Investigation on the Effect of Grammar on the Arrow of Time Using Large Language Models

Francesco La Rosa

May 31, 2025

Abstract

This report investigates whether an "Arrow of Time" exists in language modeling when models are limited to grammatical information alone. In standard large language models (LLMs) trained on natural text, prior work has observed a consistent asymmetry: forward-trained models (predicting next tokens) outperform backward-trained models (predicting previous tokens) on average, indicating an intrinsic "Arrow of Time" in how models learn language [1]. We remove lexical semantics by converting text to sequences of part-of-speech (POS) tags, isolating syntactic structure, and train Transformer LLMs on these POS sequences in both forward and backward directions. Our experiments spanning multiple model scales (from a 3-layer Nano model to a 12-layer GPT-1 sized model) and languages (English, Finnish, Italian, German) find no detectable performance difference between forward vs. backward models when only grammatical sequences are learned. All models achieve nearly identical cross-entropy losses in both directions, suggesting no inherent time-direction bias in purely syntactic data. This contrasts with the clear forward-direction advantage seen on natural text [1]. Notably, a slight negative arrow-of-time effect reported in prior English text experiments (backward model outperforming forward) [1] did not appear in our POS-only models, and remains unexplained. We report all quantitative results in terms of cross-entropy, and discuss implications that the Arrow of Time in language modeling stems from semantic or other non-syntactic factors. Future tests are outlined to pinpoint the source of the asymmetry observed in full-text models.

1 Introduction

Large Language Models are typically trained to predict the next token in a sequence of text (forward direction), implicitly assuming the natural temporal order of language is optimal for learning. An intriguing question is whether this time directionality confers any fundamental learning advantage, or if models trained in the reverse direction (predicting previous tokens) would perform just as well. Recent research by Papadopoulos et al. (2024) [1] showed that for full natural language data, forward (FW) models consistently achieve lower perplexity than backward (BW) models of identical architecture, revealing an "Arrow of Time" (AoT) effect in language modeling. This effect was found across many languages and model types, suggesting a universal directional bias ingrained in language or learned representations. However, the causes of this asymmetry remain unclear: is it due to semantic content, syntactic structure, or other aspects of language data?

In this work, we examine the Arrow of Time from a novel angle by eliminating lexical semantics and isolating grammar. We hypothesize that if the AoT effect is rooted in the way syntactic dependencies unfold in forward text (for example, English is primarily subject-verb-object order), a model trained on only grammatical sequences might still show a forward vs. backward difference. Conversely, if semantics or pragmatics drive the asymmetry (e.g., the progression of meaning or real-world causality in forward text), then removing meaning should eliminate any performance gap. To test this, we convert large text corpora into sequences of POS tags, preserving only the structural grammatical information and discarding specific word identities. We then train comparable LLMs on these sequences in both forward and reversed order and measure their modeling performance.

Our study makes three main contributions: (1) We present a methodology for training transformer LLMs on POS-tag sequences using two different tokenization schemes to represent grammar with and without token variability. (2) We empirically evaluate forward vs. backward training on purely syntactic data for English and several typologically diverse languages, finding that no significant AoT effect remains, in stark contrast to models trained on natural text. (3) We analyze the implications of this result, namely that the previously observed AoT in full text must derive from elements absent in POS-only data

(likely lexical semantics or long-range discourse coherence). We also address a puzzling observation from prior work: a slight negative AoT in English (i.e. backward model outperforming forward in one setting) [1]. Our findings indicate this anomaly does not arise from grammatical structure, and we discuss why further investigation with semantic information is needed to explain it.

The remainder of this report is organized as follows. In Section 3 (Related Work), we situate our research in context, reviewing prior studies of forward vs. backward language modeling and the known AoT phenomenon. Section 4 (Design & Methodology) details our approach to isolating grammar: the datasets, POS tagging process, tokenization schemes, and experiment design. Section 5 (Implementation) describes the model architectures, training setup, and how forward/backward training was realized in practice. In Section 6 (Evaluation), we present quantitative results comparing cross-entropies of forward and backward models across different conditions. Section 7 (Discussion) interprets the results, considering why no AoT was detected with grammatical data and what this implies about the original effect. Section 8 (Conclusion) summarizes our key findings, chiefly that restricting models to grammar removes the Arrow of Time and outlines future work to pinpoint the source of directional bias in language models.

2 Related Work

The question of forward vs. backward prediction in language goes back at least to Shannon (1951) [2]. In his seminal work on the entropy of English text, Shannon noted that human participants found backward prediction (guessing the preceding letter/word from context) subjectively harder, yet achieved only slightly worse accuracy than forward prediction [2]. This early observation hinted that, in theory, the information content of text is time-symmetric the entropy rate forward or backward should be equal. However, practical predictors might not realize this equality, leading to measurable differences.

In modern NLP, sequence-to-sequence models have occasionally leveraged reversed sequences for improved performance. Sutskever et al. (2014) [3] famously found that reversing the source sentences in an encoder-decoder LSTM greatly improved English-French machine translation results. This was a task-specific trick (helping the decoder attend to the beginning of sentences), but it demonstrated that neural networks process sequence order differently and that backward context can be learned effectively. Other works have explored using backward language models in tandem with forward models. For example, bi-directional training approaches like ULMFIT and ELMo combine forward and backward LMs to capture more information for downstream tasks. Mou et al. (2016) [4] specifically trained separate forward and backward RNN language models and used them for constrained sentence generation. They treated backward models as a tool for text generation under constraints, rather than comparing their intrinsic performance to forward models, but their work shows that backward LMs can be built and used alongside forward LMs.

The most directly relevant study is the recent work by Papadopoulos et al. (2024) [1], who conducted a comprehensive analysis of what they termed the "Arrow of Time" in autoregressive language models. They trained GPT Transformer models as well as LSTM and GRU models in both directions on a variety of natural language datasets. Their key finding was that forward-trained models consistently achieve lower validation loss (perplexity) than backward-trained models when all else is equal. This forward advantage was small but persistent, typically on the order of a few percent in perplexity. For instance, on an English corpus, a GPT-2-medium model achieved a validation cross-entropy difference of about $\sim$0.76% increase for the backward model. They observed this gap widened for larger context lengths and model sizes, and found similar forward advantages in multiple languages including French, Greek, Indonesian, etc., suggesting the AoT effect is a universal property of natural language data and not an artifact of a particular tokenization or model type. Notably, one irregular result was reported: with a very small English model (and/or very limited context), the backward model slightly outperformed the forward model (a "negative" AoT). This unexpected case "convincingly disappeared" when model capacity or context increased, and the authors did not find a clear explanation for it. Our work was partly motivated by this anomaly we wondered if some fundamental aspect of English grammar might favor backward modeling at small scales.

