Hitchhikers' Guide to Masked Latent Semantic Modeling

Anonymous authors
Paper under double-blind review

Abstract

Masked Latent Semantic Modeling (MLSM) is a recent pre-training objective which offers a sample efficient alternative to the use of Masked Language Modeling (MLM) for training encoder language models. In this paper, we identify and carefully evaluate previously unexplored important properties of MLSM pre-training. Based on the results of our rigorous experiments, we formulate a series of recommendations and best practices regarding MLSM pre-training. With these experiments, we also aim at advancing the understanding and proper use of MLSM pre-training by filling in important voids of previous empirical investigations. We release our code for reproducing our experiments at `github.com/[MASK]`

1 Introduction

Masked Latent Semantic Modeling (MLSM; Berend, 2023) has been recently proposed as a sample efficient alternative to masked language modeling (MLM). The core idea behind MLSM is that instead of requiring language models to output the exact identity of the masked words, they are expected to output a context-sensitive semantic characterization of the masked words in terms of their latent semantic properties as their pre-training task.

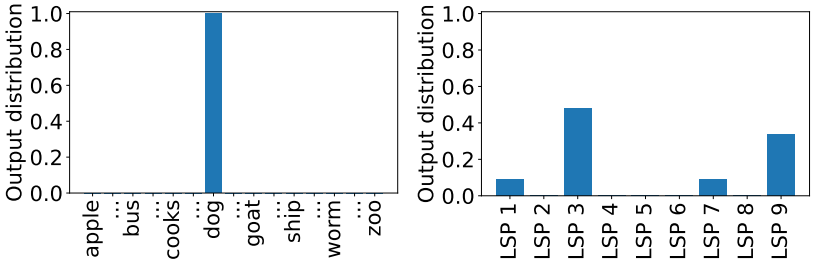
We illustrate the different kinds of outputs that models pre-trained with the MLM and MLSM objective are expected to deliver in Figure 1. In case the masked token happens to be the word *dog*, the MLM pre-training loss is minimized for the particular word if the model outputs *all* the probability mass to that exact token (Figure 1a).

This means that during the individual updates of MLM, we assume that for a particular masked token, there exist only a single unique correct substitute. As MLM pre-training progresses, language models become capable of outputting meaningful probability distributions for the substitutes of masked tokens, however, this requires vast amounts of diverse training data, compromising its sample efficiency.

In contrast to MLM pre-training, the target output distribution of the model when using MLSM ranges over the latent semantic properties (abbreviated as LSP in Figure 1b) of the masked tokens. The LSPs used during MLSM pre-training are determined in an unsupervised manner from an auxiliary teacher model by expressing its hidden representations as a sparse linear combination of a codebook of semantic atoms.

Even though MLSM pre-training has been shown to be more sample efficient than MLM, there are important open questions and practical considerations not discussed thoroughly enough, begging for further investigations and supportive empirical evidences. Our paper aims at answering those research questions via rigorous experiments for improving the understanding of MLSM pre-training. More specifically, we investigate the following research questions:

- how can we improve the efficiency of MLSM,
- what are the effects of determining LSPs on hidden representations that originate from different layers of the auxiliary model,
- what are the effects of using different number of LSPs,
- how to test and improve linguistic capabilities of MLSM pre-trained models.



(a) MLM objective (b) MLSM objective

Figure 1: Comparisons of the probability distributions required by MLM (a) and MLSM (b) pre-training, with LSP referring to the latent semantic properties devised from the hidden representations of an auxiliary model in an unsupervised manner.

Throughout the paper, we highlight our key findings with colored background. We also release the pre-trained models that we created for our experiments at HuggingFace Hub, as well as our code base at `github.com/[MASK]`.

2 Related work

Prior work suggests that sparse representations obtained from dense hidden vectors can convey the semantic properties of the words they describe (Berend, 2020; Yun et al., 2021). The core idea behind MLSM pre-training is that the latent semantic decomposition of the individual context-sensitive token representations can be exploited as a pre-training signal. Our paper advances knowledge on MLSM pre-training by providing a series of thorough experiments, related to various important implementation choices that has been previously overlooked in the literature. We also investigate the training costs and offer an improved implementation of MLSM.

MLSM also relates to prior research involving the integration of semantic categories into language model pre-training (Levine et al., 2020; Bai et al., 2022; Shani et al., 2023). The way MLSM progressed these approaches is that it determines the semantic properties to recover by the model in a context-sensitive manner, not requiring any external linguistic resources, such as WordNet (Fellbaum, 1998) or ConceptNet (Speer et al., 2017), hence it is capable of considering conceptual relations that go beyond hypernymy.

Since MLSM requires an auxiliary teacher model for determining the distribution of latent semantic properties of the masked tokens that the language model needs to recover as its pre-training task, it can be naturally framed as a special kind of knowledge distillation technique (Hinton et al., 2015; Aguilar et al., 2020). MLSM pre-training is also related to the line of research that incorporates alternative pre-training objectives as opposed to outputting the exact identity of the masked token (Levine et al., 2020; Yamaguchi et al., 2021; Alajrami & Aletras, 2022).

Despite the growing prominence of decoder-only generative large language models, encoder-based language models remain highly relevant in contemporary applications, such as retrieval-augmented generation (RAG). This is evidenced by the recent release of advanced encoder-based models like ModernBERT (Warner et al., 2024) or EuroBERT (Boizard et al., 2025). Our research also relates to research efforts that aim at enhancing the efficiency of pre-training for encoder-based language models (Izsak et al., 2021; Portes et al., 2023; Geiping & Goldstein, 2023). Our approach, however, differs from these earlier works, which primarily focused on architectural optimizations and design choices within the traditional pre-training paradigm. Our approach, however, differs from these earlier works, as they were focusing on architectural speedups and design choices of encoder models trained with the traditional pre-training paradigm, whereas we focus on modification of the learning objective in order to make pre-training more sample efficient and better aligned with human perception.

Most recently, sparse autoencoders have became a popular tool for offering post-hoc interpretabiliy of LLM behavior (Huben et al. 2024; Lieberum et al. 2024; He et al. 2024; *inter alia*). This line of research relates to ours in that both involve sparse coding of neural activations, with the core difference being that we make use of the sparsified representations for improving the pre-training phase of encoder-based models.

3 Masked Latent Semantic Modeling

We first overview the MLSM pre-training technique, as it plays a central role in our experiments. MLSM improves pre-training sample efficiency by changing the domain of the output distribution of the model from its subword units to the inventory of latent semantic properties (LSPs). Sample output distributions for the two pre-training paradigms are also illustrated in Figure 1.

The way MLSM determines the LSP distribution of a token is by relying on an auxiliary model $\mathcal{T}$. In a preparatory phase, a sample of hidden representations produced by $\mathcal{T}$ is collected from its layer l as $\{\boldsymbol{h_1^{(l)}}, \ldots, \boldsymbol{h_N^{(l)}}\}$. A dictionary learning problem (Mairal et al., 2009) is then solved of the form

$$\underset{\boldsymbol{D}^{(l)}, \boldsymbol{\alpha_j} \in \mathbb{R}^k_{\geq 0}}{\arg\min} \sum_{j=1}^{N} \frac{1}{2} \|\boldsymbol{h_j^{(l)}} - \boldsymbol{D}^{(l)} \boldsymbol{\alpha_j}\|_2^2 + \lambda \|\boldsymbol{\alpha_j}\|_1, \tag{1}$$

where $\boldsymbol{D}^{(l)} \in \mathbb{R}^{d \times k}$ is a dictionary matrix, with column vector norms bounded by 1, $\boldsymbol{\alpha_j} \in \mathbb{R}^k$ contains the sparse linear coefficients that indicate the extent to which the vectors from $\boldsymbol{D}^{(l)}$ are used in reconstructing the d-dimensional hidden representation from the l-th layer of $\mathcal{T}$, $\boldsymbol{h_j^{(l)}} \in \mathbb{R}^d$. λ serves as a regularization coefficient, controlling for the level of sparsity in $\boldsymbol{\alpha_j}$.

