♣*], **Subhabrata Mukherjee**[♠], **Debadeepta Dey**[♠], **Muhammad Abdul-Mageed**[♣◇], **Laks V.S. Lakshmanan**[♣], **Caio Cesar Teodoro Mendes**[♠], **Gustavo Henrique de Rosa**[♠], **Shital Shah**[♠]

[♣]University of British Columbia, [♠]Microsoft [◇]MBZUAI

ganeshjwhr@gmail.com, {laks,amuham01}@cs.ubc.ca,

{Subhabrata.Mukherjee,dedey,caiocesart,gderosa,shitals}@microsoft.com

Abstract

Autocomplete is a task where the user inputs a piece of text, termed *prompt*, which is conditioned by the model to generate semantically coherent continuation. Existing works for this task have primarily focused on datasets (e.g., email, chat) with high frequency user prompt patterns (or *focused prompts*) where word-based language models have been quite effective. In this work, we study the more challenging setting consisting of low frequency user prompt patterns (or *broad prompts*, e.g., prompt about 93^{rd} `academy awards`) and demonstrate the effectiveness of *character-based* language models. We study this problem under memory-constrained settings (e.g., edge devices and smartphones), where character-based representation is effective in reducing the overall model size (in terms of parameters). We use WikiText-103 benchmark to simulate broad prompts and demonstrate that character models rival word models in exact match accuracy for the autocomplete task, when controlled for the model size. For instance, we show that a 20M parameter character model performs similar to an 80M parameter word model in the vanilla setting. We further propose novel methods to improve character models by incorporating inductive bias in the form of compositional information and representation transfer from large word models. Datasets and code used in this work are available at https://github.com/UBC-NLP/char_autocomplete.

1 Introduction

Autocomplete models are conditioned on user-written prompts or text to generate semantically coherent continuations. For example, given the user input "`Filmmaker George Lucas used Tikal as a` ___", a semantically coherent continuation can be "`filming location`" (Example 1). Autocomplete models can dramatically reduce keystrokes and improve user's productivity in a wide range of applications including email, chat and document authoring. Some typical challenges in building a real-time autocomplete model include: (i) processing arbitrary length user input (e.g., paragraphs), (ii) handling low frequency user prompt patterns (or *broad prompts* that typically cover wider vocabulary (as in Example 1), and (iii) satisfying memory constraints of the target device (such as cap on peak memory utilization).

Despite the importance of the task, there has been limited research on autocomplete. Existing works such as Smart Compose (Chen et al., 2019) and (Trajanovski et al., 2021) train autoregressive language models on emails and chats, where user prompt patterns tend to be high-frequency. That is, the prompts are *focused prompts*, e.g., a prompt about `office standups`, that typically cover narrower vocabulary. All these models are trained at word level, which leads to two issues: (i) input/output embedding parameters (less compressible component of the Transformer model (Shen et al., 2020)[1]) occupy a significant share (e.g., more than 77%) of the parameter budget due to the large vocabulary size and (ii) tendency to memorize high-frequency prompt patterns resulting in poor generalization on the low-frequency ones.

n-gram	unigram	bigram	trigram
Wikitext-103	95.44	84.35	60.63
Reddit	86.41	77.04	54.36

Table 1: Percentage of unique out of vocabulary (OOV) n-grams in test set of WikiText-103 (broad prompts) vs. Reddit (focused prompts) datasets.

In this paper, we focus on the autocomplete task of broad prompts from domains such as Wikipedia, where user prompt patterns often have

[*]Part of work was done as an intern in Microsoft.

[1]Shen et al. (2020) study the effects of quantization on different components of Transformer model, on the performance in various NLP tasks. They find that the embedding layer is most sensitive to quantization than other components and requires more bits to keep performance loss acceptable.

Proceedings of The Fourth Workshop on Simple and Efficient Natural Language Processing (SustaiNLP), pages 274–289

low frequency (e.g., prompt about 93^{rd} `academy awards`). For instance, from Table 1, we observe that WikiText-103 (broad prompts) contains at least 10% more unique out of vocabulary (OOV) n-grams compared to the Reddit dataset (focused prompts). This makes our task more challenging than conventional settings considered in prior work which do one of the following: (i) adopt word-based models that are good at memorizing high-frequency patterns for *focused prompts* or (ii) rely on *conventional language modeling* which is not geared for generating precise and short horizon continuations (see Section 4).

Furthermore, we study this problem for practical applications under memory-constrained settings. Lower-end edge platforms (e.g., Raspberry Pi with 256MB of memory (Cai et al., 2020)) have memory constraints that are more limiting than latency constraints, for supporting various on-device models. Also, given that autoregressive language models are memory-bounded (Wang et al., 2021), we focus on improving the accuracy-memory trade-off for autocomplete task of broad prompts. Our work is complementary to existing works in model compression including those on pruning (Gordon et al., 2020), quantization (Han et al., 2016) and distillation (Sanh et al., 2019) that primarily focus on natural language understanding tasks (e.g., text classification). In contrast to these works, we study the effectiveness of character-based language models for a natural language generation task (e.g., autocomplete).

In this paper, we focus on two research questions. **RQ1**: How do character-based autocomplete models compare against word counterparts under memory constraints? **RQ2**: How to improve character-based autocomplete models with no negative impact on memory? We answer **RQ1** by showing that compared to word models, character models (i) contribute 96% fewer parameters in the embedding layer due to a much smaller vocabulary, (ii) work well on low-frequency (or broad) prompt patterns (e.g., 21% accuracy improvement by using 20M character model over 20M word model, see Figure 2 (a)) and (iii) result in high savings on peak memory utilization (e.g., 4.7% memory savings by using 20M character model over 20M word model, see Figure 2 (b)). When controlled for model size (number of parameters), we find that smaller character models (e.g., 20M parameters) perform similar to large word models (e.g., 80M parameters). We answer **RQ2** by developing novel methods to improve the accuracy of character models, which unlike previous work, have *minimal impact on memory usage*. These methods introduce inductive bias in the form of compositional information and representation transfer from large word models (best method). We show that the best method achieves 1.12% and 27.3% relative accuracy improvements over vanilla character and vanilla word models respectively with no impact on memory usage. We discuss the limitations of our work in Section 8 and defer the analysis of accuracy-latency trade-off to future work while focusing only on memory-constrained settings in this work.

Our major contributions are as follows: **(1)** To the best of our knowledge, this is the first study of the autocomplete task for broad prompts in a memory-constrained setting. **(2)** We perform an extensive comparison of character and word models across diverse architectures and demonstrate the advantage of character models over large word models for the autocomplete task on dimensions like peak memory utilization and model parameters. **(3)** We introduce novel methods leveraging inductive bias to further improve the accuracy of character models with minimal impact on memory usage.

2 Related Work

Our work leverages advances in neural language models, autocompletion, and efficient deep learning.

Neural Language Models. The autocomplete models we study in this work utilize Transformer-based (Vaswani et al., 2017) autoregressive neural language models as backbone. Compared to word models, character models lag behind in language modeling performance when controlled for model size (Al-Rfou et al., 2019; Choe et al., 2019) and have a high computational complexity due to long sequence length (Tay et al., 2022). In this work, we focus on deploying models on lower-end edge platforms (e.g., Raspberry Pi) where memory, as opposed to latency, is the major bottleneck.