In summary, prior research establishes that forward vs. backward training differences do exist empirically in language modeling, but their origin is not fully understood. Some hypothesize it relates to causal structure in language or how information unfolds (e.g., punctuation or new topic introduction might be easier to model in forward direction). Others point out that any true asymmetry must come from how models learn or data properties, since an ideal model on infinite data should model both directions equally [2]. By stripping away semantics and using POS sequences, our study connects to this literature by testing a new hypothesis: whether syntactic structure alone can produce an Arrow of Time.

To our knowledge, no prior work has isolated grammar in this manner for comparing forward/backward LM performance. Our findings will refine the understanding of what drives the AoT effect observed by Papadopoulos et al. [1], and we will relate our results back to those in the Discussion.

3 Design & Methodology

Our experimental design focuses on training language models under highly controlled conditions to isolate grammatical structure. We outline the key components of our methodology below: datasets and POS tagging, POS tokenization schemes, and the model training configurations for forward vs. backward experiments.

3.1 Datasets and POS Tagging

We use two large text corpora as the source of linguistic data: OpenWebText (OWT) and CC100. OpenWebText is an open replica of OpenAI's WebText, containing web pages from Reddit links (approx. 10 GB of English text). CC100 is a multilingual dataset from Common Crawl; we use specifically the English subset of CC100 (CC100-En) as well as subsets of CC100 for other languages (Finnish, Italian, German) for cross-lingual comparison. In each case, we aim to sample on the order of 1-2 GB of raw text for training to have a sizeable corpus but still manageable for POS tagging.

Before training, all text is converted into part-of-speech tag sequences. We use the spaCy NLP toolkit to tokenize and tag the text. For English, we employ spaCy's `en_core_web_sm` model to get the coarse-grained POS tags for each token (e.g., NN for noun, VBZ for verb 3rd person singular, JJ for adjective, etc.). The tagging process keeps punctuation and special tokens (each assigned a tag as well), and discards actual word identities effectively reducing each word to an abstract grammatical category. The output of this stage is a long sequence of POS tags representing the input text's syntactic structure. For example, a sentence like "The quick brown fox jumps over the lazy dog." would become something like "DT JJ JJ NN VBZ IN DT JJ NN". We performed analogous POS tagging for the other languages using the appropriate spaCy models (e.g., `it_core_news_sm` for Italian, `de_core_news_sm` for German, etc.), ensuring the tag set is language-appropriate. This procedure isolates syntax: any two different nouns or verbs will look identical in this representation (both just "NN" or "VB", etc.), so only grammatical patterns remain for the model to learn.

To create train/validation splits, we concatenate the tag sequences and then divide them. We reserved 5% of the tokens for validation and the rest for training. The data is stored in a numeric binary format for efficient loading: each POS tag is mapped to an integer ID (as described next), and the sequences are written to a `.bin` file (with an accompanying `meta.pkl` storing metadata like the vocabulary of tags). This allows the training code to memory-map and stream sequences without huge memory overhead.

3.2 POS Tokenization Schemes

One important design choice is how to map POS tags (which are discrete symbols like "NN") into token IDs for the model. We devised two tokenization schemes to explore how token vocabulary structure might impact learning of syntax:

- **(A) Unique ID per POS Tag - "ID" scheme:** Each distinct POS tag is assigned a unique integer ID, forming a small fixed vocabulary equal to the number of tag types. For example, if the tagset has 40 tags, we create 40 IDs (plus a special ID for a beginning-of-sequence token). Every occurrence of "NN" in the sequence is encoded with the same ID (say 5), every "VBZ" with another ID (say 17), etc. This results in a deterministic, lossless compression of the tag sequence the model sees an exact sequence of POS categories. In this scheme, the vocabulary size for English is roughly 50 (since Penn Treebank tagset has $\sim$45 tags plus punctuation tags), and similar magnitudes for other languages. The model trained on these IDs will effectively learn transitions and patterns among POS tags in the text.

- **(B) Random ID ranges per POS Tag - "Range" scheme:** To further abstract away any consistent token identity, we assign each POS tag a range of possible IDs, and randomly choose a different ID within that range each time the tag appears. The ranges are sized proportional to the number of unique word types that had that tag in the corpus. For example, if the tag "NN" (common noun) had 10,000 unique words in the corpus, its ID range might have 10,000 possible

IDs (say 1000-10999); each occurrence of an NN will be assigned one of those IDs at random. Another tag like "VBZ" with fewer unique words might get a smaller range (e.g., 500-599). Proper nouns (NNP) are extremely diverse, so we cap the NNP range to 2000 IDs maximum to avoid the vocabulary exploding. All these ranges are non-overlapping segments in the overall token ID space. The combined vocabulary in this scheme is much larger on the order of tens of thousands of IDs, but crucially, the model cannot rely on any token ID repeating to recognize a context. It only knows two tokens are the same POS if their numeric IDs fall in the same range (i.e., by the range interval, not by exact equality). This injects maximal uncertainty: the model sees no stable word identity at all, only a constantly varying "disguise" of each POS category. This scheme tests if the model can still learn syntactic structures when it must generalize purely by category distributions rather than memorizing specific token sequences.

These two schemes represent extremes: Scheme A gives the model a clean, low-entropy symbolic sequence of POS tags, whereas Scheme B introduces high token-level entropy while preserving category information in an implicit way (the ranges). By comparing them, we examine whether a more complex tokenization (Scheme B) makes learning grammar harder and whether it interacts differently with forward vs. backward training. In theory, if the arrow-of-time effect is robust to tokenization, both schemes should show or not show it similarly [1].

3.3 Model Training Setup

We base our models on the GPT architecture (Transformer decoder) using a modified implementation derived from the open-source nanoGPT codebase [5]. All models are decoder-only Transformers with causal self-attention and feed-forward layers, trained from scratch (no pretraining on raw text) on our POS datasets. Table 1 summarizes the model variants we use.

Table 1: Model configurations used in experiments.

Model Variant	Layers	Hidden Dim (n_embd)	Heads	Parameters (approx)
Nano	3	48	3	∼0.1 M
Mini	6	192	6	∼6 M
Small	10	380	10	∼29 M
GPT-1	12	768	12	∼110 M

All models use a feed-forward dimension 4x the embedding size (with GELU activations) and no dropout (dropout=0.0) during training. LayerNorms are applied in a standard GPT configuration. We include the GPT-1 scale model to ensure we have enough capacity to fully model the POS sequences, and to see if any subtle AoT effects emerge at a larger scale.