Solving equation 1 is a one time effort, performed before the actual pre-training phase, having a negligible ($\ll 1\%$) computational overhead compared to the typical costs of pre-training. Once the dictionary matrix $\boldsymbol{D}^{(l)}$ is determined, it is used for determining the sparse contextualized representation for any $\boldsymbol{h_i^{(l)}}$, i.e., a hidden state from layer l of $\mathcal{T}$ as

$$\underset{\boldsymbol{\alpha_i} \in \mathbb{R}^k_{\geq 0}}{\arg\min} \frac{1}{2} \|\boldsymbol{h_i^{(l)}} - \boldsymbol{D}^{(l)} \boldsymbol{\alpha_i}\|_2^2 + \lambda \|\boldsymbol{\alpha_i}\|_1. \tag{2}$$

Objective equation 2 is computationally convenient, as it does not require optimizing towards $\boldsymbol{D}^{(l)}$. With $\boldsymbol{D}^{(l)}$ being fixed from equation 1, obtaining the sparse linear coefficients $\boldsymbol{\alpha_i}$ constitutes an efficient to solve LASSO optimization problem.

Thanks to the non-negativity constraint imposed towards $\boldsymbol{\alpha_i}$ in equation 2, the ℓ_1-normalized sparse linear coefficients can be conveniently treated as probability distributions over the k latent semantic properties (that are tied to the individual column vectors of $\boldsymbol{D}^{(l)}$). MLSM pre-training then considers these sparse normalized distributions of latent semantic properties of the masked tokens as the desired target outputs and it uses the Kullback–Leibler divergence as its loss function.

3.1 Improving the efficiency of MLSM

Intuitively, MLSM pre-training may appear to be strictly more resource intensive than MLM pre-training owing to the use of the auxiliary model $\mathcal{T}$. However, this is not necessarily the case, as the output space of an MLSM pre-trained model spans a considerably smaller domain compared to that of an MLM pre-trained model. Indeed, the output space of MLM models range over the entire vocabulary of the model V, whereas MLSM models output a distribution over the k LSPs, with $|V| \gg k$.

This means that for MLSM, the final unembedding layer – transforming the output of the final transformer block into model logits – can be of the shape $d \times k$ (as opposed to $d \times |V|$), where d is the dimensions of the hidden state. The amount of FLOPS saved for producing model outputs outweigh the computational overhead of determining the target distribution of LSPs from the auxiliary model $\mathcal{T}$.

Berend (2023) mentions that "we introduce k new special symbols into the output vocabulary of the model". That is, the number of parameters in the unembedding layer was $d \times (|V| + k)$, instead of $d \times k$. Additionally, in the official implementation of MLSM, a full forward pass over $\mathcal{T}$ – including the calculation of the model logits – was conducted, even though MLSM requires only the hidden states of the auxiliary from its layer l. This means that for an auxiliary model $\mathcal{T}$, which uses hidden states of dimension $d_{\mathcal{T}}$ and consists of L layers in total, the forward pass in $\mathcal{T}$ for layers $\{l+1, \ldots, L\}$, as well as the computation related to the unembedding operation (involving multiplication with a $d_{\mathcal{T}} \times |V|$ matrix) can be saved, providing a substantial improvement in the efficiency of MLSM pre-training.

4 Experiments

Throughout our experiments, we pre-trained multiple models that we evaluate over a diverse set of downstream tasks. We next describe the details of pre-training and evaluation involving both fine-tuning and zero-shot results.

4.1 Pre-training experiments

We relied on the `20230920` dump of the English Wikipedia as our pre-training corpus that we cleaned with the WikiBERT pre-processing pipeline (Pyysalo et al., 2021). During pre-training, we used the AdamW optimizer (Loshchilov & Hutter, 2019) with linear learning rate scheduling with a peak value of 1e$-$4, having a warm-up phase that constituted 2% of the total 100,000 update steps. Throughout pre-training, we employed an effective batch size of 1024 (using gradient accumulation of 8 with a batch size of 128) and a maximal sequence length of 128 subtokens.

We additionally ensured for better comparability between the different pre-training trials that the individual batches had identical contents and were processed in identical order. All together we conducted pre-training over approximately 102 million (100,000*1024) input sequences, spanning over nearly 13 billion tokens, i.e., 4 epochs over our 3.3 billion tokens pre-training corpus. As a comparison, the official BERT models were pre-trained over 1 million pre-training steps, which corresponds to roughly 40 epochs over a 3.3 billion word corpus used (see A.2 in (Devlin et al., 2019)).

We created all our models from scratch and with the same architecture as `DeBERTa-base` (He et al., 2021). That is, our models consisted of 12 transformer blocks and employed a hidden vector of 768 dimensions.Not counting the parameters responsible for producing the outputs, this configuration of DeBERTa comes with approximately 100 million model parameters, i.e., 8.3 million per encoder layer. Our MLSM models thus had substantially higher capacity compared to the models originally pre-trained with MLSM, i.e., Berend (2023) created 8 layer encoders with 512 dimensional hidden representations that resulted in approximately 25 million parameters related to the transformer blocks. Our experimental results hence allow us to assess if the benefits of MLSM pre-training also apply for models with substantially increased capacity.

Pre-training one model for our experiments took approximately 3 GPU days on a single NVIDIA A6000. This is in stark contrast to the computation involved in the creation of the official DeBERTa model that was trained for nearly 2,000 V100 GPU days (the 1 million update steps that they conducted took about 20 days, using 96 V100 GPUs (He et al., 2021)).

4.1.1 MLSM related hyperparameters

When performing MLSM, we relied on `bert-base` as the auxiliary model $\mathcal{T}$. Our auxiliary model – excluding its unembedding layer not required for producing the LSPs – consists of approximately 96 million parameters, but thanks to the improvements discussed in Section 3.1, we did not have to perform a full forward pass over the entire network for obtaining the LSP profile of the masked tokens.

The exact amount of computation required for determining the LSPs is dependent on hyperparameter l, i.e., the layer of $\mathcal{T}$ for extracting the hidden vectors from. (Berend, 2023) only considered using the hidden representations from the very last layer of $\mathcal{T}$ (i.e., $l = 12$), whereas in Section 4.2.1, we provide a series of controlled experiments on the role of altering the value of l.

We also investigate the effects on the choice of k, i.e., the number of LSPs to consider during MLSM pre-training. The values we checked were 1, 2, 4, 8 and 16 times the dimensionality of the hidden vectors employed in $\mathcal{T}$, meaning that we had $k \in \{768, 1536, 3072, 6144, 12288\}$. We report our results obtained with different choices of k in Section 4.2.2. Unless stated otherwise, we employ $k = 3072$. For obtaining $\boldsymbol{D}^{(l)}$ according to equation 1, we relied on 2 million hidden vectors obtained from $\mathcal{T}$ produced on texts randomly sampled from our pre-training corpus.

4.1.2 Baseline models

We consider two baselines, the first being vanilla MLM pre-training, for which all overlapping hyperparameters were kept identical to those of MLSM pre-trained models. As MLSM uses a teacher–student paradigm, it is natural to compare to a knowledge distillation (KD) setting (Hinton et al., 2015; Aguilar et al., 2020).

