ProtTrans: Towards Cracking the Language of Life's Code Through Self-Supervised Deep Learning and High Performance Computing

Ahmed Elnaggar[1,*,**], Michael Heinzinger[1,*], Christian Dallago[1], Ghalia Rihawi[1], Yu Wang[2], Llion Jones[3], Tom Gibbs[4], Tamas Feher[4], Christoph Angerer[4], Debsindhu Bhowmik[5], and Burkhard Rost[1]

[1]TUM (Technical University of Munich) Department of Informatics, Bioinformatics & Computational Biology - i12, Boltzmannstr. 3, 85748 Garching/Munich, Germany
[2]Med AI Technology (Wu Xi) Ltd. , Ma Shan, Mei Liang Road, 88, 2nd floor (west), Bin Hu District, Wu Xi, Jiang Su Province, China
[3]Google AI, Google, 1600 Amphitheatre Parkway, Mountain View, CA 94043, USA
[4]NVIDIA, 2788 San Tomas Expy, Santa Clara, CA 95051, Vereinigte Staaten, USA
[5]Oak Ridge National Laboratory (ORNL), 1 Bethel Valley Rd, Oak Ridge, TN 37830, Vereinigte Staaten
[*]These authors contributed equally to this work.
[**]Corresponding author: ahmed.elnaggar [at] tum.de, tel: +49-289-17-811 (email rost: assistant [@] rostlab.org)
[***]The official GitHub repository: https://github.com/agemagician/ProtTrans

Abstract

Motivation: Natural Language Processing (NLP) continues improving substantially through auto-regressive (AR) and auto-encoding (AE) Language Models (LMs). These LMs require expensive computing resources for self-supervised or un-supervised learning from huge unlabelled text corpora. The information learned is transferred through so-called *embeddings* to downstream prediction tasks. Computational biology and bioinformatics provide vast gold-mines of structured and sequentially ordered text data leading to extraordinarily successful protein sequence LMs that promise new frontiers for generative and predictive tasks at low inference cost. As recent NLP advances link corpus size to model size and accuracy, we addressed two questions: (1) To which extent can High-Performance Computing (HPC) up-scale protein LMs to larger databases and larger models? (2) To which extent can LMs extract features from single proteins to get closer to the performance of methods using evolutionary information?

Methodology: Here, we trained two auto-regressive language models (Transformer-XL and XLNet) and two auto-encoder models (BERT and Albert) on 80 billion amino acids from 200 million protein sequences (UniRef100) and one language model (Transformer-XL) on 393 billion amino acids from 2.1 billion protein sequences taken from the Big Fat Database (BFD), today's largest set of protein sequences (corresponding to 22- and 112-times, respectively of the entire English Wikipedia). The LMs were trained on the Summit supercomputer, using 936 nodes with 6 GPUs each (in total 5616 GPUs) and one TPU Pod, using V3-512 cores.

Results: We validated the feasibility of training big LMs on proteins and the advantage of up-scaling LMs to larger models supported by more data. The latter was assessed by predicting secondary structure in three- and eight-states (Q3=75-83, Q8=63-72), localization for 10 cellular compartments (Q10=74) and whether a

Preprint. Under review.

protein is membrane-bound or water-soluble (Q2=89). Dimensionality reduction revealed that the LM-*embeddings* from unlabelled data (only protein sequences) captured important biophysical properties of the protein alphabet, namely the amino acids, and their well orchestrated interplay in governing the shape of proteins. In the analogy of NLP, this implied having learned some of the grammar of the language of life realized in protein sequences. The successful up-scaling of protein LMs through HPC slightly reduced the gap between models trained on evolutionary information and LMs. Additionally, our results highlighted the importance of bi-directionality when processing proteins as the uni-directional TransformerXL was outperformed by its bi-directional counterparts.

1 Introduction

High-Performance Computing (HPC) has recently been advancing hand-in-hand with *Deep Learning* (DL) to achieve new scientific breakthroughs in both fields. More powerful supercomputers [1, 2] and advanced libraries [3, 4, 5, 6, 7] enable the training of ever more complex models on bigger data sets using advanced processing units such as Graphics Processing Units (GPUs) and Tensor Processing Units (TPUs) at increasing speeds and efficiency. HPC hardware is advancing both through infrastructure of supercomputers, such as Fugaku [8], Summit [1] or the SuperMUC-NG [9], and through its components, such as TPU pods [2], specifically designed to ease large scale neural network training for users. Concurrent software improvements in form of more efficient libraries such as Horovod [6] allow executing general purpose code on large distributed clusters with minor code changes.

Through contextualized Language Models (LMs) [10, 11], Natural Language Processing (NLP) has been benefiting more from advances in HPC than other fields. In particular *Transformers* [12] have reached state-of-the-art performance in several tasks including translation, summarization and question answering [13, 14]. LMs are trained on unlabelled data; this independence of expensive validated data opened vast sets of raw big data allowing to up-scale LMs in NLP by orders of magnitude. The self-supervised training exclusively relies upon the sequential order of the input. Two approaches make use of this information, namely *auto-regressive* (predict next token in a sequence, given all previous tokens) and *auto-encoding* (reconstruction of corrupted input) training. Once trained, LMs can extract features, referred to as *embeddings*, to use as input in subsequently trained supervised models (*transfer-learning*). This two-step training outsources the computationally expensive LM pre-training to the HPC infrastructure while the computationally simple inference can be done on commodity hardware.

Protein research provides an excellent use-case for transfer-learning as large amounts of exponentially growing but unlabelled data contrast much more limited sets with experimental annotations. One example for this is the "sequence-structure" gap [15], i.e. the gap between the number of proteins for which one-dimensional (**1D**) sequences are known and the orders of magnitude smaller subset of proteins for which their three-dimensional (**3D**) structures are known. Knowing these structures is crucial for understanding their function. Such understanding is needed, e.g. to possibly disrupt the binding of the spiky S1 protein of the SARS-Cov-2 virus that by binding to the human receptor ACE2 caused the COVID-19 pandemic. The sequence-structure and sequence-function gaps, or more generally the sequence-annotation gaps keep growing exponentially. Closing those gaps through prediction methods based on artificial intelligence (AI) is one of the crucial challenges for computational biology and bioinformatics.

Recently, the leap of NLP through advanced LMs have successfully been generalized toward understanding the language of life through advanced LMs trained on proteins [16, 17, 18, 19, 20, 21, 22, 23, 24]. The main concept behind these approaches is to interpret protein sequences as sentences and their constituent – amino acids – as single words. Protein sequences are constrained to adopt particular 3D shapes (referred to as protein *3D structure*) optimized for accomplishing particular functions. These constraints mirror the rules of grammar and meaning in natural language thereby allowing to map algorithms from NLP directly onto protein sequences. During training, the LM learns to extract those constraints from millions of examples and store the derived knowledge in its weights. While existing solutions in Protein Bioinformatics [25, 26, 27, 28, 29, 30] usually have to search for evolutionary related proteins in exponentially growing databases, LMs offer a potential alternative to this increasingly time-consuming database search as they extract features directly from

single protein sequences. On top, the performance of existing solutions deteriorates if not a sufficient number of related sequences can be found, e.g. the quality of predicted protein structures correlates strongly with the number of effective sequences found in today's databases [31]. Additionally, some proteins are intrinsically hard to align (e.g. intrinsically disordered proteins [32] or proteins which do not have any related sequences (*dark proteome*, [33]).