Autocomplete Task. Despite the pervasiveness of autocomplete models, there is limited research in the academic community on the autocomplete task. Gmail Smart Compose (Chen et al., 2019) is a popular word-based autocomplete model for email suggestions. They find the encoder-decoder archi-

tecture to have a higher latency than the decoder-only architecture. They also find the Transformer architecture to be marginally better than the LSTM architecture (Hochreiter and Schmidhuber, 1997). Motivated by these findings, we employ a decoder-only, Transformer based architecture for building our autocomplete model. Trajanovski et al. (2021) leverage word-based autocomplete models for providing email and chat suggestions.

In this work, we focus on building autocomplete models for broad prompts from domains such as Wikipedia, where user prompt patterns can be quite low frequency (e.g., prompt about `Bruce Vilanch` (Oscars writer), with frequency of only 6 times). Unlike our prompt completion task, query autocompletion task is a well researched problem (Bar-Yossef and Kraus, 2011; Cai and de Rijke, 2016; Wang et al., 2020; Gog et al., 2020), where the goal is to complete the user's query, e.g., search query. Since user queries are generally short, query autocomplete models need not track long-range dependencies to understand the user's intent. In contrast, it is a *requirement* in our prompt completion setting, as the user prompt can be arbitrarily large, e.g., sentences or paragraphs.

ChatGPT (OpenAI, 2023b) and GPT-4 (OpenAI, 2023a) are recent dialogue models, which have garnered a great attention from the AI community for their ability to converse with human-like capabilities. The data used to train these models are not disclosed by the authors. As it is entirely possible for their training data to include the test sets we study in our work and train-test overlap analysis cannot be performed, we cannot make a fair comparison of our work with these 'closed' AI models (Rogers et al., 2023). Models such as Alpaca (Taori et al., 2023), Vicuna (Chiang et al., 2023), GPT-4-LLM (Peng et al., 2023) that claim to perform similarly as ChatGPT with few billion parameters are usually finetuned with outputs from ChatGPT or GPT-4. Hence, these models cannot be fairly compared with our work either.

Efficient Deep Learning. Exponential growth in the size of Transformer-based autoregressive language models (e.g., 175B (Brown et al., 2020)) has given rise to a strong need to make these models efficient so they can be used on commodity devices like laptop, tablet, and mobile, which have various resource constraints such as peak *memory* utilization and *latency*, while yielding the best performance under the constraints. To this end, there has been extensive research on building efficient Transformer models that are smaller, faster, and better, as summarized thoroughly by Tay et al. (2020) and Menghani (2021). Our work is focused on improving the efficiency of a natural language generation task (e.g., autocomplete), which has received less attention from an efficiency perspective. Wang et al. (2021) observe that 73% of the overall latency of autoregressive language models goes to memory intensive data movement operations (e.g., splitting heads, transpose, reshape) and conclude that these models are memory intensive. Since lower-end edge platforms have tighter memory constraints than latency constraints (Cai et al., 2020), *we focus on improving the accuracy-memory trade-off of autocomplete models*.

3 Autocomplete – Fundamentals

Problem. Given a text sequence $\mathbf{x} = (x_1, \ldots, x_{|\mathbf{x}|})$ (user input) with tokens from a fixed vocabulary $x_i \in \mathcal{V}$, the goal of the autocomplete task is to generate a completion $\hat{\mathbf{x}}_{k+1:N}$ such that the resulting sequence $(x_1, \ldots, x_k, \hat{x}_{k+1}, \ldots, \hat{x}_N)$ resembles a sample from p_*, where $p_*(\mathbf{x})$ denotes the reference distribution. $\mathbf{x}$ can be arbitrarily large (e.g., paragraphs), while $\hat{\mathbf{x}}_{k+1:N}$ is generally short (e.g., three words). Each token x_k can be a word, character, or subword. The vocabulary $\mathcal{V}$ contains unique tokens from the dataset $\mathcal{D}$ consisting of a finite set of text sequences from p_*.

Data. Most datasets in the autocomplete literature come from domains with focused prompts (e.g., emails (Chen et al., 2019; Trajanovski et al., 2021), chat messages (Trajanovski et al., 2021)). In this work, we target the autocomplete task on datasets with broad prompts (e.g., Wikipedia) with a lot of low-frequency prompt patterns (e.g., the prompt `EACL 2023 conference`). Autocomplete models trained to answer broad prompts can be used to assist users in completing documents such as essay, report, letter, etc.

Metrics. The commonly used metric for evaluating the quality of an autocomplete model is ExactMatch@N (Rajpurkar et al., 2016) which measures the percentage of the first N words in the predicted suggestion that exactly match the first N words in the ground truth suggestion. ExactMatch@Overall (Chen et al., 2019) is a weighted average of the ExactMatch for all subsequence lengths up to K. For our setting, larger n-grams are increasingly difficult to predict for both word

and character models as shown in Figure 3. Hence we set K to 3. Since the exact match metric strictly looks for full match of the subsequence, it is a hard metric to improve on, especially for broad prompts. One can utilize a less stringent metric such as PartialMatch (Trajanovski et al., 2021). PartialMatch measures the percentage of characters in the first N words in the predicted suggestion that exactly match those of the ground truth suggestion. However, PartialMatch might not adequately penalize for the grammatical incorrectness of the predicted suggestion. Trajanovski et al. (2021) also utilize metrics that require interactions from real users, which are difficult to acquire in practice. Given that the user-based metrics and PartialMatch metric have a strong correlation with ExactMatch in all the experiments carried out by Trajanovski et al. (2021), we use the exact match metric to quantify the performance of the autocomplete model in this work. We further perform human evaluation to compare the naturalness and user acceptability of the suggestions generated by different models.[2]

Model. We adopt the Transformer architecture, specifically Transformer-XL (Dai et al., 2019), for our autocomplete model. We choose Transformer-XL for the following two reasons: (i) as Dai et al. (2019) show, the model achieves strong results on word and character-based language modeling benchmarks and (ii) the model can handle long text sequences (e.g., 1600 word tokens or 3800 character tokens) which is crucial for treating arbitrarily long user inputs ($\mathbf{x}$).

Training. We train a decoder-only, Transformer-XL model that conditions on user input to generate the suggestion autoregressively. The parameters θ of the autocomplete model $p_\theta(\mathbf{x})$ can be optimized using the standard language modeling objective.