During KD, we also rely on `bert-base-cased` as the auxiliary pre-trained teacher model $\mathcal{T}$. We use $\mathcal{T}$ to provide its predicted distribution for the substitutes of the masked tokens, that we train our student models to replicate. It is important to note that the kind of improvements regarding the use of the auxiliary model $\mathcal{T}$ (as discussed in Section 3.1) cannot be applied during KD pre-training, as KD provides the training signal for the student model by relying on the final output of $\mathcal{T}$. As such, KD pre-training requires strictly more compute compared to both MLM and MLSM.

4.2 Fine-tuning experiments

We primarily measure the quality of the models that we pre-train by quantifying their fine-tuning performance over a wide range of benchmark tasks. During this kind of assessment, we rely on the Corpus of Linguistic Acceptability (COLA; Warstadt et al., 2019), the CoNLL 2003 dataset for NER (Tjong Kim Sang & De Meulder, 2003), the MNLI natural language inference dataset Williams et al. (2018), the Microsoft Research Paraphrase Corpus (MRPC; Dolan & Brockett, 2005), the QNLI benchmark Rajpurkar et al. (2016); Wang et al. (2019b) datasets, Quora Question Pairs (QQP; Iyer et al., 2017), Recognizing Texutal Entailment (RTE; Dagan et al., 2006; Haim et al., 2006; Giampiccolo et al., 2007; Bentivogli et al., 2009), Stanford Sentiment Treebank (SST2; Socher et al., 2013), Semantic Textual Similarity (STSB; Cer et al., 2017) and the Word-in-Context (WiC; Pilehvar & Camacho-Collados, 2019) datasets.

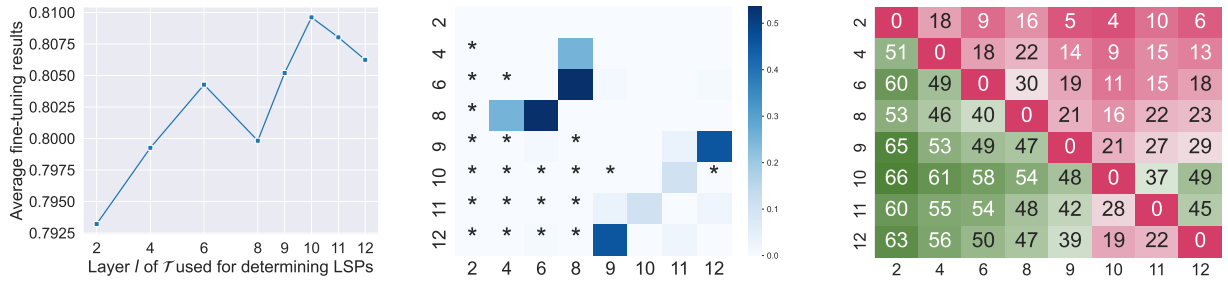
As many of the datasets are part of the GLUE (Wang et al., 2019b) and SuperGLUE benchmarks (Wang et al., 2019a), where the labels of the test set are not available, we performed our evaluation on the development sets. We opted for frequently used hyperparameters for fine-tuning the datasets. That is, we used a learning rate of 2e−5 with linear learning rate scheduling and a batch size of 32, performing 3 epochs. As the evaluation metric, we always report the fine-tuning performance after the last epoch.

We repeated every fine-tuning experiment 5 times and report the average performance of these trials with a differently initialized classification head in order to account for the potential variability in the fine-tuning performances of the individual experiments. We also perform the Wilcoxon signed rank test between pairs of differently pre-trained models, based on those pairs of fine-tuning experiments for which the classification heads of the models were initialized with the same weights. Performing the fine-tuning experiments for a model checkpoint took approximately 1 day on an NVIDIA A6000 GPU.

4.2.1 Experiments on the choice of l

Prior work on MLSM did not carefully evaluate the effects of using hidden vectors originating from other than the final layer of $\mathcal{T}$. This, however, potentially limited MLSM to reach its full potential, in light of prior evidence suggesting that the hidden representations of the last transformer block might not convey the most useful semantic information (Zhao et al., 2020). As discussed in Section 3.1, using earlier layers of $\mathcal{T}$ has the additional benefit of reducing the costs of pre-training, by making LSP calculation cheaper.

We were hence curious whether the utility of the LSPs can be improved by relying on hidden states from earlier layers of $\mathcal{T}$. As such, we went beyond the originally proposed strategy of using the final layer of $\mathcal{T}$ for obtaining LSPs, and trained a series of MLSM models while varying nothing else, but the layer of the transformer block of the auxiliary model $\mathcal{T}$ that we use for determining the LSPs.



(a) Average fine-tuning performance (b) Wilcoxon p-values (c) Number of wins

Figure 2: The effects of using hidden states from different layers of the auxiliary model. (a) Aggregated fine-tuning performance averaged over 70 evaluations for each different choices of layer index. (b) p-values of the Wilcoxon signed rank test. A $*$ at the intersection of a row, column pair labeled with (l_i, l_j) means that the MLSM model relying on layer l_i of $\mathcal{T}$ performed significantly better ($p < 0.01$) than the MLSM model using layer l_j. (c) A value at position (l_i, l_j) refers to the number of paired fine-tuning trials such that the MLSM model pre-trained over the LSPs based on layer l_i of $\mathcal{T}$ scored higher.

We pre-trained a separate MLSM model while relying on the hidden states produced by any of the even indexed transformer blocks of $\mathcal{T}$. Starting with block 9, we also investigated the use of odd indexed blocks, as we originally expected the blocks near to the end of the network to produce more useful hidden states for determining the LSPs. In this section, we use the notation MLSM_l for referring to the MLSM model that was pre-trained based on LSPs obtained from layer l of $\mathcal{T}$.

Our evaluation suite consisted of 14 evaluations, each of which were repeated 5 times, resulting in 70 experiments per model checkpoints. We report the overall averaged results over these 70 fine-tuning scores conducted for each of our pre-trained MLSM models in Figure 2. A detailed breakdown on the individual task performances averaged over the 5 experiments per evaluation criterion can be found in Table 1.

As illustrated by Figure 2a – apart from a small relapse for using layer 8 hidden representations of $\mathcal{T}$ – there is a monotonic increase in the average fine-tuning performance of MLSM_l models for $l \leq 10$, after which point there is a decline in the average fine-tuning results. Our results corroborate our hypothesis that using the last hidden states from $\mathcal{T}$ for determining the LSPs is sub-optimal.

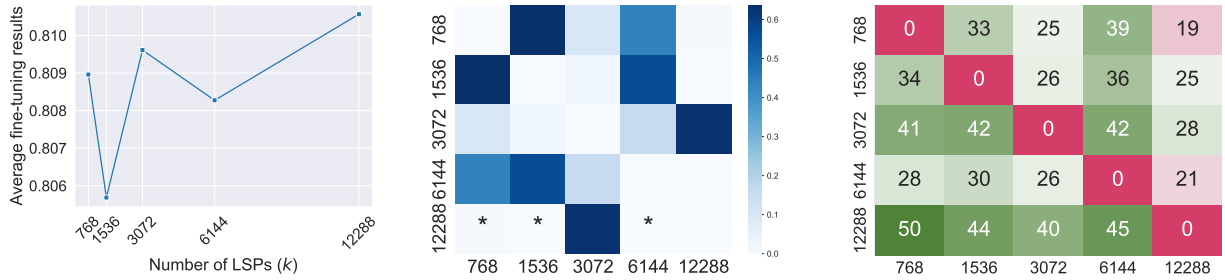
We also calculated the Wilcoxon signed rank test between the 70 pairs of fine-tuning results per pre-trained model pairs, the outcome of which is summarized in Figure 2b. We put a * to those model pairs MLSM_i and MLSM_j indicated by row and column labels i and j, such that the paired fine-tuning results of MLSM_i were significantly better ($p < 0.01$) according to the Wilcoxon signed rank test compared to those of MLSM_j. The values in Figure 2c refer to the number of paired fine-tuning experiments for which MLSM_i scored better than MLSM_j.