In this work, we pursued two objectives. Firstly, we explored the limits of up-scaling language models trained on proteins as well as protein sequence databases used for training. Secondly, we compared the effects of auto-regressive and auto-encoding pre-training upon the success of the subsequent supervised training, and compared all LMs to existing state-of-the-art solutions using evolutionary information [34].

2 Methods

2.1 Data for Language Models (LMs)

In this work, we assessed the impact of database size on performance through two data sets: UniRef100 [35] (with 216M protein sequences) and BFD [36] (with 2,122M sequences). The latter merged all protein sequences available in UniProt [37] and proteins translated from multiple metagenomic sequencing projects, making it the largest collection of protein sequences available at the time of writing. The original BFD set contained several copies of identical sequences; only one of those was kept, resulting in a subset with 2.1 billion (2.1B) protein sequences (with >393B amino acids requiring 527GB of disk space as text); we dubbed this set as *BFD*. This compared to UniRef100 with 216M proteins (80B amino acids, 150GB disk space; Fig. 1a). Overall, BFD was about eight times larger than the largest data sets used previously [19]. Despite the 8-fold increase in data, the number of tokens increased only five-fold (Fig. 1b), because UniRef100 sequences were longer than those in BFD (1.6-fold). A similar trend held for disk storage (Fig. 1c). Translating LMs from NLP to proteins interprets amino acids as words. Thereby, protein databases contain several orders of magnitude more tokens than corpora used in NLP, e.g., Google's Billion Word data set [38] is one of the biggest for NLP with about 829 million tokens (words), i.e. about 500-times fewer than BFD with 393 billion tokens. Both UniRef100 and BFD were tokenized with a single space (indicating word-boundaries) between each token. Each protein sequence was stored on a separate line, with lines/proteins representing the equivalent of "sentences". Additionally, an empty line was inserted between each protein sequence in order to indicate the "end of a document" as some LMs such as Bert use consecutive sequences for an auxiliary task, i.e. next-sentence prediction, which was not used in this work. As a minor filtering step, all non-generic or unresolved amino acids (B, O, U, Z) were mapped to 'unknown' (X). After this pre-processing, Uniref100 required 150GB GB of storage, BFD 734 GB. For training ProtTXL, the data was transformed to pytorch tensors on the fly. For ProtBert and ProtAlbert, the data had to be pre-processed and stored as tensorflow records, raising the storage to 2.3TB and 22TB for UniRef100 and BFD, respectively. Given tensorflow records with terabytes, data sets had to be chunked into 6000 files for thousands of parallel workers. We also compared the amino acid frequencies between databases as shown in Fig. 1d in order to detect potential biases.

2.2 Data for supervised training

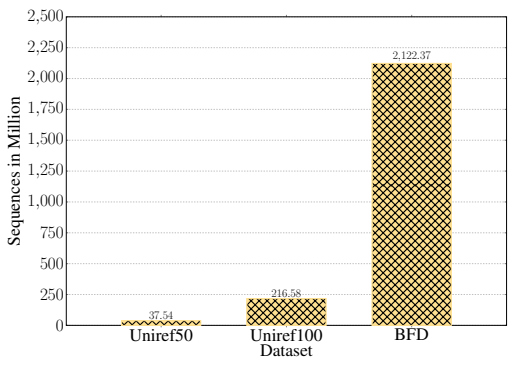
The information learnt by the LMs was condensed in form of *embeddings* which were compared quantitatively through their value for subsequent 2nd-step supervised training. Toward this end we used previously published data sets for ease of comparison to state-of-the-art methods based on evolutionary information and to methods extracting features through pre-trained LMs.

Per-residue prediction: When predicting properties on the level of single residues, the data set published alongside NetSurfP-2.0 [25] was used for 3- and 8-state secondary structure prediction. The NetSurfP-2.0 dataset was created through PISCES [39] selecting highest resolution protein structures (resolution <=2.5A) from the PDB [40]. The set was redundancy-reduced such that no pair of proteins had >25% pairwise sequence identity (PIDE), leaving 10791 proteins to train. About 500 proteins were randomly removed from this set and used as validation set to determine hyperparameters such as early stopping. The final performance was evaluated on three different data sets, each with <25% PIDE to the training set: CB513 (513 proteins; [41]), TS115 (115 proteins; [42]) and CASP12 (21 proteins; [43]).

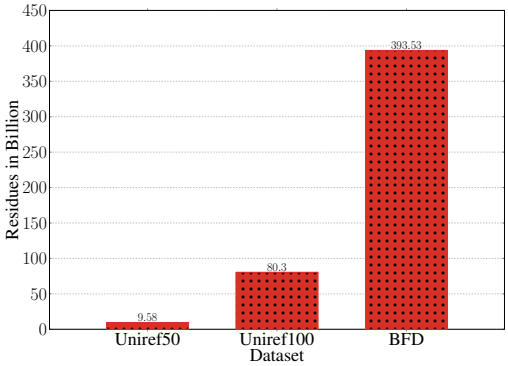
Per-protein prediction: For the prediction of features of entire proteins, the DeepLoc [26] data set was used to classify proteins into membrane-bound and water-soluble and for classifying proteins into ten classes of subcellular localization (also referred to as cellular compartments). This DeepLoc data set was created by pulling all proteins with experimentally annotated localization from UniProt (release: 2016_04). Proteins in this set were redundancy reduced at a level of PIDE<30% and split into 6621 proteins for training and 1841 for testing.

2.3 Data: unsupervised embeddings

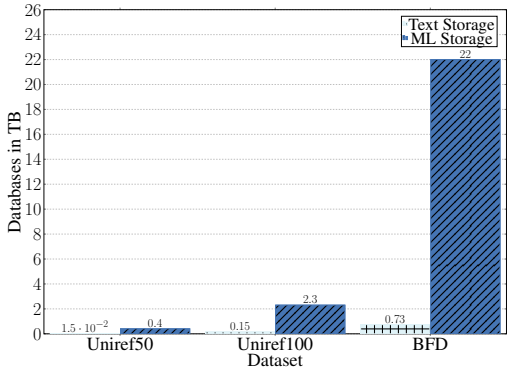
The embeddings extracted by the LMs were also evaluated visually by projecting the high-dimensional representations down to two dimensions using t-SNE [44]. A non-redundant (PIDE<40%) version of the SCOPe database [45] (release 2.07 with 14323 proteins) served as one way to interpret the t-SNE plots. For a subset of those proteins, we used experimentally annotated EC (Enzyme Commission [46]) numbers for functional classifications. Taxonomic identifiers from UniProt mapped proteins into one of the three major domains of life (*archaea*, *bacteria*, or *eukarya*) or to viruses (removing all proteins with missing classifications). The number of iterations for the t-SNE projections was set to 3000 and the perplexity to 30 for all plots with the exception of the amino acid plot for which we used a perplexity of 5.