Training Hyperparameters: We train each model for 1 epoch over the training data (one full pass through ∼1-2GB of tokens). Due to the data size, this corresponds to a fixed number of iterations (we use a `block_size` of 64 tokens per training example and a stride of 32). With these settings, one epoch is roughly 90k iterations for our dataset. We use batch size 128 and the AdamW optimizer ($\beta_1 = 0.9, \beta_2 = 0.99$) with a learning rate of 1e-4. A short linear warmup brings the learning rate to 1e-4, then it stays constant for most of training, and in the final 10% of iterations we apply a linear cooldown to a min LR of 0. This schedule helps stabilize end-of-training convergence as outlined by Hägele et al.[6]. Importantly, we train both a forward and a backward model for each configuration: the forward model sees sequences in natural order, while the backward model sees reversed sequences.

Forward vs. Backward Data Preparation: For the forward models, we simply prepend a special begin-of-sequence token (BOS) at the start of each 64-token chunk. The model is trained to predict the next POS tag at each step, including predicting the first real token after BOS. For backward models, we append the BOS token at the end of each 64-token chunk (as an end-of-sequence marker in original text order), then reverse the chunk before feeding it to the model. Thus, the backward model receives inputs that run from the original end-of-sequence toward the beginning. It is trained to predict "previous" tokens in the original sequence order, which, from the model's perspective, are the next tokens in the reversed input. Both forward and backward models use the same vocabulary and tokenization scheme, just with the order of tokens inverted for the backward case. We implemented this conveniently via a `backwards` flag in our `PyTorch Dataset`: if `backwards=True`, the dataset will take a data window, append the BOS token, reverse it, and then provide training examples such that the model learns to predict the sequence in reverse order. This approach ensures that aside from sequence order, all

other factors (data, model, optimization) are identical between a forward run and a backward run. At evaluation time, forward models calculate the usual left-to-right cross-entropy on the validation set, while backward models calculate cross-entropy right-to-left (which equivalently measures how well they model the data). All performance comparisons are based on cross-entropy (average negative log-likelihood) on the validation POS sequences, this is our primary metric. Using this methodology, we train a grid of experiments covering: each tokenization scheme (Unique-ID vs Range-ID), each model size (Nano, Mini, Small, GPT-1), and multiple languages (English as main, plus selected runs in Italian, Finnish, German for the Nano models). In total, over 20 model runs were conducted (each with a forward and backward counterpart). Training was tracked with Weights & Biases (`wandb`) for logging loss curves and final metrics.

4 Implementation

Our implementation builds upon the lightweight nanoGPT training code [5], with modifications to handle POS-tag inputs and backward training. Key aspects of the implementation include:

Data Pipeline: We wrote custom data loader classes to generate training examples from the POS-tagged corpora. The POS sequences for each dataset (OWT, CC100, etc.) are preprocessed and saved as `train.bin` and `val.bin` files of 16-bit integers. A `POSDataset` class (in `dataloader.py`) memory-maps these files and on indexing returns a pair of input and target tensors for a sequence chunk. If `backwards=False`, it inserts the BOS token ID at the beginning of the chunk and uses that as input (with the last token as target dropped off). If `backwards=True`, it appends BOS to the chunk, then reverses it, so that the model will be predicting the sequence in reverse. In both cases, BOS is represented by a token ID equal to the vocabulary size (since we assign IDs 0..V-1 to actual POS tokens, we use $ID = V$ as the BOS token). This strategy allowed us to use one unified training loop for forward and backward models, simply toggling the flag.

Vocabulary and Tokenization: During preprocessing, we build the tokenizer mapping depending on the scheme:

- For the Unique-ID scheme, we create a dictionary mapping each POS tag to a unique integer. This is straightforward e.g., {"NN":5, "VBZ":6, ...} and we store this mapping in `meta.pkl`. The vocab size V in this case equals the number of tags (e.g., $V \approx 45$ for English).
- For the Range scheme, we construct the dictionary of ranges as described in Section 4.2. The implementation computes how many unique words corresponded to each POS tag in the corpus, then allocates a range of that size (with scaling and capping for "NNP"). For example, it might assign "NN": range(0-8000), "VBZ": range(8000-8200), etc., and "NNP": range(...-+2000). The ranges are chosen to ensure the total vocab fits in a 16-bit integer ($\leq 65,535$). When converting the POS sequence to tokens, each tag instance is mapped by randomly picking an ID in that tag's range.

Model Architecture: We used a GPT model (`model.py`) configured by the parameters in each experiment's config file. This model uses standard components defined by Karpathy [5]: token embedding layer of size `n_embd`, a stack of `n_layer` Transformer decoder blocks (each with multi-head self-attention and a feed-forward network of size $4 \times$ `n_embd`), and a final linear layer back to vocab logits.

Logging and Evaluation: We integrated `wandb` logging to record training and validation loss periodically (every $N = 10$ or $N = 50$ iterations for training loss, and every `eval_interval`=2000 iterations we evaluate on the validation set). For each run (forward/backward pair), we gave descriptive names (e.g., "cc100_nano_forward" vs "cc100_nano_backwards") to easily compare them in the dashboard. The training script saves model checkpoints in an output directory per run, but for analysis we primarily rely on the logged losses. After training, we compute the final validation cross-entropy for each model as the main result. In some cases, we also ran an auxiliary script `predict_loss.py` to estimate the expected loss of the range model given the id model (see Appendix for more information).

The implementation was executed on NVIDIA GPUs. Each GPT-1 scale run (one epoch on $\sim$ 2GB of data with context 64) took on the order of a few hours to complete. Smaller models ran much faster (minutes to an hour). We ensured that each forward/backward pair was run under identical conditions (same random initialization, same data ordering) for a fair comparison.

5 Evaluation

We evaluated all models on the held-out validation set of POS sequences, comparing the forward and backward model losses to detect any Arrow of Time effect. The primary metric is validation cross-entropy (negative log-likelihood). We present our results organized by tokenization scheme, model size, and language.

5.1 English

Unique ID vs Random Range: We start with English (our largest experiments). Figure 1 shows the validation loss curves over training for English forward vs. backward models under the unique-ID scheme. The forward and backward loss trajectories are virtually indistinguishable, they overlap throughout training, and by the end of training their losses are the same within a tiny fraction. For instance, the final relative difference for the validation cross-entropy in GPT-1 size ID English model is only 0.03% (see Table 2 for exact values). For smaller models (Small, Mini, Nano), the differences we observed were on the order of 1e-3 to 1e-2 in cross-entropy, sometimes favoring forward, sometimes backward by that negligible amount. For example, the Nano size ID English model, Figure 2, in one run ended with val losses 2.19184 (FW) vs 2.18779 (BW), a difference of 0.18% relative. Although the Nano model showed a slightly higher relative difference than GPT-1, follow-up experiments confirmed this was likely just noise since the difference disappears with different seeds and datasets.