Table 1 and Figure 2 clearly indicate that choosing MLSM_{10} over MLSM_{12} should be preferred. MLSM_{10} is not only cheaper to pre-train, but it also performs significantly better than MLSM_{12}. Even though there is no statistically significant difference between MLSM_{10} and MLSM_{11}, pre-training the former is cheaper as it relies on an earlier layer of $\mathcal{T}$. For any other choice of investigated values for l, MLSM_{10} performs significantly better than MLSM_l.

From a practical consideration, when using an L-layered auxiliary model $\mathcal{T}$, it is recommendable to pre-train MLSM_l for a value of l that is not equal but close to L. Such a choice not only reduces the pre-training costs compared to MLSM_L (as the cost of determining LSPs is directly proportional to the number of layers kept from $\mathcal{T}$), but it is also likely to provide LSPs that provide more useful pre-training signal. For the above reasons, we conducted our remaining experiments with MLSM_{10}.

Table 1: The detailed effects on the per-task averaged fine-tuning performance of MLSM models as a function of the layer of $\mathcal{T}$ used for determining the LSPs.

	Layer l of the auxiliary model $\mathcal{T}$ used by MLSM$_l$							
task (metric)	2	4	6	8	9	10	11	12
CoNLL2003 (F1)	0.946	0.947	0.948	0.950	**0.951**	0.949	0.949	0.950
COLA (MCC)	0.339	0.371	0.418	0.372	0.387	0.405	**0.428**	0.410
MNLIm (accuracy)	0.805	0.809	0.815	0.818	0.821	**0.824**	0.823	0.823
MNLImm (accuracy)	0.814	0.819	0.821	0.827	0.828	0.831	**0.833**	0.829
MRPC (accuracy)	0.843	**0.864**	0.861	0.837	0.859	0.860	0.859	0.851
MRPC (F1)	0.892	**0.906**	0.904	0.886	0.902	0.902	0.903	0.896
QNLI (accuracy)	0.881	0.883	0.888	0.898	0.891	**0.902**	0.898	0.895
QQP (accuracy)	0.897	0.899	0.901	0.903	0.901	**0.903**	**0.903**	0.902
QQP (F1)	0.861	0.864	0.865	0.867	0.865	**0.868**	**0.868**	0.867
RTE (accuracy)	**0.608**	0.585	0.588	0.584	0.601	0.601	0.593	0.593
SST2 (accuracy)	0.892	0.899	0.905	0.899	0.909	**0.914**	0.910	0.906
STSB (pearson)	0.845	0.844	0.851	0.857	0.858	**0.859**	0.857	0.856
STSB (spearmanr)	0.842	0.840	0.848	0.855	0.856	**0.857**	0.855	0.853
WiC (accuracy)	0.641	0.662	0.646	0.643	0.642	**0.659**	0.633	0.656
Avg.	0.793	0.799	0.804	0.800	0.805	**0.810**	0.808	0.806



(a) Average fine-tuning performance (b) Wilcoxon p-values (c) Number of wins

Figure 3: The effects of using different number of LSPs. (a) Aggregated fine-tuning performance averaged over 70 evaluations for each different choices of k. (b) p-values of the Wilcoxon signed rank test. A $*$ at the intersection of a row, column pair labeled with (k_i, k_j) means that the MLSM model relying on k_i LSPs performed significantly better ($p < 0.01$) than the MLSM using k_j LSPs. (c) A value at position (k_i, k_j) refers to the number of paired fine-tuning trials such that the MLSM model k_i LSPs scored higher.

4.2.2 Experiments on the number of LSPs

A further previously unexplored question relates to the effects of choosing the number of latent semantic properties to consider during MLSM pre-training, i.e., the number of semantic atoms k, comprising the dictionary matrix $\boldsymbol{D}^{(l)} \in \mathbb{R}^{d \times k}$. The original implementation of MLSM relies on the choice $k = 3000$, however, it is not clear how sensitive the pre-training is to the choice of this hyperparameter.

To this end, we conducted 5 instances of MLSM pre-training under identical circumstances while varying only the value of k over the values $\{768, 1536, 3072, 6144, 12288\}$. We report our results for each task in Table 2 and the average performance over all the evaluations conducted in Figure 3. Looking at the small differences in the figures of Table 2 and Figure 3a, we can conclude that MLSM is robust to the choice of the number of LSPs.

Table 2: The detailed effects of using different number of LSPs (k).

	Number of LSPs (k)				
task (metric)	768	1536	3072	6144	12288
CoNLL2003 (F1)	0.949	0.950	0.949	0.952	0.953
COLA (MCC)	0.407	0.382	0.405	0.392	0.393
MNLIm (accuracy)	0.824	0.826	0.824	0.822	0.823
MNLImm (accuracy)	0.830	0.831	0.831	0.831	0.831
MRPC (accuracy)	0.858	0.862	0.860	0.867	0.871
MRPC (F1)	0.901	0.904	0.902	0.907	0.909
QNLI (accuracy)	0.903	0.900	0.902	0.897	0.901
QQP (accuracy)	0.902	0.903	0.903	0.901	0.903
QQP (F1)	0.867	0.867	0.868	0.866	0.869
RTE (accuracy)	0.607	0.583	0.601	0.598	0.609
SST2 (accuracy)	0.912	0.911	0.914	0.908	0.915
STSB (Pearson)	0.855	0.857	0.859	0.855	0.859
STSB (Spearmanr)	0.852	0.855	0.857	0.853	0.857
WiC (accuracy)	0.658	0.648	0.659	0.665	0.657
Avg.	0.809	0.806	0.810	0.808	0.811

This observation is further supported by the p-values of the Wilcoxon rank test that we present in 3b for the paired fine-tuning experiments belonging to a pair of MLSM models, the pre-training of which only differed in the number of LSPs employed.

Relying on orders more LSPs than previously recommended ($k \gg 3000$) not only results in a slight degradation of the fine-tuning performance of the pre-trained models (for $k = 6144$), but it also increases the computational need of pre-training, as the size of the model parameters responsible for making pre-training predictions is proportional to the number of LSPs (i.e., the unembedding parameters are of size $h \times k$). Based on the above, we conclude that the originally proposed number of latent properties was a reasonable choice, and we continue our experiments with $k = 3072$.

4.2.3 Analyzing pre-training efficiency

We next compare the efficiency of our modified MLSM model with those of MLM and KD pre-trained model variants. Our analysis covers both the investigation of the pre-training costs and the fine-tuning performance of these models.

Computational need We include the cost of pre-training expressed in FLOPS over one batch of inputs with the different pre-training objectives and different MLSM variants in Table 3. We can see that KD and the naïve implementation of MLSM pre-training requires the highest FLOPS, i.e., the extra costs related to the use of the auxiliary model introduces more than +30% training cost over vanilla MLM pre-training. At the same time, the costs related to our improved MLSM implementation is on par with the use of MLM.

Table 3: The per batch costs of different pre-training paradigms and implementations. The naïve implementation of MLSM performs unembedding over $|V| + k$ symbols and a full forward pass over $\mathcal{T}$. In contrast, we perform unembedding over $|k|$ symbols and employ early exit at layer l from $\mathcal{T}$ for MLSM_l.

Pre-training	MLM	KD	naïve MLSM	MLSM_{12}	MLSM_{10}
FLOPS/batch	1.13e+13	1.49e+13	1.50e+13	1.21e+13	1.16e+13
FLOPS %	100%	131.9%	132.7%	107.1%	102.7%

Pre-training dynamics Over the course of pre-training, we created intermediate checkpoints of the models at their 10%, 25%, 50% and 100% readiness level, i.e., after performing 10000, 25000, 50000 and 100000 pre-training update steps. Figure 4 includes the average performance of different model variants as a function of the pre-training steps conducted. Figure 4a reveals the improved sample-efficiency of MLSM pre-training as the average fine-tuning performance of the MLSM model at its 50% readiness level is strictly higher than those of the alternatively pre-trained models at their 100% readiness.