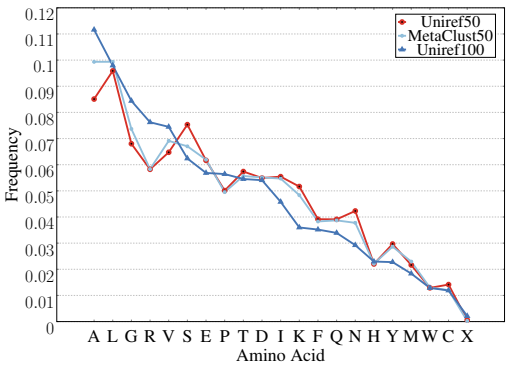
(a) Number of samples in each dataset



(b) Number of tokens in each dataset



(c) Dataset storage size



(d) Tokens frequency occurance in each dataset

Figure 1: **Large Scale Dataset Training:** here we compare the two datasets that were used in this study for language modelling (UniRef100, BFD) with a frequently used, redundancy reduced dataset (UniRef50). a) shows the number of sequences in each dataset in millions. (b) shows the number of residues/tokens in each dataset in billions. (c) shows size of each dataset raw text files as well as after converting to tensors in terabytes. (d) shows the frequency of each amino-acid/token in the each dataset

2.4 Models stage 1: LMs to extract embeddings

In this work, four LMs which achieved significant improvements in NLP (BERT [47], Albert [48], Transformer-XL [49] and XLNet [13]) were trained on protein sequences. Bert was the first bidirectional model in NLP which tried to reconstruct corrupted tokens, and is considered the de-facto standard for transfer learning in NLP. Albert reduced Bert's complexity by hard parameter sharing between its attention layers which allows to increase the number of attention heads (64 chosen here). Transformer-XL was chosen because it overcomes the problem of having a maximum sequence length, which was inherent to all previous Transformer based models (including Bert and Albert). With the average length of an English sentence around 15-30 words [50], an upper sentence length limit is no problem for sentence-level NLP tasks but many proteins are more than 10-times longer resulting in an average length of about 350 residues (*residues* is the term used to describe amino acids joined in a protein sequence, i.e. the *sentence length* measured in number of words). For example, around 20% of the sequences in UniRef100 (216M sequences) are longer than 510. Transformer-XL still cuts sequences into fragments but allows for flow of information between fragments for longer proteins by re-using hidden states of fragments which have already been processed. This memory is uni-directional as fragments are processed sequentially. XLNet uses the memory mechanism introduced by Transformer-XL to also allow for processing of sequences of arbitrary length. While the memory remains uni-directional for both, Transformer-XL and XLNet, only XLNet allows to gather bidirectional context within one memory fragment while Transformer-XL has only access to uni-directional context.

All these models were trained on UniRef100 and Transformer-XL was additionally trained on BFD (Table 1 for model parameters). Largely, we used configurations successfully transferred from NLP to protein sequences [21, 24, 51], with the exception of the number of layers that was increased to optimize memory utilization. Bert, TransformerXL and XLNet were trained with a hidden layer size (dimensionality of the features which can be extracted) of 1024 while Albert was trained with a hidden layer size of 4096. Models which use positional encoding like Bert and Albert, can process only sequences shorter or equal to the length of the positional encoding which has to be set before training. Setting the length of the positional encoding to 40k allowed the models to process protein sequences up to a length of 40k. Albert, Bert and Transformer-XL were optimized using the Lamb optimizer [52] designed for large batch sizes, while XLNet was optimized using Adam. No auxiliary tasks like Bert's next-sentence prediction were used for any model described here.

ProtTXL: The Transformer-XL versions trained here on protein sequences are referred to as to *ProtTXL* (only *ProtTXL* when trained on UniRef100 and *ProtTXL-BFD* when trained on BFD). Both LMs were trained with the configuration shown in Table 1, sharing a dropout rate of 15%, a memory length of 512 tokens and using mixed precision . The number of layers, number of heads, batch size, learning rate, weight decay, training steps and warm-up steps were adjusted according to training set size as well as GPU utilization. We focused especially on the complex interplay between learning rate and the number of warm-up steps which was shown to be crucial to prevent deeper layers of creating instability during training [53] and speed-up model convergence [54]. Here, the number of warm-up steps was set to cover at least one epoch for each data set. We tested initial learning rates between 0.001 and 0.005 which were increased linearly at every training step over the warm-up period. To avoid model divergence during training, the learning rate had to be (i) reduced along with the warm-up steps (for BFD), or (ii) increased for both (for Uniref100). Even after increasing the warm-up steps to two epochs, the maximum learning rate remained at 0.0025 for both data sets. Beyond this point, the training diverged. Using weight decay to regularize the network increased the GPU memory usage as it required to compute the norm of all weight vectors on our models, thus reducing the batch size. ProtTXL-BFD was trained for 40k steps in total, with 13.6k warm-up steps using a learning rate of 0.0005, while ProtTXL was trained for 31k steps with 5k warm-up steps using a learning rate of 0.002. The Lamb optimizer was able to handle the resulting batch sizes of 44k and 22k for ProtTXL-BFD and ProtTXL, respectively, without divergence.

ProtBert: For simplicity, we referred to the Bert model trained on UniRef100 as to *ProtBert*. We used the configuration proposed by the original publication (Table 1). The number of layers was increased in order to potentially reach better performance in supervised downstream tasks, while keeping inference time as well as GPU memory consumption at a reasonable level. Unlike Transformer-XL which was trained on Nvidia GPUs, mixed-precision was not used to train other

Hyperparameter	ProtTXL		ProtBert	ProtXLNet	ProtAlbert
Dataset	BFD100	Uniref100	Uniref100	Uniref100	Uniref100
Number of Layers	32	30	30	30	12
Hidden Layers Size	1024		1024	1024	4096
Hidden Layers Intermediate Size	4096		4096	4096	16384
Number of Heads	14	16	16	16	64
Positional Encoding Limits	-		40K	-	40K
Dropout	0.15		0.0	0.1	0.0
Target Length	512		512/2048	512	512/2048
Memory Length	512		-	384	-
Masking Probability	-		15%	-	15%
Local Batch Size	8	5	30/5	2	21/2
Global Batch Size	44928	22464	15360/2560	1024	10752/1024
Optimizer	Lamb		Lamb	Adam	Lamb
Learning Rate	0.0005	0.002	0.002	0.00001	0.002
Weight Decay	0.0	0.01	0.01	0.01	0.01
Training Steps	40.7K	31.3K	300K/100K	847K	150K/150K
Warm-up Steps	13.6K	5.5K	40K/0K	20K	40K/5K
Mixed Precision	FP16 Model Weight Fp32 Master Weight		None	None	None
Number of Parameters	562M	567M	420M	409M	224M
System	Summit	Summit	TPU Pod	TPU Pod	TPU Pod
Number of Nodes	936		64	64	64
Number of GPUs/TPUs	5616		512	512	512