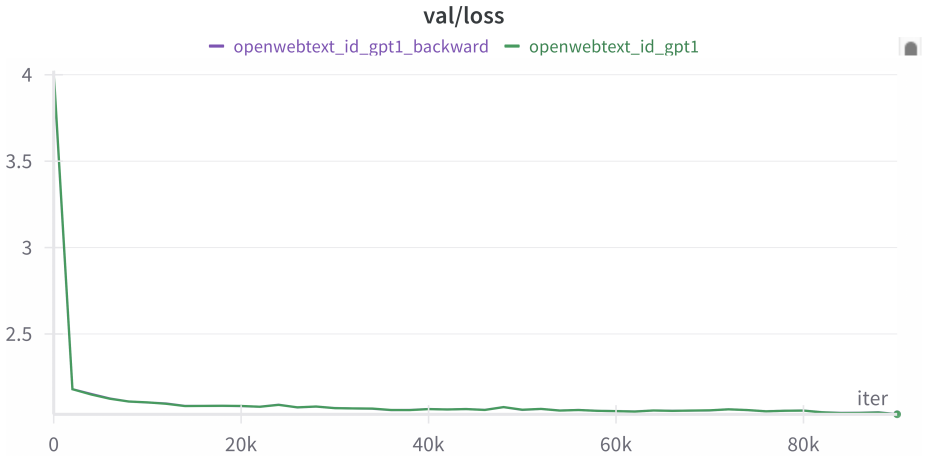

Figure 1: Validation loss for GPT-1 English ID Model.(Lines are overlapping due to no Arrow-of-Time)

We repeated the above for the random-range tokenization scheme. One might expect this higher-entropy input (65k vocabulary of random IDs) to be generally harder to model than the unique-ID case, and indeed the losses are higher in absolute terms due to the increased uncertainty. However, the forward vs. backward comparison remained effectively identical. Figure 3 plots the forward vs backward validation loss for the range-based English models. All model sizes again show overlapping performance. As a concrete example, the Small size Range English model had, a 0.01% difference (see Table 2 for exact values), effectively zero. Table 2 summarizes all the English results: none of the forward-backward loss deltas exceeded 0.2% (and most were $< 0.05\%$). We also verified the stability of this finding by running a couple of Nano models with different random seeds for the tokenization randomness; in those, one run might get a backward loss slightly lower than forward by -0.01, another run might reverse that, confirming that these tiny gaps (in the nano model) are not consistent and likely just random fluctuations. In short, when modeling English syntax alone, we find no evidence of an inherent Arrow of Time, the models learn equally well in either direction.

5.2 Other Languages

To test whether this conclusion generalizes, we conducted additional experiments on a few typologically diverse languages: Finnish, Italian, and German. These languages differ from English in word order flexibility and morphology. For example, Finnish is highly inflectional and allows relatively free word

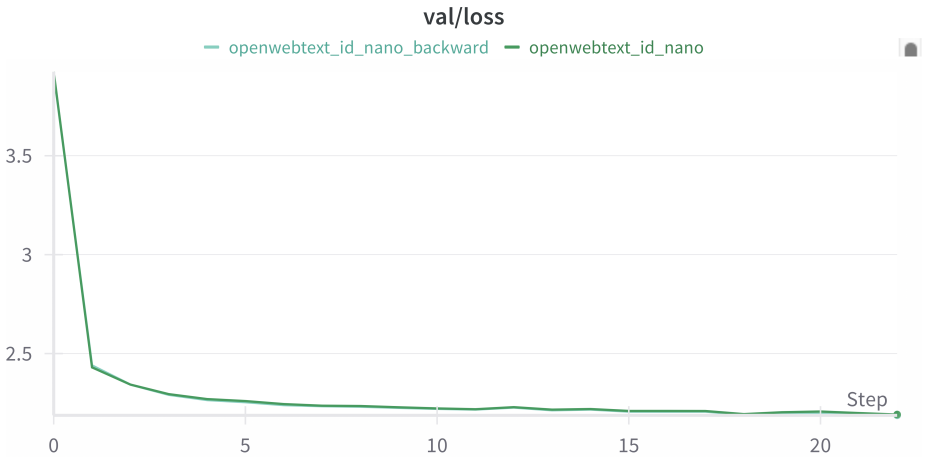

Figure 2: Validation loss for Nano English ID Model.(Lines are overlapping due to no Arrow-of-Time)

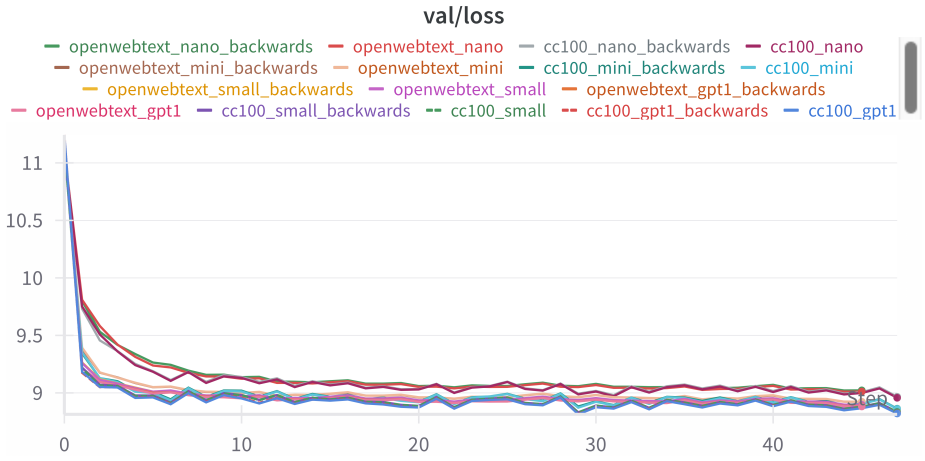

Figure 3: Validation loss for all sizes English Range models.(Lines are overlapping due to no Arrow-of-Time)

Table 2: English POS-only models (OpenWebText): validation cross-entropy. Δ is the percentage change (BW − FW)/FW × 100.