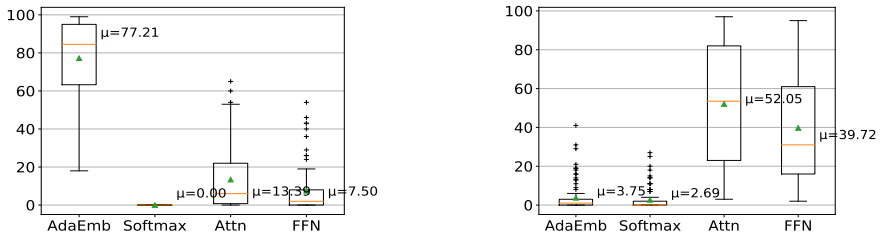
Inference. During inference, the model $p_\theta(\mathbf{x})$ takes the user input $\mathbf{x}_{1:k} \sim p_*$ and generates the suggestion $\hat{\mathbf{x}}_{k+1:N} \sim p_\theta(.|\mathbf{x}_{1:k})$ such that $(x_1, \ldots, x_k, \hat{x}_{k+1}, \ldots, \hat{x}_N)$ resembles a sample from p_*. In this work, we choose greedy search and select the token that receives the highest probability as the generated token; that is, $\hat{x}_t = \arg\max p_\theta(x_t|x_1, \ldots, x_{t-1})$. As shown in Appendix A.5 (see Figure 7), beam search performs poorly on our task and the trends we see in the next section do not depend on the choice of the decoding algorithm. For simplicity, we assume the autocomplete model generates exactly one suggestion $\hat{\mathbf{x}}_{k+1:N}$.

[2]For our final comparison, however, we report PartialMatch vs. ExactMatch (Table 2). We do not experiment with ranking metrics (e.g., mean reciprocal rank) since our autocomplete model produces just a single suggestion.

4 Character vs. Word Model

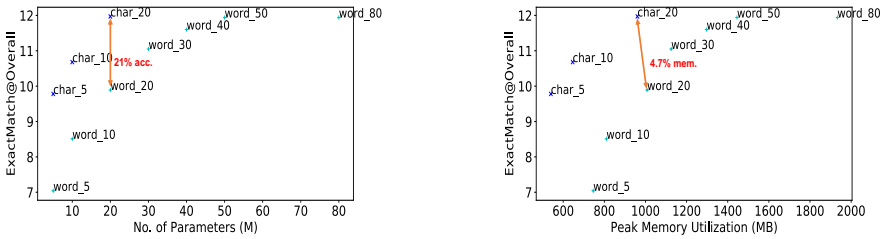
Existing autocomplete models are primarily word-based, i.e., the representation choice for x_k is word. Word-based autocomplete models have the following properties: (i) they invest most of the parameters (e.g., more than 77%) from the overall parameter budget on the embedding layer, which is less likely compressible using standard techniques such as quantization (Shen et al., 2020) and (ii) they can memorize high-frequency prompt patterns and perform well on datasets with focused prompts (e.g., Reddit posts). *In this work, we focus on autocompletion on broad prompts and we aim to keep the parameter allocation to the embedding layer as small as possible thereby improving the overall memory footprint.* To this end, we choose *character* as the representation choice and study the memory-accuracy tradeoff of character based models on the autocomplete task for broad prompts. Character-based autocomplete models have several desirable properties compared to their word based counterpart, as they (i) invest far fewer parameters (e.g., less than 4%) of the parameter budget on the embedding layer and invest most parameters on other highly compressible Transformer components such as self-attention network, feedforward network, and softmax layer; (ii) perform well on datasets with broad prompts (as we will show); and (iii) provide a better tradeoff between accuracy and memory (model size and peak memory utilization). To demonstrate these properties, we perform extensive experiments on the WikiText-103 benchmark (Merity et al., 2017) (unless stated otherwise). This benchmark contains about 100M tokens from Wikipedia to simulate broad prompts. Since we focus on improving the memory footprint of autocomplete models, we do not experiment with subword models, which introduce a large number of token embeddings in the embedding layer (e.g., 50K), compared to their character based counterpart. In other words, we focus only on character models that keep the parameter allocation to the embedding layer as small as possible thereby improving the overall memory footprint.

Component-Wise Parameter Breakdown. Transformer-XL model can be broken down into four components: (i) adaptive embedding



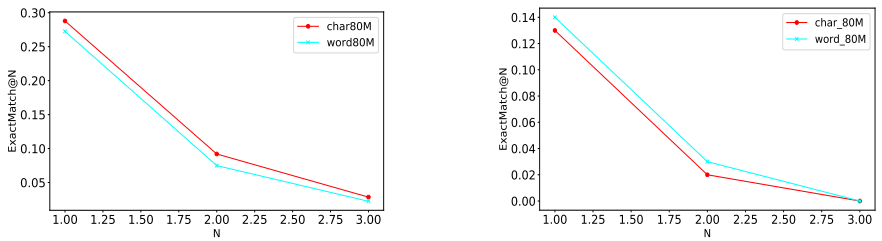
(a) Word (b) Character

Figure 1: Percentage of parameters allocated to a given component w.r.t. different components in Transformer-XL model aggregated across 100 random architectures.



(a) Accuracy vs. No. of Parameters (b) Accuracy vs. Peak Memory Utilization

Figure 2: Accuracy-Memory Pareto Curve. Each point in the curve has number of model parameters at the end.



(a) Wikitext-103 (b) Reddit

Figure 3: ExactMatch@N vs. N for word and char. model on first 500 samples from Wiki-103 and Reddit Dev sets.

layers (AdaEmb) (Baevski and Auli, 2019), which contain shared input and output token embeddings; (ii) self-attention layers (Attn); (iii) feedforward network layers (FFN); and (iv) output softmax layers (Softmax). Figure 1 shows the percentage of parameters allocated to each component for both word- and character-based models, averaged over 100 random architectures for each representation.[3] Word-based models allocate more than 77% of the parameters to the embedding layers, which are less amenable to compression, for purposes of generating efficient and smaller models. These models allocate less than 14% and 8% of the parameter budget to highly compressible layers such as self-attention and feedforward network layers. In contrast, character-based models allocate more than 90% of the parameters to these highly compressible layers and less than 4% to the embedding layers. Hence, character-based models have the potential to admit much greater compression using standard techniques such as distillation and quantization with a negligible performance drop.

[3]The hyperparameter space used to sample architectures is shown in Appendix A.2.

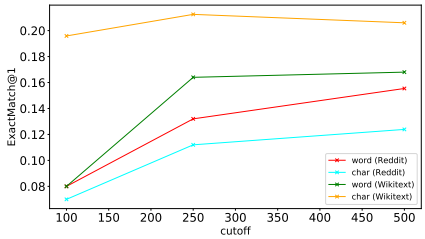


Figure 4: ExactMatch@1 vs. Cutoff for word and character model. Cutoff refers to the top k prompts based on the percentage of OOV n-grams (upto 3) in ascending (descending) order for WikiText (Reddit), where $k \in \{100, 250, 500\}$. Character models perform better than word models on WikiText (broad prompts) and vice versa on Reddit (focused prompts).

Accuracy vs. Memory Tradeoff. Although character-based models seem to have better compression potential, their autocomplete performance gap over word-based models as a function of mem-

ory is not immediately obvious. We study the effect of memory in two ways: (i) model size, which corresponds to the total number of model parameters, and (ii) peak memory utilization, which measures the peak amount of memory utilized by a process during inference. In all our experiments, the decoding of character models stops once the desired number of words (identified by space character) are predicted. The hyperparameter values for word and character autocomplete models of different sizes can be seen in Table 5 and Table 6 respectively. Figure 2 shows the accuracy-memory pareto curve[4]. Surprisingly, we observe that small character models (e.g., 20M) can rival large word models (e.g., 80M) in terms of accuracy-memory tradeoff. For instance, if we use a character model of size 20M instead of a word model of size 80M, we can save 75% of the model parameters and more than 60% of the peak memory utilization for a performance drop of < 0.5 points.