Table 1: **Large Scale Deep Learning Training:** the table shows the configurations used for training the protein language models introduced here (ProtTXL, ProtBert, ProtXLNet, ProtAlbert) using either Summit or a TPU Pod v3.

models because those were trained on TPUs. Similar to the Bert version trained in the Lamb paper [52], ProtBert was first trained for 300k steps on sequences with a maximum length of 512 and then for another 100k steps on sequences with a length of a maximum length of 2k. This allows the model to first extract useful features from shorter sequences while using a bigger batch size, which makes training on longer sequences and thus overall training more efficient.

ProtAlbert: We referred to Albert trained on UniRef100 as to *ProtAlbert*. We used the configuration from the official GitHub repository for Albert (version: xxlarge v2) with 12 attention layers. For Albert the number of layers is increased through the number of times that Albert stacks its single layer. Compared to the original publication, we were able to increase the global batch size from 4096 to 10752 despite using the same hardware. The reason for this counter-intuitive effect is the reduced vocabulary size in protein sequences because the entire diversity of the protein universe is mapped to 20 different amino acids, compared to tens of thousands of different words. As ProtAlbert was also trained on TPUs, no mixed-precision was used for training. Similar to ProtBert, ProtAlbert was first trained for 150k steps on sequences with a maximum length of 512 and then for another 150k steps on sequences with a maximum length of 2k.

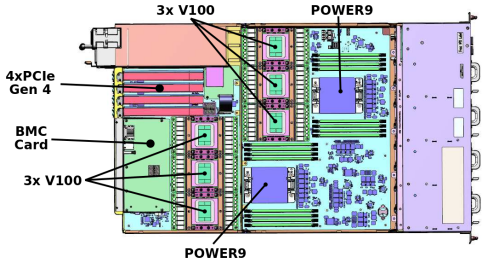
ProtXLNet: XLNet was trained on UniRef100 (*ProtXLNet*) using the original NLP configuration [13] (Table 1) except for the number of layers that was increased to 30 layers which reduced the global batch size to 1024. Due to the relatively small batch-size, we used the original optimizer: Adam with a learning rate of 0.00001. The model was trained through more steps, i.e. 20k warm-up and 847k steps to compensate for the smaller batch-size of this model.

2.5 Models stage 2: supervised models using embeddings

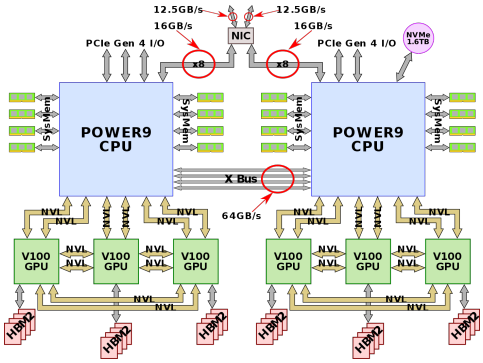
The second-stage supervised models using the embeddings from the LMs as input were deliberately kept relatively minimal to focus the differential analysis on the power of the LM embeddings. All our experiments used the pre-trained LMs as feature extractors without fine-tuning, i.e. without gradient back-propagating to the LMs. Thereby, we could proxy the information contained in the embeddings through the performance of the supervised tasks. The supervised models have been described before [17]. To briefly summarize: we applied tasks on two different levels, namely per-

residue and per-protein predictions. For the **per-residue prediction** a simple two-layer convolutional neural network (CNN) was trained on the embeddings. The first layer of our CNN compressed the output of the language models down to 32 dimensions using a window size of 7 (1024 for ProtBert, ProtTXL and ProtXLNet, 4096 for ProtAlbert). The compressed representation was fed to two different CNNs each having again a window size of 7. One of these CNNs was trained on predicting secondary structure in 3-states, the other was trained on predicting 8-states. The network was trained on both outputs simultaneously by adding their losses (multi-task learning). For the **per-protein prediction** features were also extracted from the last layer of the LMs. However, for this task the representations were averaged (mean-pooled) over the length-dimension of the protein resulting in a fixed-size representation for all proteins. The resulting vector (1024-dimensional for ProtBert and ProtTXL, 4096-dimensional for ProtAlbert) was used as an input to a single feed forward layer with 32 neurons which compressed information before making the final predictions for both per-protein tasks simultaneously (multi-task learning).

2.6 Hardware



(a) Summit Single Node Overview



(b) Summit Node Internal Architecture

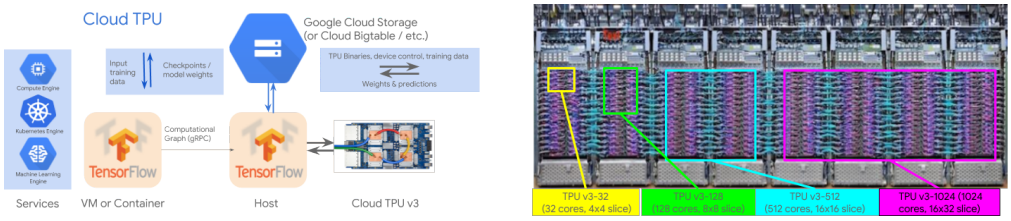
Figure 2: **Summit Architecture:** Panel (a) shows a single node of the Summit super computer consisting of two power9 CPUs and 6 V100 GPUs while (b) shows how the CPUs are connected with the GPUs including the connection speed between them.

ORNL Summit & Rhea: The Oak Ridge National Laboratory (ORNL) provides several clusters for researchers who need computational resources not provided by research facilities such as universities. Here, we used *Summit* and *Rhea*. Summit was used to train the deep learning models, while Rhea was used for the pre-processing of data sets including the distributed generation of tensorflow records.

Summit is the world's second fastest computer, consisting of approximately 4618 nodes. Each node has two IBM POWER9 processors and six NVIDIA Volta V100 with 16GB of memory each (Figure 2[1]). Every POWER9 processor is connected via dual NVLINK bricks, each capable of a 25GB/s transfer rate in both directions. A single node has 0.5 TB of DDR4 main memory and 1.6TB of non-volatile memory that can be used as a burst buffer. Summit is divided into racks with each rack having 18 nodes. In all of our experiments we reserved 936 nodes for training. As having nodes on the same rack decreases the communication overhead, we reserved entire racks. By using 936 nodes with 5616 GPUs, each LM trained in about two days.