If comparison was made on the basis of FLOPS then the advantage of MLSM over KD was even more pronounced, while the comparison would practically be the same between MLSM and MLM (as KD has +30% extra compute cost over both MLM and MLSM pre-training, whereas MLM and MLSM have comparable computational need). Figure4 (b)–(o) provides further insights into the per evaluation setting performances of the differently pre-trained but otherwise identically fine-tuned models.

In Figure 7 we report the performance metrics of the individual experiments that we conducted by fine-tuning our models at their different readiness levels. In that figure, every marker covers a pair of fine-tuning results obtained by a pair of differently pre-trained models (for the same amount of updates) when initialized with the same set of task oriented classification parameters at the beginning of their fine-tuning. The identical initialization (and the fact that the batches came with identical contents and identical order) makes such fine-tuning experiments comparable with each other. Pairs of fine-tuning experiments for which the MLSM pre-trained model performed better are above the dashed line of the subplots.

We can see in Figure 7 that fine-tuning results obtained by the MLSM pre-trained model almost always reach or surpass those of the alternatively pre-trained models for all tasks and model readiness levels. COLA is the single notable exception, for which the fine-tuning results of alternatively pre-trained models is almost always better than that of MLSM models, which is something that we will focus on next.

4.3 Investigating the linguistic capabilities

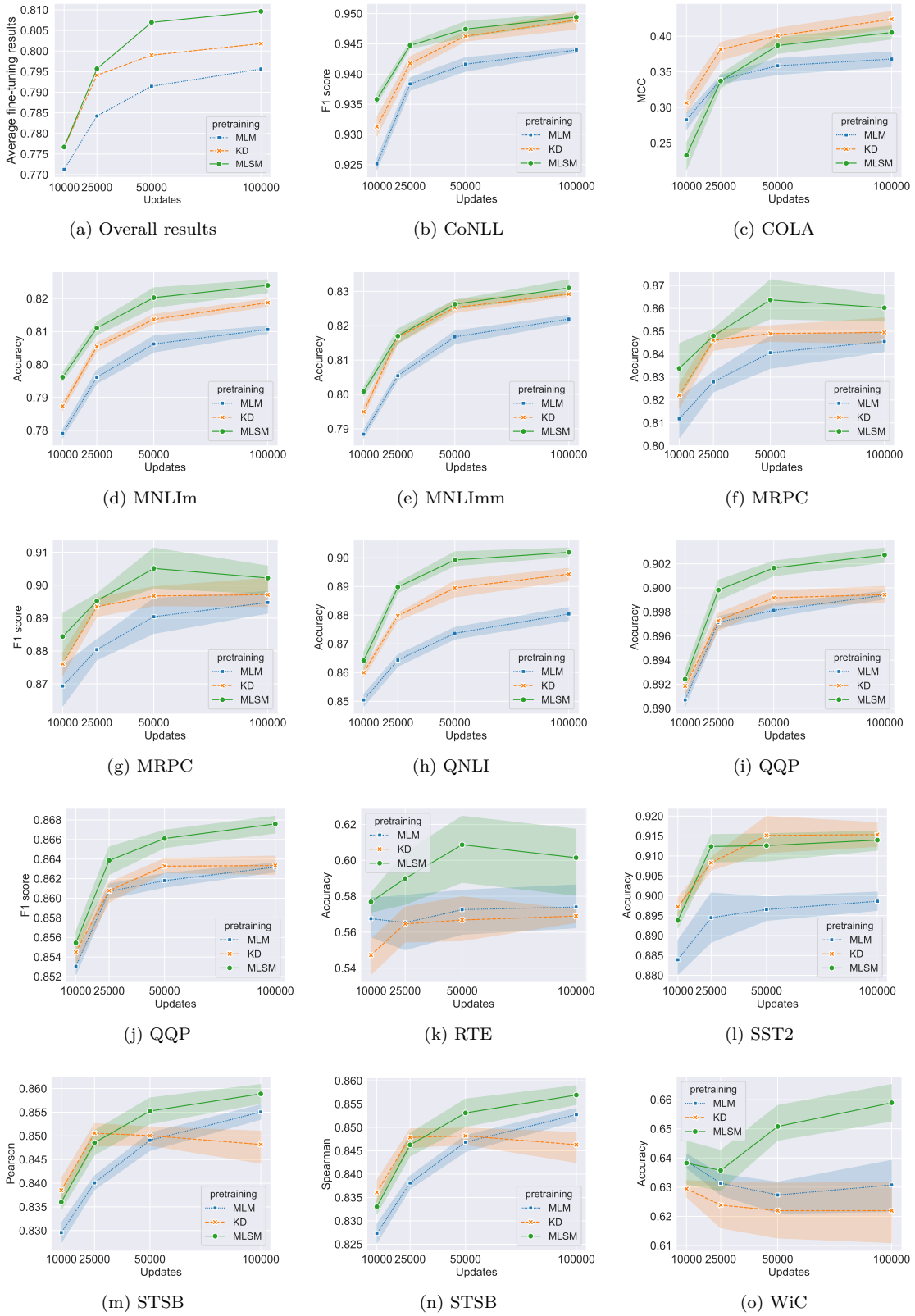
Figure 4 and Figure 7 reveal that COLA is the single task where fine-tuning MLSM models could not reach the performance of alternatively pre-trained models. This can be attributed to the fact that the alternative pre-training objectives aim at predicting *actual word forms* (as opposed to semantic latents) which provides them increased transferability to such a task, where the goal is to decide on the linguistic acceptability of texts.

Indeed, linguistic acceptability is less related to semantics, and more about syntax and actual word forms, e.g., if a masked word was originally in its singular form, replacing it by its – semantically highly related – plural form will result in an agrammatic sentence. As an attempt to mitigate this shortcoming of MLSM, we additionally implemented a multi-task learning (MTL) approach, during which the MLSM and MLM objectives are jointly taken into account with hyperparameter κ controlling the extent to which the MLM objective is considered in the final loss term, i.e.,

$$\mathcal{L}_{MTL} = \mathcal{L}_{MLSM} + \kappa \mathcal{L}_{MLM}.$$

To this end, we trained and evaluated 3 additional MTL models with $\kappa \in \{0.25, 0.5, 1\}$. MTL not only introduces an extra hyperparameter (κ) to account for, but we also need to sacrifice most of the efficiency improvements of MLSM pre-training, as unembedding has to be performed towards $|V| + k$ symbols (the additional $|V|$ being necessitated by the use of the MLM objective).

Based on our results included in Table 4 and Figure 5, we can conclude that the increased computational costs for MTL do not pay off. For all the investigated $\kappa > 0$ settings, we see worse performances compared to the $\kappa = 0$ case. Additionally, the differences are statistically significant according to the Wilcoxon signed rank test for all but the $\kappa = 1$ case. Even for that single case, the computational resources needed by the MTL model variant is substantially higher compared to the non-MTL ($\kappa = 0$) setting as discussed earlier. As such, we recommend against MTL pre-training.