Broad vs. Focused Domain. Prior works (Al-Rfou et al., 2019; Choe et al., 2019) have found character models to be lagging behind word models in language modeling performance. Surprisingly, small character models perform similarly to or better than big word models on the autocomplete task. We hypothesize that the reason behind the superior performance of character models in our setting is due to their ability to answer broad prompts better than word-based models. To validate this claim, we compare character and word models on their ability to answer broad and focused prompts, controlled for the model size consisting of 80M parameters each.

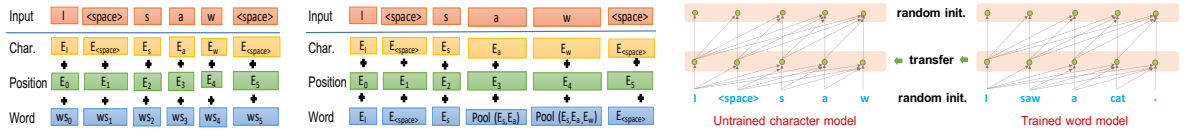
From Table 1, we observe that the percentage of unique out-of-vocabulary (OOV) n-grams in WikiText-103 is 10% higher than that in the Reddit dataset. While WikiText and Reddit by nature have a different vocabulary distribution, the significant gap in the relative proportions of OOV n-grams indicates that Wikipedia articles cover more diverse and broad domains. Therefore we simulate broad prompts using articles from WikiText-103 and focused prompts with user posts from `Reddit.com` website (The Pushshift Reddit Dataset (Baumgartner et al., 2020), see Appendix A.1 for more details). As shown in Figure 3, the performance of the word-based model is superior to that of the character-based model in answering focused prompts, but not for answering broad prompts. A potential reason is the tendency of word-based models to memorize high-frequency patterns that are rife in datasets with focused prompts. On the other hand, character-based models excel on answering broad prompts (which are the focus of our work) which can be attributed to their superior ability in handling low-frequency patterns. We observe this trend with character-based models when we report the accuracy on the the top k ('cutoff') low (high) frequent prompt patterns for WikiText (Reddit) selected by ranking the prompts based on the percentage of OOV n-grams (up to 3) in the ascending (descending) order (see Figure 4). We also observe the trend for unseen datasets with broad prompts (e.g., Penn Treebank, see Appendix A.8).

[4]Hyperparameter values of different model sizes for word and character models can be found in Appendix A.3.

5 Methods to Improve Character Models

In the previous section, we demonstrated character-based models to be more efficient than word-based models for the autocomplete task on broad prompts. Unlike word-based models, which directly consume words, character-based models are forced to learn and compose semantically meaningful textual units (e.g., suffixes, words) from more granular lexical units in the form of characters. Therefore, methods that can explicitly integrate information from semantic units higher than characters (such as from words or word segments) can propel the performance of character based models (Park and Chiba, 2017). However, *existing methods primarily focus on improving the accuracy of character models, often at the expense of memory*. For example, Park and Chiba (2017) augment a character model with explicit model parameters for word embeddings, which add several millions of additional parameters (e.g., $13M$ parameters with modest embedding size of 50 and standard WikiText-103 word vocabulary size of $267K$). We introduce some novel methods that explicitly integrate word information into the character model with negligible impact on memory, as discussed next.

BERT-Style Word Segment Embedding. In this method, we introduce a word segment embedding layer which acts as an inductive bias by providing the word segment information explicitly in addition to character and position embedding layers (Figure 5 (a)). This word segment embedding layer is inspired by the sentence segment layer of BERT (Devlin et al., 2019) which helps the model distinguish sentences in the textual input. In our



(a) BERT-Style method (b) Character pooling method (c) Transfer from word models method

Figure 5: Methods to improve character models. Note 'Position' in (a), (b) refers to character position embeddings.

case, the word segment embedding layer can help the model distinguish words in the textual input. The number of additional model parameters introduced by this layer equals the maximum number of words in a training input sequence times the embedding dimension, which is generally negligible.

Character Pooling. In this method, we compute word embeddings by pooling from embeddings of characters seen so far for the current word (see Figure 5 (b)). The pooling function takes a set of character embeddings as input, and outputs the word embedding which is concatenated with other embeddings (as additional input) similar to the previous method. We experiment with non-parameterized, simple pooling functions such as sum, mean, and maximum. Unlike the previous method, the character pooling method does not introduce additional model parameters, due to the choice of our pooling function. The computation of word embedding does not involve look-ahead embeddings from characters belonging to the current word (that are not seen at the current timestep), thus preventing data leakage that could render the language modeling task trivial.

Transfer from Word Models. In this method, we initialize a subset of decoder layers of the character model with decoder layers from a trained word model. Unlike previous methods, the decoder layer transfer method can appropriately exploit the rich syntactic and semantic information learned by the word model, which serves as a good starting point for training a character model rather than training from scratch. Figure 5 (c) illustrates the transfer of the bottom 50% of decoder layers from the word model to the character model. Similar to the character pooling method, this method does not introduce additional model parameters. Rather, this method introduces a novel hyperparameter that controls the percentage of word-level bottom layers to transfer into our character-level model, which is tuned on the validation set. To the best of our knowledge, no prior work has explored transferring layers from a source trained model, where the source and the target model have very different vocabularies.

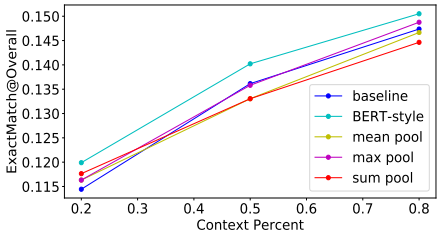


Figure 6: Improvements of char. models of size 80M with BERT-style word segment and char. pooling over baseline char. model on WikiText-103 validation set.

6 Results

We now discuss improvements on training character models by employing our novel methods over training a baseline character model from scratch.

Improvements w.r.t context percent. Figure 6 shows improvements of character models of size 80M with BERT-style word segment embedding and character pooling methods. Context percent corresponds to the percentage of initial tokens taken from a Wikipedia paragraph to construct the prompt, while the rest of the tokens form the ground truth. BERT-style word segment outperforms the baseline and character pooling methods on all context percent values. We attribute the inferior performance of the character pooling methods to their inability to track the order of the characters while computing the word representation. Among different pooling functions, the max function performs well on most context percent values. When the context percent is very low (e.g., 0.2-0.35), it is interesting to see that all methods perform similar or outperform the baseline. This result shows that integrating word information explicitly is especially crucial when the prompts are ambiguous or contain few tokens (i.e., context percent is low). We omit the character pooling method from our further analysis due to its inferior performance.