The smaller cluster (Rhea) contains two partitions: Rhea and GPU. The Rhea partition has 512 node, each with 128 GB of memory and two Intel® Xeon® E5-2650. The GPU partition has only 9 nodes, each with 1 TB of memory and two Intel® Xeon® E5-2695. Reha reduced the time needed for creating tensorflow records for the BFD dataset from 7.5 months (!) to fewer than two days, by converting the original sequential script to distributed processing using MPI. The generation script used two nodes of the GPU partition, with a total of 112 parallel threads.

Google TPU Pod: In 2016, Google introduced tensor processing unit (TPU) as its application-specific integrated circuit optimized for training neural networks. TPUs can be accessed through Google Cloud. Training the protein LMs used the latest TPU generation (V3) with 512 cores. These cores are divided into hosts with each host having access to 8 cores. Consequently, we had access to 64 hosts, and each core had 16 GiB of high-bandwidth memory. Training on the TPUs required



(a) TPU V3 Training Pipeline Overview

(b) TPU Pod V3 Slices Overview

Figure 3: **TPU Training:** The figures show (a) an overview of the training pipeline for a single TPU V3-8 and (b) the difference of available TPU Pod v3 configuration.

access to a virtual machine on Google Cloud and storage on Google Bucket [55]. The workflow as well as the different scales of TPUs are depicted in Fig. 3. With Google TPU V3-512, ProtBert trained in about 9.5 days (completing 400K training steps), ProtAlbert trained in about 15.5 days (completing 300K training steps), and ProtXLNet in about 11 days (completing 847k training steps).

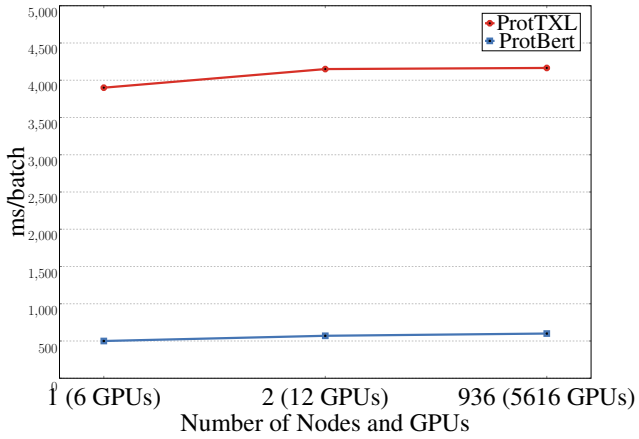
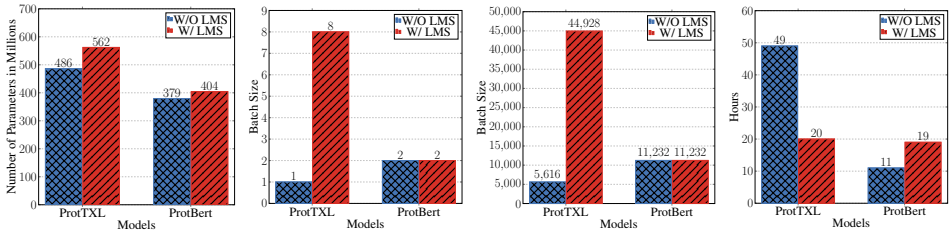


Figure 4: **Large Scale Dataset Training:** The figure shows the overhead of increasing the number of nodes/gpus for both ProtTXL and ProtBert. The overhead increases slightly from 1 to 2 nodes but remains constant even when scaling up to 936 nodes with a total of 5616 GPUs.



(a) Model parameters with and without LMS

(b) Local Batch Size with and without LMS

(c) Global Batch Size with and without LMS

(d) Hours per epoch with and without LMS

Figure 5: **Large Scale Deep Learning Training:** The figures show the effect of using large model support (LMS) on both, model size as well as batch size, when we tested ProtTXL or ProtBert on Nvidia V-100 16GB GPUs. It highlights the difference between applying LMS inside PyTorch (ProtTXL) or tensorflow (ProtBert). Panel (a) shows the effect of LMS on the maximum model size that can fit in the memory of a single V-100 when LMS is enabled. Panels (b,c) focus on the effect of LMS on the maximum local (b) and global batch size (c) that can fit in the GPU. The number of hours required to finish a single epoch using 936 nodes, each with 6 GPUs when LMS being enabled is shown in (d).

2.7 Software

Summit integrates several pre-configured modules which include the most popular libraries and tools required for simulation, deep learning, distributed training and other purposes. We used the IBM Watson Machine Learning module versions 1.6.0 and 1.6.2 for our deep learning training. In contrast to this, the Google Cloud server, which we used for the TPU Pod training, had to be configured manually because only the operating system was installed.

Pytorch was used to train ProtTXL, tensorflow to train ProtBert, ProtAlbert and ProtXLNet. Both libraries used the Horovod framework [6] to train the models on distributed clusters such as Summit. Horovod supports distributed GPU training with minimal change in the code. It supports different backends including MPI, NCCL and IBM PowerAI distributed deep learning (DDL). We tested all three backends and found DDL to be the fastest for our training purpose on Summit. The time needed to finish a single batch with ProtTXL-BFD increased from one to two nodes due to the communication overhead (Fig. 4). After two nodes the communication overhead plateaued, even when scaling up to 936 nodes with 5616 GPUs. Summit has integrated DDL in their Watson Machine Learning module which comes with most DDL libraries including pytorch, tensorflow, apex, DDL and horovod. However, Summit has only a license for using DDL up to 954 nodes. Contrary to Summit, training on TPU Pods did not require any changes in the Tensorflow code to use either a single TPU host or to distribute workload among multiple TPU hosts.

Mixed precision allows to fit bigger models and batch sizes into GPU memory by using 16-bit precision only or a mix of 16-bit and 32-bit precision. Nvidia's APEX library [56] was used for mixed precision training of ProtTXL, due to its pytorch support. As ProtTXL training became instable when training with 16 Bit precision, we switched to almost half precision training (storing all model weights at 16 Bit precision; exception: batch-normalization layers), while keeping a master copy of the model's weights in 32 Bit. We did not use mixed-precision for models trained on TPUs.