(a) Overall results (b) CoNLL (c) COLA

(d) MNLIm (e) MNLImm (f) MRPC

(g) MRPC (h) QNLI (i) QQP

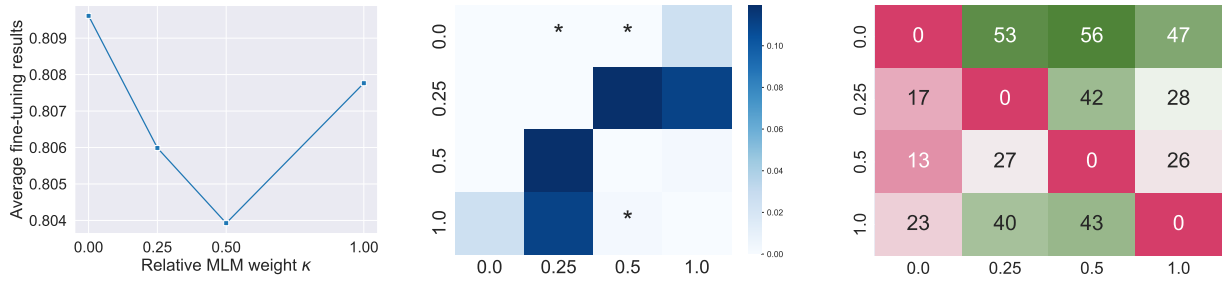
(j) QQP (k) RTE (l) SST2

(m) STSB (n) STSB (o) WiC

Figure 4: The average fine-tuning performance of differently pre-trained models as a function of their pre-training update steps.

Table 4: The detailed effects of using a multi-task learning pre-training with an MLM weight of κ.

	MLM weight κ			
task (metric)	0.0	0.25	0.5	1.0
CoNLL2003 (F1)	0.949	0.947	0.948	**0.951**
COLA (MCC)	**0.405**	0.382	0.392	0.399
MNLIm (accuracy)	**0.824**	0.822	0.822	0.819
MNLImm (accuracy)	**0.831**	**0.831**	0.827	0.828
MRPC (accuracy)	0.860	0.849	0.850	**0.867**
MRPC (F1)	0.902	0.895	0.896	**0.907**
QNLI (accuracy)	**0.902**	0.895	0.894	0.901
QQP (accuracy)	**0.903**	0.901	0.900	0.901
QQP (F1)	**0.868**	0.865	0.864	0.866
RTE (accuracy)	0.601	0.625	0.594	**0.613**
SST2 (accuracy)	**0.914**	0.908	0.912	0.912
STSB (pearson)	**0.859**	0.857	0.857	0.855
STSB (spearmanr)	**0.857**	0.855	0.854	0.853
WiC (accuracy)	**0.659**	0.649	0.644	0.637
Avg.	**0.810**	0.806	0.804	0.808



(a) Average fine-tuning performance (b) Wilcoxon p-values (c) Number of wins

Figure 5: The effects on the average fine-tuning performance of models jointly pre-trained with a combination of MLSM and MLM losses, κ controlling for the extent to which the MLM loss is considered.

4.3.1 Evaluation on BLiMP

There are arguably situations where the ability of a model to provide meaningful substitutes to a (masked) token position in an input sequence is beneficial. MLSM models, however, totally lack this capability, as they are not trained to reconstruct the exact identity of word forms, but they are capable of outputting the latent semantic properties of the tokens.

To this end, our final experiment investigates the extent to which we can predict the identity of actual masked tokens from the LSPs that are produced by an MLSM pre-trained model. Such an experiment also provides a way to assess the quality of the LSPs, i.e., in case we can accurately predict the actual masked word forms, then the LSPs likely encode useful semantic information.

As stated above, MLSM model cannot be meaningfully used for solving tasks that explicitly require predictions over the vocabulary of word forms of the model, such as it is the case with BLiMP (Warstadt et al., 2020), where models are evaluated by measuring the extent to which they assign higher pseudo log-likelihood (Salazar et al., 2020) score for linguistically appropriate token substitutions. Since the output space of MLSM is not tied to subword units, a direct evaluation of MLSM models would not be possible out of the box.

In order to still be able to evaluate MLSM pre-trained models for the BLiMP setting, we extended our model with a single linear layer on top of the outputs of the MLSM model, which can be viewed as a form of linear probing (LP). In this experiment, we froze all the model weights of the MLSM pre-trained models and added a single linear layer having $k \times |V|$ parameters for predicting token outputs based on the semantic latents produced by the MLSM model.

We kept the training of the single linear head deliberately short, i.e., we conducted 10,000 update steps altogether (whereas the MLSM model itself was trained for 10 times the update steps). The rational behind this was that if the LSPs learned during MLSM pre-training are useful and transfer to the task of masked language modeling, then a short training of the classifier head should suffice. Not only the training of the linear head took much shorter, but also most parts of the network did not require gradient computation, as we trained only a single linear layer of the network. As such, training the linear probe took no more than 3 GPU hours. Similar to our pre-training experiments, we created checkpoints at 10%, 25%, 50% and 100% readiness levels, meaning that our first linear probe required less than 20 minutes of extra compute.

Figure 6 contains our results on the BLiMP task for differently obtained models. We also include results of the jointly pre-trained MLSM+MLM model that corresponds to the kind of multi-task learning setting introduced in Section 4.3.1. We report results for the MTL model with $\kappa = 1$, as this is the MTL model that puts the largest emphasis on the MLM loss, something that is beneficial for a task like BLiMP.

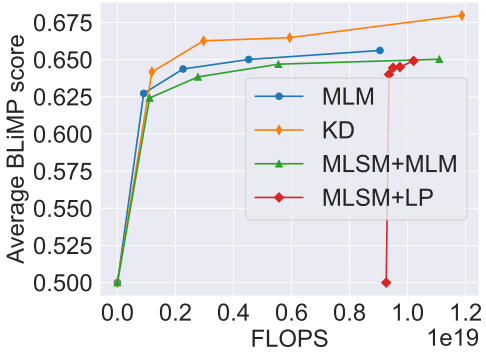


Figure 6: Average BLiMP evaluation results of the differently obtained model variants.

Figure 6 includes the average BLiMP performance as a function of the amount of compute that was required for obtaining a given model variant. When reporting results for the MLSM model with the linear probe (MLSM+LP), we also include the costs of pre-training the MLSM model for which the extra linear head was added. This is the reason why the first marker for the MLSM+LP curve is not located at 0.0, but at the amount of computation that corresponds to the pre-training costs of the MLSM model itself.

We can see from Figure 6 that we could extend the masked language modeling capabilities of our MLSM pre-trained model by using a negligible amount (less than 3 hours) of extra masked language modeling specific post-training. This extra training did not affect the previously analyzed capabilities of our model, as the backbone weights were frozen. The BLiMP performance of the MLSM+LP model is on par with the MLM and MLSM+MLM models and only 2.5 points away from that of KD (but the KD model on the other hand is not able to output latent semantic properties, something that the MLSM+LP model is still capable of).

5 Conclusions

Our goal in this paper was to advance the understanding of masked latent semantic modeling in multiple important aspects. To this end, we set up different research questions and conducted carefully designed experiments to answer them. Our experiments have revealed multiple previously unexplored characteristics of MLSM pre-training. Additionally, we have also verified via linear probing that the latent semantic features MLSM pre-trained models develop are meaningful towards performing actual masked word prediction task – something that MLSM pre-trained models are not capable at all otherwise.

References

Gustavo Aguilar, Yuan Ling, Yu Zhang, Benjamin Yao, Xing Fan, and Chenlei Guo. Knowledge distillation from internal representations. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(05): 7350–7357, Apr. 2020. doi: 10.1609/aaai.v34i05.6229. URL https://ojs.aaai.org/index.php/AAAI/article/view/6229.

Ahmed Alajrami and Nikolaos Aletras. How does the pre-training objective affect what large language models learn about linguistic properties? In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (eds.), *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 131–147, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-short.16. URL https://aclanthology.org/2022.acl-short.16.

He Bai, Tong Wang, Alessandro Sordoni, and Peng Shi. Better language model with hypernym class prediction. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (eds.), *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1352–1362, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-long.96. URL https://aclanthology.org/2022.acl-long.96/.