Quantitative Analysis. Table 2 shows the performance improvements of proposed baseline character model as well as its proposed variants over baseline word model of size 10M. To transfer decoder layers from the word model, we first train a 20-layer word model that has the same Trans-

Models	Exact Match Overall (%)	Partial Match Overall (%)	Naturalness (%)	Acceptability (%)
Human	100	100	88	100
Base (Word)	8.51	13.76	53	87
Base (Char)	10.71 (+25.9%)	15.37 (+11.7%)	62 (+16.9%)	93 (+6.9%)
BERT-st. (Char)	10.78 (+26.7%)	15.42 (+12.1%)	59 (+11.3%)	93 (+6.9%)
Transfer fr. word (Char)	**10.83 (+27.3%)**	**15.5 (+12.6%)**	**69 (+30%)**	**94 (+8.1%)**

Table 2: Improvements of various proposed models over baseline word model of the same size (10M parameters) on the WikiText-103 test set.

former shape (i.e., number of heads, head dimension, model dimension, and inner dimension in feedforward layer) as the baseline word model and transfer the bottom 10% of the decoder layers from the word model to initialize our character model.[5] Consistent with the findings of Trajanovski et al. (2021), we observe the improvements in ExactMatch@Overall and PartialMatch@Overall metrics to be highly correlated. Both "BERT-style word segment" and "transfer from word model" methods improve upon the baseline word model by at least 26% and 12% (shown in Table 2), in terms of ExactMatch and PartialMatch respectively. These methods also improve upon the baseline character model by at least 0.7% and 0.3% (not explicitly shown in Table 2), in terms of ExactMatch and PartialMatch respectively. Importantly, compared to the "BERT-style word segment" method that introduces 384K additional parameters, our "transfer from word model" method does not introduce any additional parameters. This demonstrates the advantage of "transfer from word models" in improving baseline character model (as compared to our other methods), while leaving no impact on memory. We also perform human evaluation of suggestions generated by various autocomplete models based on their naturalness and acceptability. Naturalness measures how natural the suggestion is with respect to the prompt while acceptability measures how likely the suggestion will be accepted by user (details in A.11). Human suggestions taken from WikiText-103 have a naturalness and user acceptability score of 88% and 100% as rated by annotators. We observe that the "transfer from word models" method generates most natural and user acceptable suggestions (69%, 94% resp.), which is better than the baseline character (62%, 93% resp.) second only to the human baseline (88%, 100% resp.).

Prompt and Suggestions
Prompt: The Olmec civilization developed in the lowlands of southeastern Mexico ... , the Indus Valley Civilization of south Asia **Ground truth**: , the civilization **Baseline**: , and the **BERT-style**: , the indus **Transfer from word models**: , the civilization
Prompt: Typhoon Lupit formed on November 18 from the monsoon trough to the west of the Marshall Islands . Early in its duration , it moved generally to **Ground truth**: the west or **Baseline**: the north of **BERT-style**: the west of **Transfer from word models**: the west of

Table 3: Sample suggestions of length 3 words generated by baseline and proposed character autocomplete models. See Appendix A.9 for more examples.

Qualitative Analysis. Tables 3 and 9 (Appendix A.9) show sample suggestions generated by the proposed baseline character autocomplete model as well as its proposed variants. Suggestions generated by the strongest method seem to have better match with the ground truth and factually (e.g., direction of typhoon) correct.[6]

7 Conclusion

In this work, we investigated the challenging task of building autocomplete models for answering broad prompts under memory-constrained settings. To this end, we introduced some novel methods that integrate word information into a character model with negligible impact on memory. Employing our methods, we demonstrated that character models can achieve a better accuracy-memory trade-off as compared to word models.

8 Limitations

The limitations of this work are as follows:

- **English.** Our work builds autocomplete models for English language only.
- **Accuracy-memory tradeoff only.** Our work primarily focuses on deploying models on lower-end edge platforms where memory, as opposed to latency, is the major bottleneck. Hence, our methods may not improve the accuracy-latency tradeoff, which is a focus for future work.
- **WikiText-103 dataset** Our work explores only WikiText-103 dataset for creating broad prompts. In the future, we will study

[5]The hyperparameter space for the transfer from word models method can be seen in Appendix A.4.

[6]We provide a qualitative analysis of the baseline and proposed character models in the Appendix A.10.

other datasets (e.g., 1 Billion Word Language Model benchmark (Chelba et al., 2013)) that explore the full range of low-frequency prompt patterns, which can arise in real-world situations.

- **Transformer-XL architecture** Our work studies only Transformer-XL architecture to build word based and character based autocomplete models. In the future, we will study other popular architectures (e.g., GPT-2 (Radford et al., 2018)) to see the generalizability of proposed techniques.

Acknowledgements

MAM acknowledges support from Canada Research Chairs (CRC), the Natural Sciences and Engineering Research Council of Canada (NSERC; RGPIN-2018-04267), Canadian Foundation for Innovation (CFI; 37771), and Digital Research Alliance of Canada.[7] Lakshmanan's research was supported in part by a grant from NSERC (Canada).

References

Rami Al-Rfou, Dokook Choe, Noah Constant, Mandy Guo, and Llion Jones. 2019. Character-level language modeling with deeper self-attention. In *AAAI*.

Alexei Baevski and Michael Auli. 2019. Adaptive input representations for neural language modeling. In *International Conference on Learning Representations*.

Ziv Bar-Yossef and Naama Kraus. 2011. Context-sensitive query auto-completion. In *Proceedings of the 20th International Conference on World Wide Web*, WWW '11, page 107–116. Association for Computing Machinery.

Jason Baumgartner, Savvas Zannettou, Brian Keegan, Megan Squire, and Jeremy Blackburn. 2020. The Pushshift Reddit Dataset. *CoRR*, abs/2001.08435.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

[7] https://alliancecan.ca

Fei Cai and Maarten de Rijke. 2016. *A Survey of Query Auto Completion in Information Retrieval*. Now Publishers Inc.

Han Cai, Chuang Gan, Ligeng Zhu, and Song Han. 2020. Tinytl: Reduce memory, not parameters for efficient on-device learning. In *Advances in Neural Information Processing Systems*, volume 33, pages 11285–11297.

Ciprian Chelba, Tomás Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, and Phillipp Koehn. 2013. One billion word benchmark for measuring progress in statistical language modeling. *CoRR*, abs/1312.3005.

Mia Xu Chen, Benjamin N. Lee, Gagan Bansal, Yuan Cao, Shuyuan Zhang, Justin Lu, Jackie Tsay, Yinan Wang, Andrew M. Dai, Zhifeng Chen, Timothy Sohn, and Yonghui Wu. 2019. Gmail Smart Compose: Real-Time Assisted Writing. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery And Data Mining*, KDD '19, page 2287–2295.

Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, and Eric P. Xing. 2023. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality.

Dokook Choe, Rami Al-Rfou, Mandy Guo, Heeyoung Lee, and Noah Constant. 2019. Bridging the Gap for Tokenizer-Free Language Models. *CoRR*, abs/1908.10322.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc Le, and Ruslan Salakhutdinov. 2019. Transformer-XL: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2978–2988, Florence, Italy. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Yao Dou, Maxwell Forbes, Rik Koncel-Kedziorski, Noah Smith, and Yejin Choi. 2022. Is GPT-3 text indistinguishable from human text? scarecrow: A framework for scrutinizing machine text. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7250–7274, Dublin, Ireland. Association for Computational Linguistics.