Another optimization technique/library crucial for our training on Summit was IBM's large model support (LMS) [57]. Similar to gradient checkpointing [58], LMS virtually extends the GPU memory by outsourcing parts of the model from GPU to main memory. This allows training models larger than the GPU memory. The obvious drawback of LMS is the increase in training time due to shuttling data between CPU and GPU and back. However, the reduced memory consumption of the model allows to increase the batch size, potentially compensating for the communication overhead. Compared to gradient checkpointing, LMS provides easier integration into existing code by operating directly on a computational graph defined by users and automatically adds swap-in and swap-out nodes for transferring tensors from GPU memory to main memory and vice versa. We have tested LMS on ProtTXL as well as ProtBert (Figure 4). As Pytorch and tensorflow have different strategies to integrate LMS, we also compared the effect of LMS on batch-size, model size and training time using the two different libraries. ProtTXL was used to evaluate the effect of Pytorch's implementation of LMS while ProtBert was trained for a few steps BFD using Summit to evaluate tensorflow's implementation of LMS. Training ProtBert for a few steps was sufficient to assess the effect of LMS on batch-size, model size as well as an estimate of training time. In the end, we used LMS only for ProtTXL to strike a balance between model size and training time. The number of LM parameters could be increased by about 15.6% for ProtTXL-BFD and to 6.6% for ProtBert (5a). Additionally, we could increase the batch size by 700% for ProtTXL-BFD (Figures 5b and 5c). The NV-Link between CPU and GPU on Summit-nodes, reduced the training time for ProtTXL by 60%while it increased by 72% for ProtBert (Figure 5d).

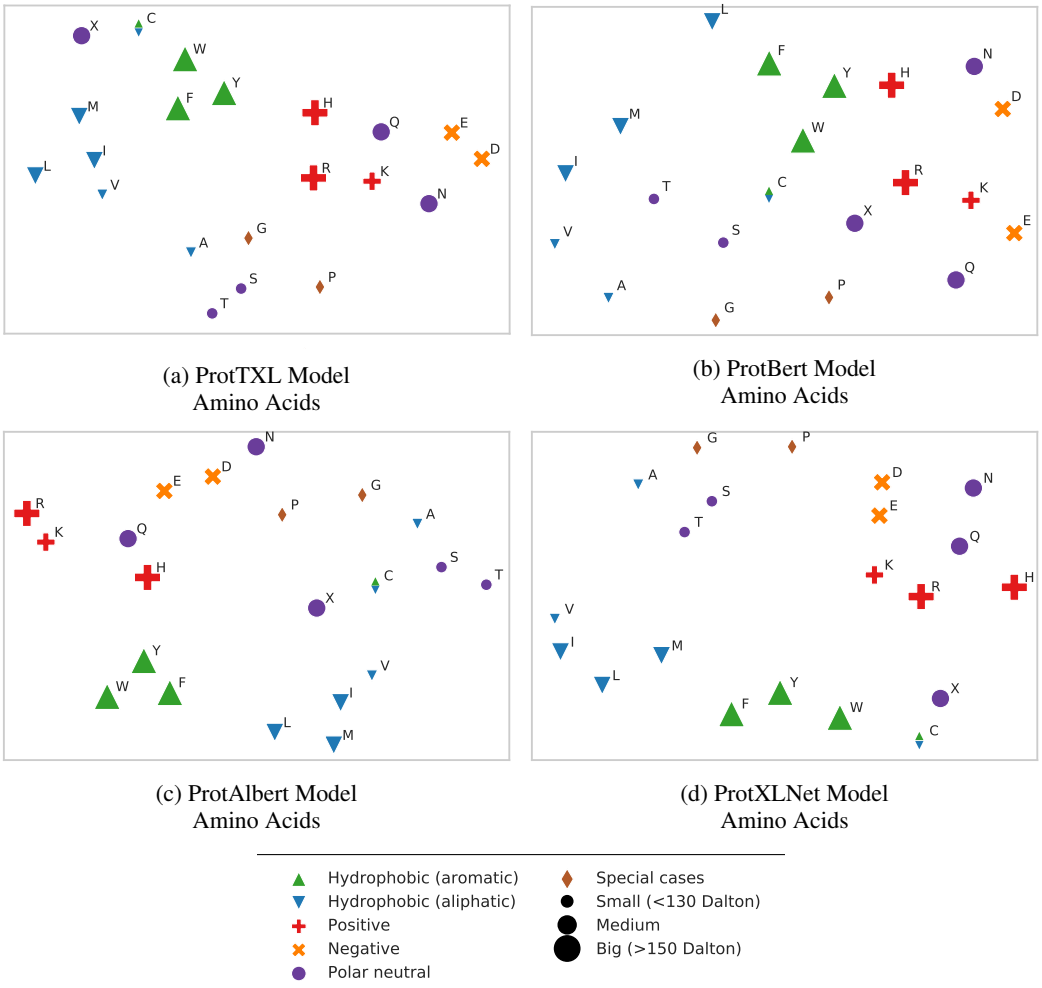
3 Results

3.1 Unsupervised learning: embeddings from LMs informative

The embeddings extract some of the information learned by the LMs in the first stage of unsupervised learning. To establish that our protein LMs have extracted an understanding akin to the grammar in NLP, we projected the high-dimensional embedding space down to two dimensions using t-SNE [44] and visualized proteins according to annotated structural, functional or evolutionary information.

Capturing biophysical features of amino acids. Applying t-SNE to the first embedding layer visualized information extracted by the LMs representing individual amino acids irrespective of their surrounding context (residues next to it). As previously established for another protein LM [24], the

t-SNE projections (Fig. 6) suggested that all LMs captured essential biophysical aspects of amino acids. These included charge, polarity, amino acid size (small amino acids A, C, G, P, S, T separated from large F, H, R, W, Y), hydrophobicity, even to the level of aliphatic (A, I, L, M, V) vs. aromatic (W, F, Y).



(a) ProtTXL Model Amino Acids

(b) ProtBert Model Amino Acids

(c) ProtAlbert Model Amino Acids

(d) ProtXLNet Model Amino Acids

Figure 6: **2D t-SNE projections of uncontextualized token embeddings for single amino acids:** all models learnt to cluster the 20 standard amino acids according to their biochemical and biophysical properties, i.e. hydrophobicity, charge and size. For example, the mostly hydrophobic and polar role of Cysteine (C) is conserved.

Capturing protein structure classes. To assess which aspects of protein structure were captured through the self-supervised LMs, we averaged over the length-dimension of the representations derived from the last layer of each model. This created fixed-size representations for each protein. We applied this encoding to the SCOPe database [45] which classifies proteins according to their 3D structures (Methods). On the most coarse-grained level, SCOPe distinguishes between all-alpha, all-beta, alphalbeta, alpha&beta, multi-domain, membrane/cell surface and small proteins. ProtTXL and ProtBert produced higher entropy embeddings, while ProtAlbert and ProtXLNet packed proteins into denser clusters (Fig. 7). Consequently, ProtAlbert and especially ProtXLNet embeddings visually separated the proteins better than ProtTXL embeddings (Fig. 7). Although sequence length is not explicitly encoded in the models, and our pooling squeezed sequences to a fixed vector size, small proteins were separated from longer proteins for all models (light blue Fig. 7). Also, all models learnt to distinguish between soluble proteins and transmembrane proteins (brown, Fig. 7) and to some

extent secondary structure composition, i.e. all-alpha versus all-beta (dark blue vs. dark green, Fig. 7).