Model	Forward ↓	Backward ↓	Δ (%)
Nano-ID	2.2146	2.2147	+0.00
Nano-Range	9.0113	9.0220	+0.12
Mini-ID	2.1226	2.1251	+0.12
Mini-Range	8.9315	8.9338	+0.03
Small-ID	2.0446	2.0451	+0.02
Small-Range	8.8946	8.8940	–0.01
GPT-1-ID	2.0350	2.0357	+0.03
GPT-1-Range	8.8858	8.8885	+0.03

Model	OpenWebText		CC100-en		Δ CC100–OWT	
	FW ↓	BW ↓	FW ↓	BW ↓	FW	BW
Nano	9.0113	9.0220	8.9575	8.9662	−0.0538	−0.0558
Mini	8.9229	8.9227	8.8601	8.8626	−0.0628	−0.0601
Small	8.8964	8.8960	8.8338	8.8341	−0.0626	−0.0618
GPT-1	8.8820	8.8850	8.8209	8.8229	−0.0611	−0.0621

Table 3: English range-token models: validation cross-entropy on OpenWebText vs. CC100. Δ shows the absolute loss change; negative means CC100 is easier (0.06 → 0.7 % relative). Forward/backward parity remains, corroborating the no-AoT finding across corpora.

order; German has more verb-second and case marking that might carry information from the end of a sentence backward. We trained small Nano-sized models on POS sequences for these languages (using the range scheme) for one epoch on ∼ 2GB of each. The results mirrored the English outcome. In Italian, the forward vs backward validation losses backward was 0.002 lower, essentially no difference. Finnish had forward 0.001 lower. German had backward lower by 0.001. These differences are minuscule (see (see Table 4 for exact values). Figure 4 illustrates the forward/backward loss comparison for Italian, German and Finnish models. We note an interesting side point: the loss differed by language (German's POS loss was lower, likely because German's strong grammatical markings reduce uncertainty in tag sequences, whereas Finnish, having many possible forms, resulted in higher loss). But crucially, no language we tested exhibited a forward direction advantage on POS data the backward model was just as good. This suggests that the absence of AoT under grammatical-only modeling is not specific to English but holds across languages, even ones with very different syntactic structures.

Table 4: Nano-Range models (CC100) in three additional languages.

Language	Forward ↓	Backward ↓	Δ (%)
Italian	9.1680	9.1663	–0.02
Finnish	9.9221	9.9233	+0.01
German	8.0346	8.0340	–0.01

5.3 Comparison to Full-Text Models

It is informative to contrast our findings with the known AoT effect on full text. Papadopoulos et al. [1] reported that for a comparable small Transformer on English text, the forward model achieved slightly better loss, and that this gap grows with model size and context length. In our case, even at the largest model (GPT-1 size) and with a context of 64, we saw no gap. To ensure we weren't missing anything, we also ran a control experiment training a Nano model on English word-level text (using a GPT-2 BPE tokenizer on the same data, for 3 epochs). Indeed, on actual text the forward model began outperforming backward noticeably as training progressed (we observed a small cross-entropy gap opening up, consistent with the literature). That gap was nowhere to be found when the same experiment was done on POS

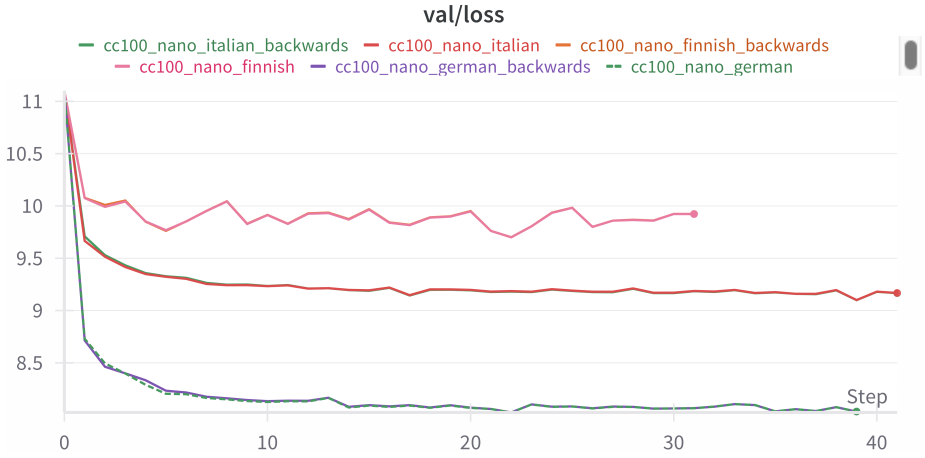

Figure 4: Validation loss for Italian, Finnish, German Nano English Range Model.(Lines are overlapping due to no Arrow-of-Time)

tags. This reinforces our conclusion: whatever causes forward models to have an edge on real text is nullified when lexical content is removed.

5.4 Negative Arrow-of-Time Check

We also specifically looked for any scenario in our experiments where the backward model would outperform the forward model ("negative AoT"). As mentioned, Papadopoulos et al. [1] saw a tiny backward win for a small English model. In our results, the backward loss was sometimes slightly lower than forward (see German above, or one seed of English nano), but these differences were extremely small and not consistently in one direction. We suspect that the previously reported English anomaly might have been a statistical quirk or related to specific word-level effects (e.g., perhaps the particular initialization or the presence of rare tokens). In the grammar-only domain, we did not find any repeatable negative AoT effect. The largest relative backward gain we saw was $\sim 0.02\%$ (which is negligible). Thus, our work does not offer a direct explanation for that anomaly, since grammar alone doesn't produce it, it likely arises from some interaction of semantics or token distribution in actual language. We note that our runs were only 1 epoch; it's conceivable that with prolonged training backward vs forward could diverge in some way, but given how closely the losses tracked, it seems unlikely. All evidence from our evaluation indicates a symmetric learnability of syntax in both time directions.

Overall, our evaluation answers the core question: when restricted to grammatical information, language models show no inherent Arrow of Time. All quantitative comparisons were made in terms of cross-entropy loss, and in every case the forward and backward cross-entropies were the same within experimental error. We did not need to employ significance testing because the differences were on the order of 1e-3 (far smaller than day-to-day training noise). If an Arrow of Time is present, it is below the detectable limit in this setting. The next section discusses what this implies about the source of the Arrow of Time in full language modeling and future investigations.

5.5 Residual Semantic Signal in POS-Only Evaluation

To understand if some semantic information remained in our tokenization, we created a script (evaluate-sentence-loss.py) that streams a 1 MB slice of CC-100–EN, rewrites it as a *backwards* POS-tag sequence, and then reports two token-level cross-entropy metrics:

- **Sentence-avg**: each sentence is re-initialised independently (`BOS` token reset), so no information can flow across sentence boundaries;
- **Full-context**: the whole slice is treated as one contiguous stream, letting the model use previous-sentence context.

Table 5: Backward-direction validation loss on POS tags (1 MB CC-100–EN). Lower is better.