Simon Gog, Giulio Ermanno Pibiri, and Rossano Venturini. 2020. Efficient and effective query auto-

completion. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '20, page 2271–2280.

Mitchell A. Gordon, Kevin Duh, and Nicholas Andrews. 2020. Compressing BERT: Studying the Effects of Weight Pruning on Transfer Learning. *CoRR*, abs/2002.08307.

Song Han, Huizi Mao, and William J. Dally. 2016. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *ICLR*.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Comput.*, 9(8):1735–1780.

Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. 1993. Building a large annotated corpus of english: The penn treebank. *Comput. Linguist.*, 19(2):313–330.

Gaurav Menghani. 2021. Efficient deep learning: A survey on making deep learning models smaller, faster, and better. *CoRR*, abs/2106.08962.

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

OpenAI. 2023a. Gpt-4 technical report.

OpenAI. 2023b. Introducing chatgpt.

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

Dae Hoon Park and Rikio Chiba. 2017. A neural language model for query auto-completion. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '17, page 1189–1192. Association for Computing Machinery.

Baolin Peng, Chunyuan Li, Pengcheng He, Michel Galley, and Jianfeng Gao. 2023. Instruction tuning with gpt-4. *arXiv preprint arXiv:2304.03277*.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving Language Understanding by Generative Pre-Training. https://s3-us-west-2.amazonaws.com/openai-assets/research-covers/language-unsupervised/language_understanding_paper.pdf.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas. Association for Computational Linguistics.

Anna Rogers, Niranjan Balasubramanian, Leon Derczynski, Jesse Dodge, Alexander Koller, Sasha Luccioni, Maarten Sap, Roy Schwartz, Noah A. Smith, and Emma Strubell. 2023. Closed ai models make bad baselines.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *CoRR*, abs/1910.01108.

Sheng Shen, Zhen Dong, Jiayu Ye, Linjian Ma, Zhewei Yao, Amir Gholami, Michael W. Mahoney, and Kurt Keutzer. 2020. Q-BERT: hessian based ultra low precision quantization of BERT. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 8815–8821. AAAI Press.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca.

Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. 2020. Efficient transformers: A survey. *CoRR*, abs/2009.06732.

Yi Tay, Vinh Q. Tran, Sebastian Ruder, Jai Gupta, Hyung Won Chung, Dara Bahri, Zhen Qin, Simon Baumgartner, Cong Yu, and Donald Metzler. 2022. Charformer: Fast character transformers via gradient-based subword tokenization. In *International Conference on Learning Representations*.

Stojan Trajanovski, Chad Atalla, Kunho Kim, Vipul Agarwal, Milad Shokouhi, and Chris Quirk. 2021. When does text prediction benefit from additional context? an exploration of contextual signals for chat and email messages. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies: Industry Papers*, pages 1–9, Online. Association for Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems*, pages 6000–6010.

Hanrui Wang, Zhekai Zhang, and Song Han. 2021. Spatten: Efficient sparse attention architecture with cas-

cade token and head pruning. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 97–110. IEEE.

Sida Wang, Weiwei Guo, Huiji Gao, and Bo Long. 2020. Efficient neural query auto completion. In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management*, CIKM '20, page 2797–2804.

A Appendices

A.1 Reproducibility

We experiment with both Reddit and WikiText-103 datasets. WikiText-103 is a public dataset and widely adopted as a language modeling benchmark. WikiText-103 is downloaded from `tinyurl.com/yajy5wjm`. The Reddit dataset used in this work is a sample of publicly available *Pushshift Reddit dataset* (Baumgartner et al., 2020). The sample contains 4M train, 20K validation and 20K test posts. The key feature of the Reddit dataset is the significantly low percentage of unique out of vocabulary n-grams compared to WikiText-103, as shown in Table 1 and discussed in Section 4. For reproducibility, datasets and code used in this work is available at `tinyurl.com/bdd69r34` (anonymized) and will be made publicly available should paper be accepted.

A.2 Hyperparameter space for computing component-wise parameter breakdown

Table 7 displays the Transformer-XL hyperparameter space used to create 100 random architectures for computing component-wise parameter breakdown plot (Figure 1) for both word and character models. Rest of the hyperparameters come from the default configuration of Transformer-XL model.

A.3 Hyperparameter values for word and character models of different sizes

Table 5 displays the hyperparameter values for word models of different sizes used in the paper. Table 6 displays the hyperparameter values for character models of different sizes used in the paper.

A.4 Hyperparameter space for transfer from word models method

Table 7 displays the hyperparameter space for the proposed transfer from word models method.

A.5 Greedy vs. Beam search decoding

Figure 7 shows the pareto-curve for greedy and beam search. It is clear that smaller character models rival bigger word models regardless of the choice of decoding algorithm. Strikingly, we find greedy search to outperform beam search by a large margin. Two possible reasons are: (i) the noise injected by the adaptive softmax approximation of predicted probability distribution over vocabulary, and/or (ii) sensitivity of beam search to explore

Hyperparameter Name	Hyperparameter Values for Sampling
Number of hidden layers	{ 2, 4, 8, 12, 16, 24, 32 }
Number of attention heads	{ 2, 4, 8, 16, 32, 64 }
Dimension of attention head	{ 8, 16, 32, 64, 128 }
Dimension of input/output embedding	{ 256, 512, 1024, 2048 }
Inner dimension of feedforward layer	{ 256, 512, 1024, 2048 }
Dimension of model	{ 256, 512, 1024, 2048 }

Table 4: Hyperparameter space for computing component-wise parameter breakdown for both word and character models.

Hyperparameter name / Model size	$5M$	$10M$	$20M$	$30M$	$40M$	$50M$	$80M$
Number of hidden layers	3	4	6	12	14	16	16
Number of attention heads	4	4	8	8	8	8	32
Dimension of attention head	24	24	32	32	32	32	32
Dimension of input/output embedding	18	36	74	100	128	160	256
Inner dimension of feedforward layer	60	150	200	768	900	800	768
Dimension of model	18	36	74	100	128	160	256
Number of tokens to predict during training	192	192	192	192	192	192	192
Number of tokens cached from previous iterations during training	192	192	192	192	192	192	192
Learning rate	0.01	0.01	0.01	0.01	0.01	0.01	0.01
Number of iterations for learning rate warmup	1K	1K	1K	1K	1K	1K	1K
Maximum number of training steps	200K	200K	200K	200K	200K	200K	200K
Batch size	256	256	256	256	256	256	256
Number of tokens to predict during evaluation	192	192	192	192	192	192	192
Number of tokens cached from previous iterations during evaluation	192	192	192	192	192	192	192
Vocabulary size	267736	267736	267736	267736	267736	267736	267736

Table 5: Hyperparameter values for word models of different sizes.

spurious hypothesis when the user prompt patterns are low frequency.

A.6 Differences of Autocomplete from Conventional Language Modeling Task.