Luisa Bentivogli, Ido Dagan, Hoa Trang Dang, Danilo Giampiccolo, and Bernardo Magnini. The fifth pascal recognizing textual entailment challenge. In *In Proc Text Analysis Conference (TAC'09)*, 2009.

Gábor Berend. Sparsity makes sense: Word sense disambiguation using sparse contextualized word representations. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu (eds.), *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 8498–8508, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.683. URL https://aclanthology.org/2020.emnlp-main.683.

Gábor Berend. Masked latent semantic modeling: an efficient pre-training alternative to masked language modeling. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 13949–13962, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-acl.876. URL https://aclanthology.org/2023.findings-acl.876.

Nicolas Boizard, Hippolyte Gisserot-Boukhlef, Duarte M. Alves, André Martins, Ayoub Hammal, Caio Corro, Céline Hudelot, Emmanuel Malherbe, Etienne Malaboeuf, Fanny Jourdan, Gabriel Hautreux, João Alves, Kevin El-Haddad, Manuel Faysse, Maxime Peyrard, Nuno M. Guerreiro, Patrick Fernandes, Ricardo Rei, and Pierre Colombo. EuroBERT: Scaling multilingual encoders for european languages, 2025. URL https://arxiv.org/abs/2503.05500.

Daniel Cer, Mona Diab, Eneko Agirre, Iñigo Lopez-Gazpio, and Lucia Specia. SemEval-2017 task 1: Semantic textual similarity multilingual and crosslingual focused evaluation. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pp. 1–14, Vancouver, Canada, August 2017. Association for Computational Linguistics. doi: 10.18653/v1/S17-2001. URL https://aclanthology.org/S17-2001.

Ido Dagan, Oren Glickman, and Bernardo Magnini. The pascal recognising textual entailment challenge. In Joaquin Quiñonero-Candela, Ido Dagan, Bernardo Magnini, and Florence d'Alché Buc (eds.), *Machine Learning Challenges. Evaluating Predictive Uncertainty, Visual Object Classification, and Recognising Tectual Entailment*, pp. 177–190, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. ISBN 978-3-540-33428-6.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 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)*, pp. 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL https://aclanthology.org/N19-1423.

William B. Dolan and Chris Brockett. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the Third International Workshop on Paraphrasing (IWP2005)*, 2005. URL https://aclanthology.org/I05-5002.

Christiane Fellbaum. *WordNet: An Electronic Lexical Database*. Bradford Books, 1998.

Jonas Geiping and Tom Goldstein. Cramming: training a language model on a single gpu in one day. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org, 2023.

Danilo Giampiccolo, Bernardo Magnini, Ido Dagan, and Bill Dolan. The third PASCAL recognizing textual entailment challenge. In *Proceedings of the ACL-PASCAL Workshop on Textual Entailment and Paraphrasing*, pp. 1–9, Prague, June 2007. Association for Computational Linguistics. URL https://aclanthology.org/W07-1401.

R Bar Haim, Ido Dagan, Bill Dolan, Lisa Ferro, Danilo Giampiccolo, Bernardo Magnini, and Idan Szpektor. The second PASCAL recognising textual entailment challenge. In *Proceedings of the Second PASCAL Challenges Workshop on Recognising Textual Entailment*, volume 7, 2006.

Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. DeBERTa: Decoding-enhanced BERT with disentangled attention. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=XPZIaotutsD.

Zhengfu He, Wentao Shu, Xuyang Ge, Lingjie Chen, Junxuan Wang, Yunhua Zhou, Frances Liu, Qipeng Guo, Xuanjing Huang, Zuxuan Wu, et al. Llama scope: Extracting millions of features from llama-3.1-8b with sparse autoencoders. *arXiv preprint arXiv:2410.20526*, 2024.

Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. *CoRR*, abs/1503.02531, 2015. URL http://arxiv.org/abs/1503.02531.

Robert Huben, Hoagy Cunningham, Logan Riggs Smith, Aidan Ewart, and Lee Sharkey. Sparse autoencoders find highly interpretable features in language models. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=F76bwRSLeK.

Shankar Iyer, Nikhil Dandekar, and Kornél Csernai. First quora dataset release: Question pairs, 2017. URL https://data.quora.com/First-Quora-Dataset-Release-Question-Pairs.

Peter Izsak, Moshe Berchansky, and Omer Levy. How to train BERT with an academic budget. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (eds.), *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 10644–10652, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.831. URL https://aclanthology.org/2021.emnlp-main.831/.

Yoav Levine, Barak Lenz, Or Dagan, Ori Ram, Dan Padnos, Or Sharir, Shai Shalev-Shwartz, Amnon Shashua, and Yoav Shoham. SenseBERT: Driving some sense into BERT. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault (eds.), *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 4656–4667, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.423. URL https://aclanthology.org/2020.acl-main.423.

Tom Lieberum, Senthooran Rajamanoharan, Arthur Conmy, Lewis Smith, Nicolas Sonnerat, Vikrant Varma, Janos Kramar, Anca Dragan, Rohin Shah, and Neel Nanda. Gemma scope: Open sparse autoencoders everywhere all at once on gemma 2. In Yonatan Belinkov, Najoung Kim, Jaap Jumelet, Hosein Mohebbi, Aaron Mueller, and Hanjie Chen (eds.), *Proceedings of the 7th BlackboxNLP Workshop: Analyzing and Interpreting Neural Networks for NLP*, pp. 278–300, Miami, Florida, US, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.blackboxnlp-1.19. URL https://aclanthology.org/2024.blackboxnlp-1.19/.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

Julien Mairal, Francis Bach, Jean Ponce, and Guillermo Sapiro. Online dictionary learning for sparse coding. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pp. 689–696, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-516-1. doi: 10.1145/1553374.1553463. URL http://doi.acm.org/10.1145/1553374.1553463.

Mohammad Taher Pilehvar and Jose Camacho-Collados. WiC: the word-in-context dataset for evaluating context-sensitive meaning representations. 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)*, pp. 1267–1273, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1128. URL https://aclanthology.org/N19-1128.

Jacob Portes, Alexander Trott, Sam Havens, DANIEL KING, Abhinav Venigalla, Moin Nadeem, Nikhil Sardana, Daya Khudia, and Jonathan Frankle. Mosaicbert: A bidirectional encoder optimized for fast pretraining. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 3106–3130. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/095a6917768712b7ccc61acbeecad1d8-Paper-Conference.pdf.

Sampo Pyysalo, Jenna Kanerva, Antti Virtanen, and Filip Ginter. WikiBERT models: Deep transfer learning for many languages. In *Proceedings of the 23rd Nordic Conference on Computational Linguistics (NoDaLiDa)*, pp. 1–10, Reykjavik, Iceland (Online), May 31–2 June 2021. Linköping University Electronic Press, Sweden. URL https://aclanthology.org/2021.nodalida-main.1.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 2383–2392, Austin, Texas, November 2016. Association for Computational Linguistics. doi: 10.18653/v1/D16-1264. URL https://aclanthology.org/D16-1264.

Julian Salazar, Davis Liang, Toan Q. Nguyen, and Katrin Kirchhoff. Masked language model scoring. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault (eds.), *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 2699–2712, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.240. URL https://aclanthology.org/2020.acl-main.240.

Chen Shani, Jilles Vreeken, and Dafna Shahaf. Towards concept-aware large language models. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 13158–13170, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-emnlp.877. URL https://aclanthology.org/2023.findings-emnlp.877.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pp. 1631–1642, Seattle, Washington, USA, October 2013. Association for Computational Linguistics. URL https://aclanthology.org/D13-1170.

Robyn Speer, Joshua Chin, and Catherine Havasi. Conceptnet 5.5: An open multilingual graph of general knowledge. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, AAAI'17, pp. 4444–4451. AAAI Press, 2017.