Model	Sentence-avg	Full-context
GPT-1 (110 M)	9.0554	8.9480
GPT-nano (3 M)	9.1744	9.1401

The consistent gap between the two columns (Table 5) is small (roughly $\sim$0.1 nats) but systematic: allowing context across sentence boundaries *always* reduces loss. Because lexical identities have been stripped out, the only information available in the context window is high-level syntactic patterning plus whatever **semantic regularities still leak through POS sequences** (e.g. discourse-level topic continuity reflected in tag frequencies).

This residual gain tells us that completely removing semantics would require *both* lexical anonymisation *and* randomising sentence order; otherwise the model can still exploit subtle distributional cues such as:

1. stylistic consistency of authors across successive sentences;
2. discourse markers (`IN`, `CC`) that hint at upcoming clause structure;
3. punctuation patterns (`.`, `:`, `--`) signalling rhetorical flow.

Hence, although Sections 5.1 and 5.2 demonstrated the disappearance of a measurable Arrow-of-Time once lexical content is removed, Table 5 shows that a non-trivial amount of higher-order information is still present. Future ablations (e.g. shuffling sentence order or injecting random dummy sentences) could quantify exactly how much of this remaining signal is truly semantic versus purely sequential. This however does not invalidate our results since even with the remaining semantic content we still do not find a meaningful Arrow of Time.

6 Discussion

Our findings suggest a clear conclusion: syntactic structure by itself does not induce a preferred temporal direction for learning. Unlike full text, where models consistently perform a bit better in the forward direction [1], POS tag sequences are equally well learned in either direction. We interpret this to mean that the previously observed Arrow of Time (AoT) in language models is not due to inherent asymmetry in grammatical dependencies. In other words, the grammar of English (or Italian, Finnish, etc.) does not seem to be fundamentally easier to model forward than backward. This is perhaps unsurprising from a linguistics perspective, grammar rules can often be applied bidirectionally. For example, if a model knows that determiners are followed by nouns in forward text, in backward text it sees nouns are preceded by determiners; the statistical relationship is essentially symmetric. Our results empirically confirm that a Transformer can capture these syntactic relations in either direction without preference.

So why then do LLMs trained on actual words show an Arrow of Time? The implication is that the AoT effect stems from semantic and/or distributional properties of lexical sequences, rather than pure syntax. Several possible reasons arise:

- **Semantic Coherence:** In natural text, information is introduced and developed as the sentence progresses forward. Causal or logical relations typically go forward in time (cause precedes effect, subject before predicate, etc.). A forward LM can leverage partial meaning to predict what comes next, whereas a backward LM is trying to infer causes from effects, which may be inherently harder. Our POS-only models had no semantics to leverage or infer, neutralizing this factor.
- **Uneven Information Distribution:** Often the beginning of a sentence (or document) sets context that constrains what can come later more than vice versa. For example, in English the first word could be a capitalized proper noun which gives a lot of info about the subject; backward, the model sees a period first which only tells it the sentence ended. Even within a sentence, certain languages (like English, Chinese) concentrate more information in prefixes (e.g., determiners, function words early) while others (like German subordinate clauses) may reveal crucial info at the end. Papadopoulos et al. [1] noted AoT magnitude differed by language, French had a larger forward advantage than English, possibly due to grammatical gender agreement that goes from

noun to adjective (backward model would see adjective first without knowing gender). In our POS experiments, however, such patterns are still present (the tags indicate gender, number, etc.), yet no AoT appeared. This might mean that while those grammatical agreements exist, the model can learn them backward just as well (e.g., seeing an adjective tag that implies a noun of certain gender came before – it can learn that inverse relation).

- **Tokenization Artifacts & Rare Words:** Another consideration is that in real text, unusual or rare tokens might behave differently forward vs backward. For example, a backward model encountering an uncommon word might not have seen its continuation often. We removed all specific words, so this effect is gone. Papadopoulos et al. [1] experimented with different tokenizations and still saw AoT, so it's not solely a subword artifact, but rare named entities or numbers could still asymmetrically affect backward models (perhaps backward needs to predict preceding context for a number which is harder).
- **Model initialization and training dynamics:** There is a chance that Transformers simply optimize better in the forward direction because that's how their causal mask is oriented by default (though theoretically backward is just a relabeled version of forward with reversed data, optimization should be the same). We gave both models identical optimization settings and saw no issue converging backward. This suggests training dynamics are not the cause; the backward models were not inherently harder to train on POS data. On full data, however, one could imagine the backward task being slightly less smooth or learnable early on. Our results can't fully rule that out, but if it were just an optimization quirk, one might think it would also appear with POS data (it did not).

Regarding the "negative AoT" anomaly Since our POS experiments did not show any case where backward significantly outperformed forward, we hypothesize that the earlier report of a backward edge for a tiny English model was likely due to lexical factors in that specific setting. Perhaps with a very small model, predicting some high-frequency functional words backward was easier than forward by chance. It could also have been an artifact of how the data was structured or a statistical fluctuation (the difference was very small). To truly get to the bottom of that, one could replicate the original scenario multiple times to see if backward reliably wins or if it averages out. Our controlled grammar-only environment indicates no inherent reason for backward to be better; thus if backward sometimes wins on real text, it might just be noise or a balance tipping due to content in that dataset. Further tests on semantics vs syntax: One future direction is to introduce some semantic content in a controlled way to see when AoT reappears. For example, one could shuffle word order within sentences (breaking syntax but keeping words) or conversely keep syntax and shuffle content words (nonsense sentences with correct grammar) to see which condition yields an AoT. Our results combined with Papadopoulos et al.'s [1] suggest it is the content that matters: a model on real sentences (even if word order is randomized beyond grammar) might still show forward bias if meaning flows forward. This is speculative, but our work narrows the possibilities.

Cross-lingual observations It is interesting that languages as different as Finnish and German showed the same no-effect result. One might have expected that in a language like German, where verbs often appear at the end of a clause, a backward model (seeing the verb first) might have an easier time modeling the rest of the clause than a forward model (which must wait to see the verb at the end). However, our POS models did not exhibit any forward/backward difference for German. This suggests that, at least at the POS level, any such effect is negligible. It could be that because our context window was relatively short (64 tokens), we didn't capture entire long clauses; in a longer context setting, perhaps language-specific differences could emerge. Papadopoulos et al. [1] did find that longer context windows increased AoT magnitude.