Capturing aspects of protein function. Using the same proteins as for SCOPe but different annotations (EC-numbers [59]), we assessed whether the LM embeddings captured aspects of protein function, namely EC numbers (proteins from SCOPe without known ECs were removed, making Figs. 12 and 7 not directly comparable). Although most proteins were scattered for all LMs, ProtTXL clustered some proteins into transferases, hydrolases and oxidoreductases (particular types of enzymes).

Capturing domains of life and viruses. Typically, the following three domains of life are distinguished: *archaea*, *bacteria*, and *eukarya*, while *viruses* are not considered as life. For ease of comparison, we again used the SCOPe proteins and the same fixed-size representations for this analysis. Despite being trained differently (ProtTXL/ProtXLNet predicting next token vs. ProtBert/ProtAlbert reconstructing noise), all models were able to capture domain-specific aspects (Fig. 13). In general, Eukarya and bacteria were separated best by all LMs, while viruses and archaea formed less homogeneous clusters. When comparing the different LMs, the same trend as for protein structure classes 7 could be observed: ProtTXL and ProtBert produced higher entropy clusters while ProtAlbert and ProtXLNet produce visually easier separable clusters. Interestingly, ProtBert is the only LM that produces a well-separable cluster for Archaea.

Using a different set of proteins [26], we analyzed whether or not the embeddings captured protein function as proxied by the *cellular compartment* (also referred to as *subcellular localization*) and membrane-association. All LMs learned to distinguish some aspects of localization with nuclear and extracellular proteins forming the most coherent clusters (Fig. 15). The LMs also picked up the membrane-association, clustering most proteins homogeneously (Fig. 14).

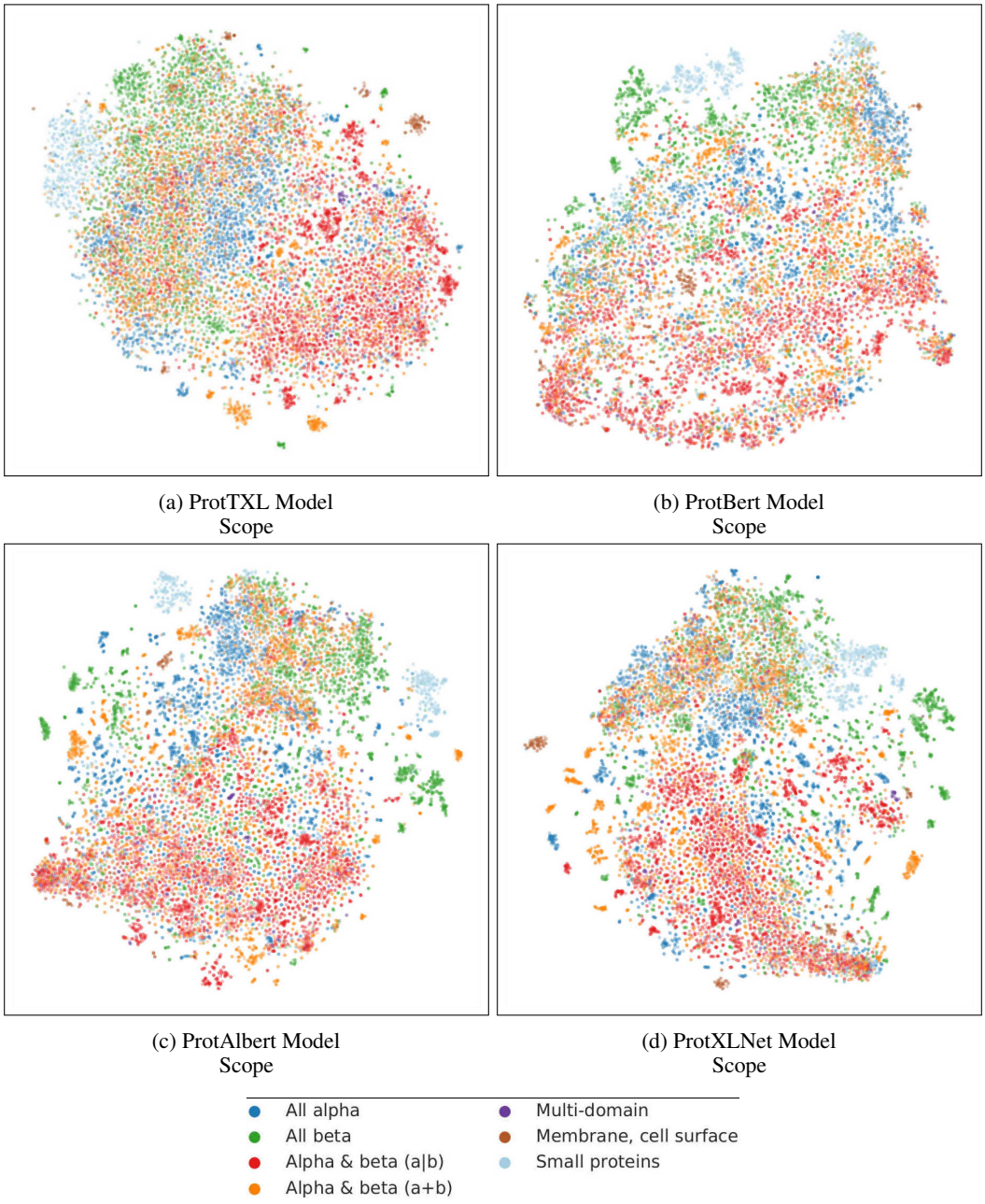
3.2 Supervised learning: embeddings yield good predictions

Successful protein predictions exclusively using embeddings as input constitutes an even more important acid test than any statistical clustering analysis could. Toward this end, we compared secondary structure (per-residue level) and localization (per-protein level) predictions, along with the classification into membrane/non-membrane proteins (per-protein level).

Per-residue prediction of secondary structure. Secondary structure was predicted by CNNs using only embeddings extracted from the last layer of our pre-trained LMs. All models were evaluated using standard measures for performance (Q3/Q8: three/eight-state per-residue accuracy, i.e. percentage of residues predicted correctly in either of the 3/8 states). Performance differed slightly between different data sets: from Q3(CASP12)=71-75% (interval marks one standard error), over Q3(CB513)=74-81%, to Q3(TS115)=75-83% (Fig. 8; results for 8-state predictions confined to Fig. 11 Supplementary Material). The computed standard error intervals fail to completely reflect the real spread of the data, because the three data sets were not consistent, i.e. the average over their performance differed by some level of statistical significance. Ultimately, this reflects problems with each of those data sets: CASP12 was too small, but completely new to all methods compared; CB513 was the largest set (513 proteins), but allowed for substantial redundancy, and TS115 (115 proteins) allowed for even more redundancy. Despite these shortcomings, these data sets enabled direct comparison to state-of-the-art methods using evolutionary information.