The autocomplete task is a well-defined problem with rich prior literature (see Section 2). Existing autocomplete research, including ours, is focused on building a conventional language model that computes the likelihood of a text sequence. The training procedure for our autocomplete task and that for conventional language modeling (CLM) task are generally similar. However, the goal of our autocomplete task is to generate suggestions with high precision (as captured by ExactMatch) while the main goal of CLM is to maximize the overall data likelihood (as captured by perplexity). Chen et al. (2019) show that perplexity and ExactMatch metrics are only weakly correlated as improvements in perplexity could be "mostly in places where the model is relatively low in likelihood score". As shown in Figure 8, autocomplete models with poorer perplexity scores (e.g., character model of size 20M) can enjoy better ExactMatch scores compared to models with better perplexity scores (e.g., word model of size 20M). We also perform a theoretical analysis to show how perplexity scores can change drastically for the same ExactMatch score (details in Appendix A.7). Thus, building a good language model is not enough to solve the autocomplete task. Another major conceptual difference between CLM and autocomplete tasks is that the former focuses mainly on generating long horizon (typically 128-512 tokens) continuation while the latter focuses on generating short horizon (typically 3-5 tokens) continuation.

A.7 Theoretical analysis on differences in perplexity and Exact Match metrics

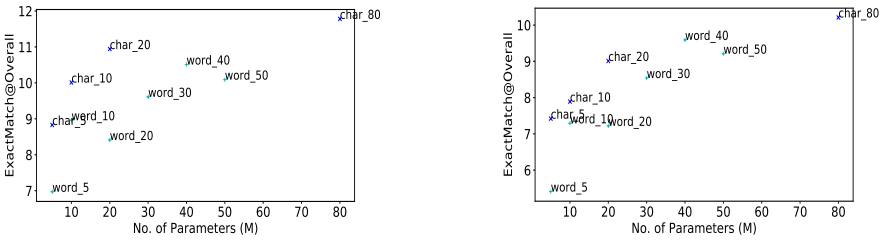
We will conduct a theoretical study to show the differences in the information captured by perplexity and Exact Match metric. Specifically, we show that the exact match score can be perfect whereas perplexity score can either be perfect or worsen by a large margin (**Claim 1**). Conversely, we also show that the exact match score can be the worst (i.e., zero) whereas the perplexity score can be poor or better by a large margin (**Claim 2**). Without loss of generality, we assume the vocabulary size $\mathcal{V}$ to be 2. Let A, B be the two tokens corresponding to the first and second index in the vocabulary respectively. Consider a single token prediction ($\hat{x}_j$) and

Hyperparameter name / Model size	$5M$	$10M$	$20M$	$80M$
Number of hidden layers	12	12	12	16
Number of attention heads	8	8	8	8
Dimension of attention head	32	32	64	64
Dimension of input/output embedding	278	512	550	750
Inner dimension of feedforward layer	128	165	250	2048
Dimension of model	278	512	550	750
Number of tokens to predict during training	512	512	512	512
Number of tokens cached from previous iterations during training	512	512	512	512
Learning rate	0.001	0.001	0.001	0.001
Number of iterations for learning rate warmup	4K	4K	4K	4K
Maximum number of training steps	400K	400K	400K	400K
Batch size	128	128	128	128
Number of tokens to predict during evaluation	512	512	512	512
Number of tokens cached from previous iterations during evaluation	2K	2K	2K	2K
Vocabulary size	128	128	128	128

Table 6: Hyperparameter values for character models of different sizes.

Hyperparameter Name	Hyperparameter Values
Number of hidden layers	{ 4, 8, 12, 16, 20, 24 }
Percentage of bottom most layers to transfer	{ 10%, 20%, 30%, 40%, 50% }

Table 7: Hyperparameter space for transfer from word models method.



(a) Greedy search

(b) Beam search

Figure 7: Greedy search vs. Beam search on WikiText-103 test set. Beam size and prompt context percentage is set as 5 and 20% respectively.

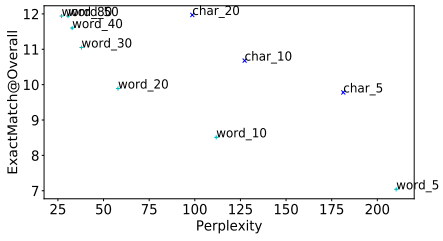


Figure 8: Perplexity vs. ExactMatch. For comparison, perplexity output by character models (also known as bits per byte) is converted to perplexity per word using the formula proposed in Choe et al. (2019).

let the ground truth token be B, that is, $\hat{x}_j = [0, 1]$. Table 8 shows the differences in perplexity score and Exact Match score as a function of $\hat{x}_j$, as it varies slightly. The first six rows in the table validate **Claim 1**, where exact match score is 1 but the perplexity ranges $-9.9\text{e}{-10}$ to 0.67. The rest of the rows validate **Claim 2**, where the exact match score is 0 but the perplexity score ranges from 0.69 to 20.72.

A.8 Accuracy-Memory Pareto-Curve on Unseen Datasets

We study the accuracy-memory pareto curve of autocomplete models trained on WikiText-103 and evaluate on the test set of two unseen datasets: LAnguage Modeling Broadened to Account for Discourse Aspects (Paperno et al., 2016) (LAMBADA, mostly focused prompts) and Penn Treebank (Marcus et al., 1993) (PTB, mostly broad prompts). From Figure 9, we observe that the

Ground truth (x_j)	Prediction ($\hat{x}_j$)	Exact Match	Perplexity
$[0, 1]$	$[0, 1]$	1	$-9.9\text{e}{-10}$
$[0, 1]$	$[0.1, 0.9]$	1	0.11
$[0, 1]$	$[0.2, 0.8]$	1	0.22
$[0, 1]$	$[0.3, 0.7]$	1	0.36
$[0, 1]$	$[0.4, 0.6]$	1	0.51
$[0, 1]$	$[0.49, 0.51]$	1	0.67
$[0, 1]$	$[0.5, 0.5]$	0	0.69
$[0, 1]$	$[0.51, 0.49]$	0	0.71
$[0, 1]$	$[0.6, 0.4]$	0	0.92
$[0, 1]$	$[0.7, 0.3]$	0	1.2
$[0, 1]$	$[0.8, 0.2]$	0	1.61
$[0, 1]$	$[0.9, 0.1]$	0	2.3
$[0, 1]$	$[1.0, 0]$	0	20.72

Table 8: Differences in perplexity and Exact Match as function of small changes in $\hat{x}_j$ when the ground truth is $[0, 1]$.

trend where smaller character models rival larger word models that holds true for answering broad prompts (PTB) but not clearly for answering focused prompts (LAMBADA). It is striking that the trend holds true for broad prompts even when the examples are unseen during the training of the autocomplete model.

A.9 Qualitative examples of suggestions from autocomplete models

Table 9 displays sample suggestions generated by vanilla and proposed character autocomplete models, grouped by the type of artifact in the generation.