In conclusion, our discussion points to a likely cause for the Arrow of Time in full LMs: the presence of meaning and how information is structured in real language. The grammatical backbone alone is time-symmetric for learning purposes. This insight refines our understanding of LLM training, it suggests that to fully explain AoT, we need to look at higher-level factors (semantics, world knowledge, discourse). It also is a reassuring result in a way: it tells us that our models are not inherently biased by the direction on trivial syntactic grounds; instead, they are picking up genuine asymmetry from data. For language generation applications, this could mean that combining a forward and backward model might capture more nuance (since grammar is handled fine by both, and differences come from content). Indeed, prior works like bidirectional LMs used that intuition, though not framed as AoT.

7 Conclusion

We conducted an in-depth investigation into the role of grammar in the Arrow of Time phenomenon for language models. By training transformers on sequences of POS tags, thereby isolating syntactic structure and removing lexical semantics, we tested whether a forward vs. backward performance asymmetry still occurs. Our experimental results are unambiguous: there is no detectable Arrow of Time when models are limited to grammatical information only. Forward and backward-trained models achieved virtually identical cross-entropies across all conditions (model sizes, languages, tokenization schemes) in our study. This contrasts sharply with previous findings on natural text, where forward models held a consistent slight advantage. The implication is that the source of the forward-direction bias in language modeling lies in aspects of language that were absent in our POS data, likely the realm of semantics and how information is conveyed in normal word sequences.

We also addressed the curious case of a "negative" Arrow of Time reported for English with a small model. Our grammar-only experiments did not reproduce any scenario where the backward model robustly outperforms the forward model. This suggests that that anomaly is not rooted in grammatical structure; it remains an open question requiring further investigation with lexical data. One hypothesis is that it was an artifact or that certain semantic cues in English small data might occasionally favor backward prediction. Resolving this will require targeted experiments (for example, testing many random initializations or examining specific constructions that could invert difficulty).

Future Work The conclusions of this project lead to several follow-ups. To pinpoint the cause of AoT, one could incrementally introduce aspects of language back in. For example, train models on semantically scrambled text (to break real-world logic but keep word distributions) or on POS tags augmented with some semantic signals, to see when the forward/backward gap reappears. Another avenue is examining discourse-level order, our experiments were sentence-local (short context); perhaps the Arrow of Time is more pronounced with paragraphs or narrative progression. Finally, exploring the AoT with different architectures (e.g., recurrent networks) on grammar-only data could confirm if our findings hold universally (the original AoT paper suggests they likely would [1]). Overall, this study has eliminated one potential explanation (syntactic one) for the Arrow of Time. The quest now is to explain the remaining asymmetry observed in natural language modeling, an endeavor that will deepen our understanding of how LLMs learn the structure and meaning of human languages in different temporal directions.

Code Availability

All code used to train the models, reproduce the POS-only evaluator, and generate the results in is released under the permissive **MIT License**. The latest version is archived at

`https://github.com/larosafrancesco289/syntactic-arrow-of-time`

References

[1] V. Papadopoulos, J. Wenger, and C. Hongler, "Arrows of Time for Large Language Models," *arXiv:2401.17505 [cs.CL]*, 2024.

[2] C. E. Shannon, "Prediction and Entropy of Printed English," *Bell System Technical Journal*, vol. 30, no. 1, pp. 50-64, 1951.

[3] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to Sequence Learning with Neural Networks," in *Advances in Neural Information Processing Systems 27*, Proc. of NIPS, 2014, pp. 3104-3112.

[4] L. Mou, R. Yan, G. Li, L. Zhang, and Z. Jin, "Backward and Forward Language Modeling for Constrained Sentence Generation," *arXiv:1512.06612 [cs.CL]*, 2016.

[5] A. Karpathy, "NanoGPT," *GitHub repository*, 2023. [Online]. Available: `https://github.com/karpathy/nanogpt`

[6] A. Hägele, E. Bakouch, A. Kosson, L. Ben Allal, L. Von Werra, and M. Jaggi, "Scaling Laws and Compute-Optimal Training Beyond Fixed Training Durations," *arXiv:2405.18392 [cs.LG]*, 2024.

[7] La Rosa, Y. N. (2025). *Arrow-of-Time Grammar Codebase (v1.0)*. MIT License. Available at `https://github.com/larosafrancesco289/syntactic-arrow-of-time`.

A Problem Statement

We have two Transformer-based language models trained on sequences derived from POS tags:

1. **ID Model:**
   - **Input:** Sequence of POS tags.
   - **Tokenization:** Each unique POS tag (e.g., 'NN', 'VB') is mapped to a unique integer ID. The vocabulary consists of these unique tag IDs. Let the size of this vocabulary be V_{id}.
   - **Prediction Task:** Predict the integer ID corresponding to the *next* POS tag in the sequence.
   - **Loss Function:** Standard cross-entropy loss, denoted as $\mathcal{L}_{\text{id}}$.
2. **Bigger Model:**
   - **Input:** Sequence of integers derived from POS tags.
   - **Tokenization:** Each unique POS tag is assigned a continuous range of integers, $[\text{start}_{\text{pos}}, \text{end}_{\text{pos}})$. Let $\text{Size}(\text{pos}) = \text{end}_{\text{pos}} - \text{start}_{\text{pos}}$ be the number of integers allocated to tag pos. During tokenization, each occurrence of pos is mapped to a *randomly chosen* integer within its assigned range. The total vocabulary size is $V_{\text{big}} = \sum_{\text{pos}} \text{Size}(\text{pos})$.
   - **Prediction Task:** Predict the specific integer token corresponding to the *next* POS tag's randomly assigned integer in the sequence.
   - **Loss Function:** Standard cross-entropy loss, denoted as $\mathcal{L}_{\text{big}}$.

The goal is to find a theoretical relationship between $\mathcal{L}_{\text{id}}$ and $\mathcal{L}_{\text{big}}$ and develop a procedure to estimate $\mathcal{L}_{\text{big}}$ given a trained ID model and the configuration of the Bigger Model's vocabulary.

B Model Definitions and Loss Functions

Let the sequence of target tokens (for a given dataset of length N) be $y_1, y_2, \ldots, y_N$.

B.1 ID Model

The vocabulary size is V_{id}. The model outputs a probability distribution $p_{\text{id}}(y|\text{context})$ over the V_{id} possible POS tag IDs. The cross-entropy loss is:

$$\mathcal{L}_{\text{id}} = -\frac{1}{N} \sum_{i=1}^{N} \log p_{\text{id}}(y_{\text{id},i}|\text{context}_i) \tag{1}$$

where $y_{\text{id},i}$ is the correct POS tag ID at step i, and log denotes the natural logarithm.