Erik F. Tjong Kim Sang and Fien De Meulder. Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*, pp. 142–147, 2003. URL https://www.aclweb.org/anthology/W03-0419.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. Superglue: A stickier benchmark for general-purpose language understanding systems. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019a. URL https://proceedings.neurips.cc/paper/2019/file/4496bf24afe7fab6f046bf4923da8de6-Paper.pdf.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. 2019b. In the Proceedings of ICLR.

Benjamin Warner, Antoine Chaffin, Benjamin Clavié, Orion Weller, Oskar Hallström, Said Taghadouini, Alexis Gallagher, Raja Biswas, Faisal Ladhak, Tom Aarsen, Nathan Cooper, Griffin Adams, Jeremy Howard, and Iacopo Poli. Smarter, better, faster, longer: A modern bidirectional encoder for fast, memory efficient, and long context finetuning and inference, 2024. URL https://arxiv.org/abs/2412.13663.

Alex Warstadt, Amanpreet Singh, and Samuel R. Bowman. Neural network acceptability judgments. *Transactions of the Association for Computational Linguistics*, 7:625–641, 2019. doi: 10.1162/tacl_a_00290. URL https://aclanthology.org/Q19-1040.

Alex Warstadt, Alicia Parrish, Haokun Liu, Anhad Mohananey, Wei Peng, Sheng-Fu Wang, and Samuel R. Bowman. BLiMP: The benchmark of linguistic minimal pairs for English. *Transactions of the Association for Computational Linguistics*, 8:377–392, 2020. doi: 10.1162/tacl_a_00321. URL https://aclanthology.org/2020.tacl-1.25.

Adina Williams, Nikita Nangia, and Samuel Bowman. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pp. 1112–1122. Association for Computational Linguistics, 2018. URL http://aclweb.org/anthology/N18-1101.

Atsuki Yamaguchi, George Chrysostomou, Katerina Margatina, and Nikolaos Aletras. Frustratingly simple pretraining alternatives to masked language modeling. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (eds.), *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 3116–3125, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.249. URL https://aclanthology.org/2021.emnlp-main.249.

Zeyu Yun, Yubei Chen, Bruno Olshausen, and Yann LeCun. Transformer visualization via dictionary learning: contextualized embedding as a linear superposition of transformer factors. In Eneko Agirre, Marianna Apidianaki, and Ivan Vulić (eds.), *Proceedings of Deep Learning Inside Out (DeeLIO): The 2nd Workshop on Knowledge Extraction and Integration for Deep Learning Architectures*, pp. 1–10, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.deelio-1.1. URL https://aclanthology.org/2021.deelio-1.1.

Mengjie Zhao, Philipp Dufter, Yadollah Yaghoobzadeh, and Hinrich Schütze. Quantifying the contextualization of word representations with semantic class probing. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pp. 1219–1234, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.findings-emnlp.109. URL https://aclanthology.org/2020.findings-emnlp.109.

A The detailed pairwise comparison of the individual fine-tuning experiments

In Figure 7, we detail the results of the individual fine-tuning experiments with the differently pre-trained models at their different readiness levels. Each scatter plot refers to a different evaluation scenario, whereas the bar chart provides an aggregate overview on the proportion of those fine-tuning experiments for which the MLSM pre-trained model performed better than the alternatively pre-trained model variants at the same pre-training readiness level.

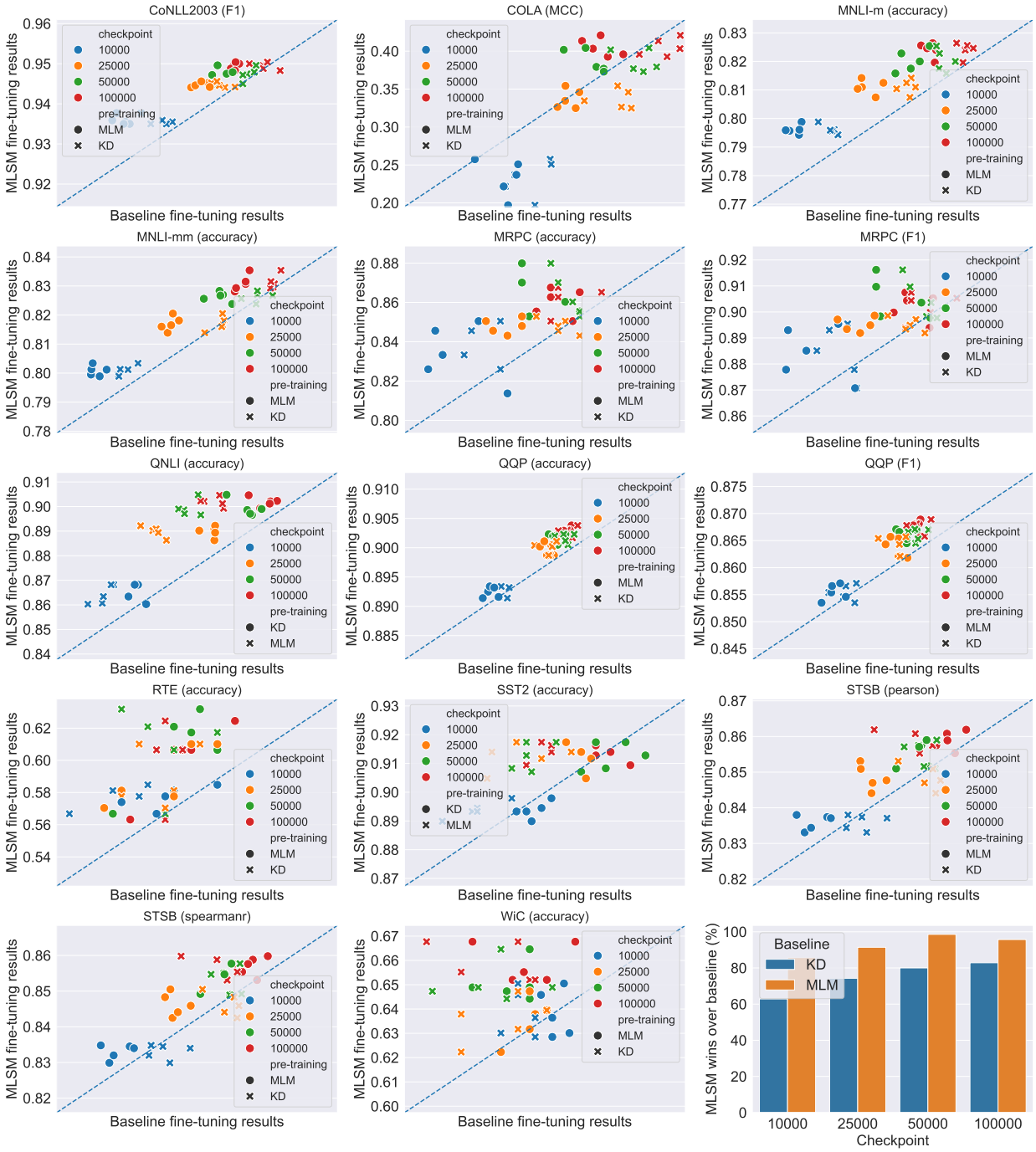


Figure 7: Scatterplot of paired fine-tuning experiments initialized with the same classification head. Colors indicate the checkpoint of the investigated models. Markers with circle and cross shapes are used to differentiate between comparisons against MLM and KD pre-trained models, respectively. Markers above the dashed diagonal line indicate fine-tuning experiments when the MLSM pre-trained model performed better than the alternatively pre-trained model variant at a given readiness level. The bar chart indicates the fraction of experiments such that the fine-tuning performance of the MLSM pre-trained model outperformed that of the alternatively pre-trained model variant for the different checkpoints.