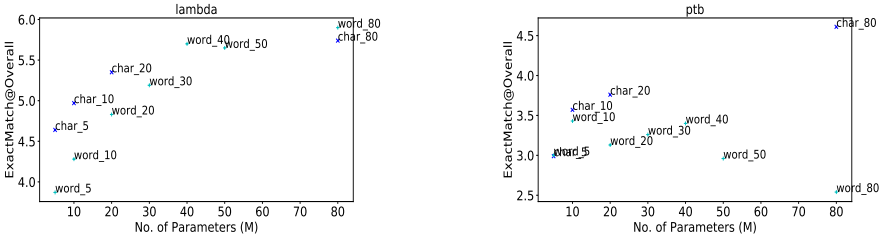
A.10 Qualitative analysis of vanilla and proposed character models

We manually inspect the suggestions generated by vanilla and proposed character models[8]. Table 10 displays the percentage of different artifacts: *plausible* (plausible suggestion that does not have exact match with the ground truth), *semantic error* (e.g., new n-gram, incorrect n-gram usage), *repetition* (e.g., n-gram with repetitions), and *grammatical error*. Compared to baseline and BERT-style word segment model, character model with decoder layer transfer from word model results in less undesirable artifacts overall.

A.11 Human annotation of suggestions

We conduct human annotation of suggestions outputted by various autocomplete models based on *naturalness* (how natural the suggestion is with respect to the prompt?) and *acceptability* (whether the suggestion will be accepted by user or not?). Some aspects of natural suggestion are borrowed from Dou et al. (2022). The annotation guideline for naturalness and acceptability can be seen in Table 11 and Table 12 respectively. We ask 8 annotators to rate 10 suggestions each.

[8] Sample suggestions from different autocomplete models can be seen in Appendix A.9.



(a) LAMBADA (Paperno et al., 2016)

(b) PTB (Marcus et al., 1993)

Figure 9: Accuracy-Memory Pareto Curve for Autocomplete models trained on WikiText-103 and evaluated on test set of two unseen datasets: LAMBADA and PTB.

Artifact type	**Prompt and Suggestions**
Plausible	**Prompt**: In 2006 Boulter starred in the play Citizenship written by Mark Ravenhill . The play was part of a series which featured different playwrights , titled Burn / Chatroom / Citizenship . In a 2006 **Ground truth**: interview , fellow **Baseline**: interview , ravenhill **BERT-style**: interview with the **Transfer from word models**: interview with the
Plausible	**Prompt**: In December 759 , he briefly stayed in Tonggu (modern Gansu) . He departed on December 24 for Chengdu (Sichuan province) , where he was hosted by local Prefect and **Ground truth**: fellow poet Pei **Baseline**: servant and served **BERT-style**: chief executive officer **Transfer from word models**: commissioned as a
Semantic error	**Prompt**: In his lifetime and immediately following his death , Du Fu was not greatly appreciated . In part this can be attributed to his stylistic and formal innovations, some of which are still "considered extremely daring and bizarre by Chinese critics ." There are few contemporary references to him — only eleven poems from six writers — and these describe him in terms of affection, but not as a **Ground truth**: paragon of poetic **Baseline**: reference to his **BERT-style**: poem . the **Transfer from word models**: consequence of his
Semantic error	**Prompt**: Other translators have placed much greater weight on trying to convey a sense of the poetic forms used by Du Fu . Vikram Seth in Three Chinese Poets uses English @-@ style rhyme schemes , whereas Keith Holyoak in Facing the Moon approximates the Chinese rhyme scheme ; both use end @-@ stopped lines and preserve some degree of parallelism . In The Selected Poems of Du Fu , Burton Watson follows the parallelisms quite strictly , persuading the western reader to adapt to the poems rather than **Ground truth**: vice versa . **Baseline**: to the poems **BERT-style**: adapt the poems **Transfer from word models**: the parallelisms of
Repetition	**Prompt**: Although initially he was little @-@ known to other writers , his works came to be hugely influential in both **Ground truth**: Chinese and Japanese **Baseline**: the writers and **BERT-style**: writers and writers **Transfer from word models**: the ancient and
Repetition	**Prompt**: In the 20th century , he was the favourite poet of Kenneth **Ground truth**: Rexroth , who **Baseline**: kenneth kenneth kenneth **BERT-style**: county . the **Transfer from word models**: kenneth kenneth kenneth
Grammatical error	**Prompt**: Hung summarises his life by concluding that , **Ground truth**: " He appeared **Baseline**: according to ksummarises **BERT-style**: in the same **Transfer from word models**: as a result

Table 9: Sample suggestions of length 3 words generated by vanilla and proposed character autocomplete models, grouped by the type of artifact in the generation.

Artifact type	Baseline	BERT-style w. seg.	Transfer from word models
Plausible (↑)	40	40	**42**
Semantic Error (↓)	7	**6**	7
Repetition (↓)	7	7	**5**
Gram. Error (↓)	3	3	**2**

Table 10: Percentage of different artifacts in the generated suggestion from vanilla and proposed character models, by manual inspection of 100 WikiText-103 examples. ↑ indicates higher the better, ↓ indicates lower the better.

Autocomplete is a task where the user inputs a text, which is conditioned by the model to generate 'natural' continuation (or suggestion). The goal of this annotation effort is to rate the quality of suggestions generated by various autocomplete models based on the 'natural'ness. Each suggestion will be at most three words. Keep in mind that there could be more than one 'natural' suggestion for a text.

Some aspects of suggestion (but don't restrict only to these) that makes a suggestion NOT natural can be: grammatical error (missing words, extra words, incorrect or out of order words), redundancy (extra unnecessary information, word repetition), off-prompt (suggestion is unrelated to the text), self-contradiction (suggestion contradicts the text), incoherence (grammatical, not redundant, on prompt, not contradictory but still CONFUSING), factual or commonsense errors (violates our basic understanding of the world) and so on. Assume a broad definition of 'natural'ness and use your best judgement to rate.

You will be asked to annotate TEN texts. For each text, you will see a suggestion and you will rate by picking exactly one of the two choices:
(i) natural - Select this option if suggestion is natural with respect to the text
(ii) NOT natural - Select this option if suggestion is NOT natural with respect to the text

Table 11: Annotation guideline for human annotators to rate the quality of suggestions generated by autocomplete models and humans based on naturalness.

Autocomplete is a task where a user inputs a text (prompt), which is conditioned by the model to generate 'natural' continuation (or suggestion). For example, the user can give the prompt "Filmmaker George Lucas used Tikal as a", and the system may give a suggestion such as "filming location". An autocomplete system is successful if it can reduce the keystrokes a user would need to make, improving user productivity. The goal of this annotation task is to decide if (i) a suggestion generated by an autocomplete model will be accepted by a user (to reduce the keystrokes) or (ii) not. Each suggestion will be at most three words.

You can accept the suggestion if it is useful. A suggestion can be useful for one or more reasons (but don't restrict only to these): (i) the suggestion seems completely relevant to the prompt; (ii) the suggestion can be minimally edited for it to be useful. Note that reasons for acceptability are generally subjective. Hence, please assume a broad definition of "usefulness" and employ your best judgment to rate.

You will be asked to annotate 10 texts. For each text, you will see a suggestion and you will rate by picking exactly one of the two choices:
(i) yes - Select this option if you will accept the suggestion
(ii) no - Select this option if you will not accept the suggestion

The following is an example:
Filmmaker George Lucas used Tikal as a
Suggestion: filming location
Rating choices:
(i) yes - Select this option if you will accept the suggestion
(ii) no - Select this option if you will not accept the suggestion
Rating [type 'yes' or 'no' here in this line]: yes

Table 12: Annotation guideline for human annotators to rate the quality of suggestions generated by autocomplete models and humans based on acceptability.