B.2 Bigger Model

The vocabulary size is V_{big}. Each POS tag pos has an associated range Range(pos) of size Size(pos). The total vocabulary is the union of these ranges. The model outputs a probability distribution $p_{\text{big}}(y|\text{context})$ over the V_{big} possible integer tokens. The cross-entropy loss is:

$$\mathcal{L}_{\text{big}} = -\frac{1}{N} \sum_{i=1}^{N} \log p_{\text{big}}(y_{\text{big},i}|\text{context}_i) \tag{2}$$

where $y_{\text{big},i}$ is the specific integer token (randomly chosen from the correct POS tag's range) at step i.

C Relating the Cross-Entropies

Let's consider a single prediction step i.

- Let pos_i be the true POS tag corresponding to the target at step i.
- Let $y_{\text{id},i}$ be the integer ID for pos_i in the ID Model's vocabulary.

- Let $y_{\text{big},i}$ be the specific integer token assigned to this instance of pos_i in the Bigger Model's data. Note that $y_{\text{big},i} \in \text{Range}(\text{pos}_i)$.

The contribution to the loss at step i for each model is:

$$H_{\text{id}}(i) = -\log p_{\text{id}}(y_{\text{id},i}|\text{context}_i) \tag{3}$$
$$H_{\text{big}}(i) = -\log p_{\text{big}}(y_{\text{big},i}|\text{context}_i) \tag{4}$$

We can decompose the probability assigned by the Bigger Model using the law of total probability, conditioning on the underlying POS tag pos_i:

$$p_{\text{big}}(y_{\text{big},i}|\text{context}_i) = p_{\text{big}}(y_{\text{big},i}|\text{pos}_i, \text{context}_i) \times P_{\text{big}}(\text{pos}_i|\text{context}_i) \tag{5}$$

where $P_{\text{big}}(\text{pos}_i|\text{context}_i)$ is the total probability mass the Bigger Model assigns to the correct POS tag's range, i.e., $P_{\text{big}}(\text{pos}_i|\text{context}_i) = \sum_{t \in \text{Range}(\text{pos}_i)} p_{\text{big}}(t|\text{context}_i)$.

C.1 Approximations

To relate the two models, we introduce two key approximations:

1. **POS Prediction Equivalence:** We assume that the Bigger Model, despite its more complex task, learns to predict the underlying *correct POS tag* with roughly the same accuracy as the ID Model. Mathematically, the total probability mass assigned to the correct tag's range by the Bigger Model is approximately equal to the probability assigned to the correct tag ID by the ID Model:

$$P_{\text{big}}(\text{pos}_i|\text{context}_i) \approx p_{\text{id}}(y_{\text{id},i}|\text{context}_i) \tag{6}$$

2. **Uniform Within-Range Distribution:** Since the specific token $y_{\text{big},i}$ within the range $\text{Range}(\text{pos}_i)$ was chosen *randomly* during tokenization, there is no semantic information distinguishing tokens within the same range. We assume the Bigger Model learns to distribute its probability mass approximately uniformly across the tokens within the correctly predicted POS tag's range. Therefore, the conditional probability of predicting the specific token $y_{\text{big},i}$, given that the model has focused on the correct tag pos_i, is:

$$p_{\text{big}}(y_{\text{big},i}|\text{pos}_i, \text{context}_i) \approx \frac{1}{\text{Size}(\text{pos}_i)} \tag{7}$$

C.2 Derivation

Substituting these approximations into Equation (5):

$$p_{\text{big}}(y_{\text{big},i}|\text{context}_i) \approx \frac{1}{\text{Size}(\text{pos}_i)} \times p_{\text{id}}(y_{\text{id},i}|\text{context}_i) \tag{8}$$

Now, consider the cross-entropy contribution $H_{\text{big}}(i)$:

$$\begin{aligned}
H_{\text{big}}(i) &= -\log p_{\text{big}}(y_{\text{big},i}|\text{context}_i) && (9)\\
&\approx -\log \left(p_{\text{id}}(y_{\text{id},i}|\text{context}_i) \times \frac{1}{\text{Size}(\text{pos}_i)} \right) && (10)\\
&= -\left[\log p_{\text{id}}(y_{\text{id},i}|\text{context}_i) + \log \left(\frac{1}{\text{Size}(\text{pos}_i)} \right) \right] && (11)\\
&= -\log p_{\text{id}}(y_{\text{id},i}|\text{context}_i) - (-\log \text{Size}(\text{pos}_i)) && (12)\\
&= H_{\text{id}}(i) + \log \text{Size}(\text{pos}_i) && (13)
\end{aligned}$$

To find the average cross-entropy loss over the entire dataset ($\mathcal{L}_{\text{big}}$), we average $H_{\text{big}}(i)$ over all N

steps:

$$\mathcal{L}_{\text{big}} = \frac{1}{N}\sum_{i=1}^{N} H_{\text{big}}(i) \tag{14}$$

$$\approx \frac{1}{N}\sum_{i=1}^{N} (H_{\text{id}}(i) + \log \text{Size}(\text{pos}_i)) \tag{15}$$

$$= \left(\frac{1}{N}\sum_{i=1}^{N} H_{\text{id}}(i)\right) + \left(\frac{1}{N}\sum_{i=1}^{N} \log \text{Size}(\text{pos}_i)\right) \tag{16}$$

$$= \mathcal{L}_{\text{id}} + E[\log \text{Size}(\text{pos})] \tag{17}$$

where $E[\log \text{Size}(\text{pos})]$ is the expected value, or average, of the natural logarithm of the range size corresponding to the *true* POS tag at each step i.

C.3 The Estimation Formula

The estimated cross-entropy for the Bigger Model ($\mathcal{L}_{\text{bigest}}$) is given by:

$$\mathcal{L}_{\text{bigest}} = \mathcal{L}_{\text{id}} + \text{AvgLogRangeSize} \tag{18}$$

where

$$\text{AvgLogRangeSize} = \frac{1}{N}\sum_{i=1}^{N} \log \text{Size}(\text{pos}_i) \tag{19}$$

and pos_i is the true POS tag for the target token at step i. $\mathcal{L}_{\text{id}}$ should ideally be the measured *validation* loss of the trained ID model, as this reflects its generalization performance.

D Conclusion for Appendix

The cross-entropy loss of the Bigger Model ($\mathcal{L}_{\text{big}}$) can be estimated from the cross-entropy loss of the ID Model ($\mathcal{L}_{\text{id}}$) by adding an offset term, AvgLogRangeSize. This term represents the average logarithmic size of the token ranges corresponding to the true POS tags in the sequence. It quantifies the additional uncertainty introduced by having to predict a specific token within a range, assuming the underlying POS tag prediction capability remains similar and the probability is distributed uniformly within the correct range. This provides a valuable theoretical estimate for understanding the expected difficulty increase due to the vocabulary expansion strategy.