For simplicity, we use the worst and the best performance among the three data sets in the following to highlight the performance variation depending on the test set. For the four LMs trained on UniRef100 this resulted in Q3(ProtTXL)=71-76, Q3(ProtBert)=75-83, Q3(ProtAlbert)=74-82, and Q3(ProtXLNet)=73-81 (for 8-states: Q8(ProtTXL)=59-64, Q8(ProtBert)=63-72, Q8(ProtAlbert)=62-70 and Q8(ProtXLNet)=62-69). For ProtTXL we could also analyze the influence of the size of the database used to train the LMs: the 10-times larger BFD improved slightly over UniRef100, i.e. Q3(ProtTXL - ProtTXL-BFD)= +1. However, this difference was not statistically significant, especially, in the light of the relatively high variation between test sets.

All databases and all models (ProtTXL/ProtBert/ProtAlbert/ProtXLNet, BFD/UniRef) improved significantly over the approach using only context-free feature extractors such as word2vec-based approaches (dubbed DeepProtVec in Figs. 8 and 11). However, none of the solutions improved in any way over the state-of-the-art methods using evolutionary information (methods left of the dashed vertical line in Figs. 8 and 11), with ProtBert reducing the gap between those different approaches.



(a) ProtTXL Model Scope

(b) ProtBert Model Scope

(c) ProtAlbert Model Scope

(d) ProtXLNet Model Scope

Figure 7: **Unsupervised training captures structural features of proteins:** A redundancy reduced version (40%) of the Structural Classification of Proteins – extended (SCOPe) database was used to assess whether the language models (LMs) captured structural features of proteins without any labels. Towards this end, contextualized, fixed-size representations were generated for all proteins in the SCOPe dataset by mean-pooling over the representations extracted from the last layer of each model (average over the length of the protein). The high-dimensional embeddings were projected to 2D using t-SNE. All LMs trained here captured structural information as annotated in the main classes in SCOPe without ever having been explicitly trained on structural features.

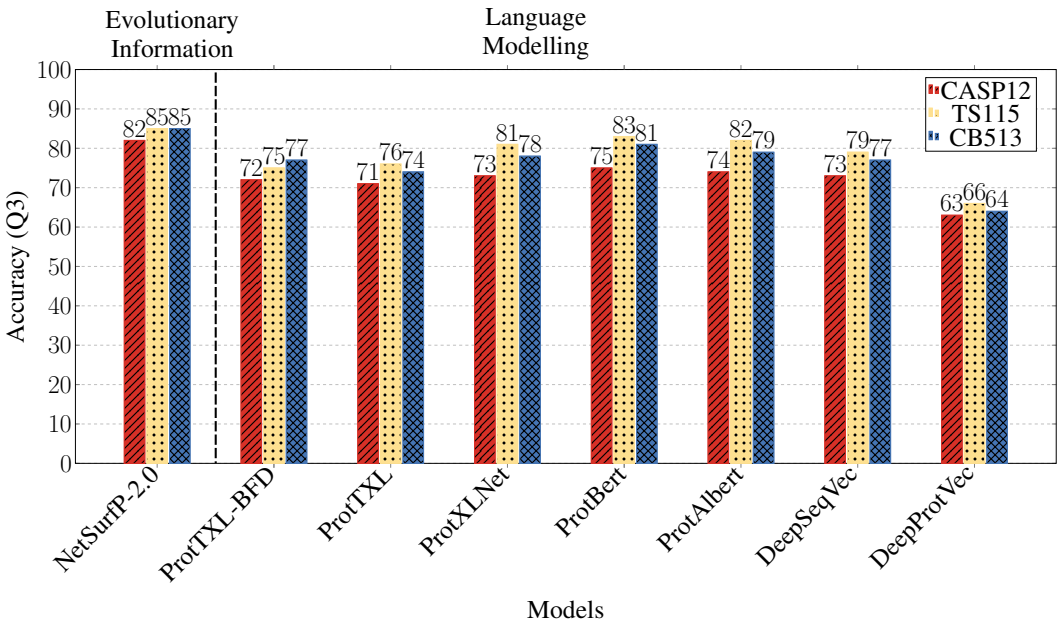


Figure 8: **Performance comparison of Language models on supervised tasks:** the predictive power of the embeddings derived from the Language Models (LMs) trained here (ProtBert, ProtAlbert, ProtTXL, ProtXLNet) was assessed via three-state secondary structure prediction (y-axis: Q3). To simplify comparability to other approaches, we used the same training and test data sets (CASP12, TS115, CB513) as an existing approach, i.e. NetSurfP-2.0 [25]. All LMs developed here were evaluated by training a simple network on top of the representations extracted from the last layer of the pre-trained LMs. As comparison, a method using evolutionary information was also added (NetSurfP-2.0, left side of the bar chart). Approaches using only the proposed embeddings (ProtBert, ProtAlbert, ProtTXL, ProtXLNet) are located one the right side of the bar chart. While outperforming uncontextualized (ProtVec [60]) as well as existing, LSTM-based LMs (SeqVec [17]), all LMs trained here still fall short compared to methods using evolutionary information.

Per-protein prediction of 10-state localization and 2-state membrane/non-membrane proteins. The feed forward model was trained to predict protein localization in ten different classes and to binary classify membrane/non-membrane proteins. For simplicity, performance was evaluated using standard accuracy (Q10 for localization, Q2 for membrane/non-membrane). ProtBert and ProtAlbert numerically performed best: Q10(ProtBert)=74, Q10(ProtAlbert)=74, while ProtTXL as well as ProtXLNet performed substantially worse: Q10(ProtTXL)=66, Q10(ProtXLNet)=68. The 10-fold increase from UniRef100 to BFD when training ProtTXL appeared to have little or detrimental effect: Q10(ProtTXL-BFD)=65 (Fig. 9). However, again those differences were not statistically significant either way.

For the binary classification into membrane/non-membrane proteins (Q2), the trend observed for localization (Q10) largely remained: ProtBert and ProtAlbert performed best (Q2(ProtBert)=89, Q2(ProtAlbert)=88, Fig. 9). However, for Q2 ProtXLNet largely closed the performance gap from Q2 (Q2(ProtXLNet)=87) while ProtTXL again performed worst (Q2(ProtTXL)=85). As for localization, there was little difference between the small (UniRef100) and large (BFD) data set used for generating the LMs: Q2(ProtTXL-BFD-ProtTXL)= +1, although the trend form localization (worse for larger data set) was reversed.

On one hand, the per-protein predictions using only embeddings as input, like those for secondary structure, remained behind the best state-of-the-art methods using evolutionary information (methods left of the dashed vertical line in Fig. 9). On the other hand, performance was substantially and statistically significantly higher for ProtAlbert/ProtBert/ProtTXL/ProtXLNet than for the word2vec-like solutions (DeepProtVec in Fig. 9). However, in contrast to the per-residue solutions, the per-protein predictions outperformed some popular methods that did use evolutionary information

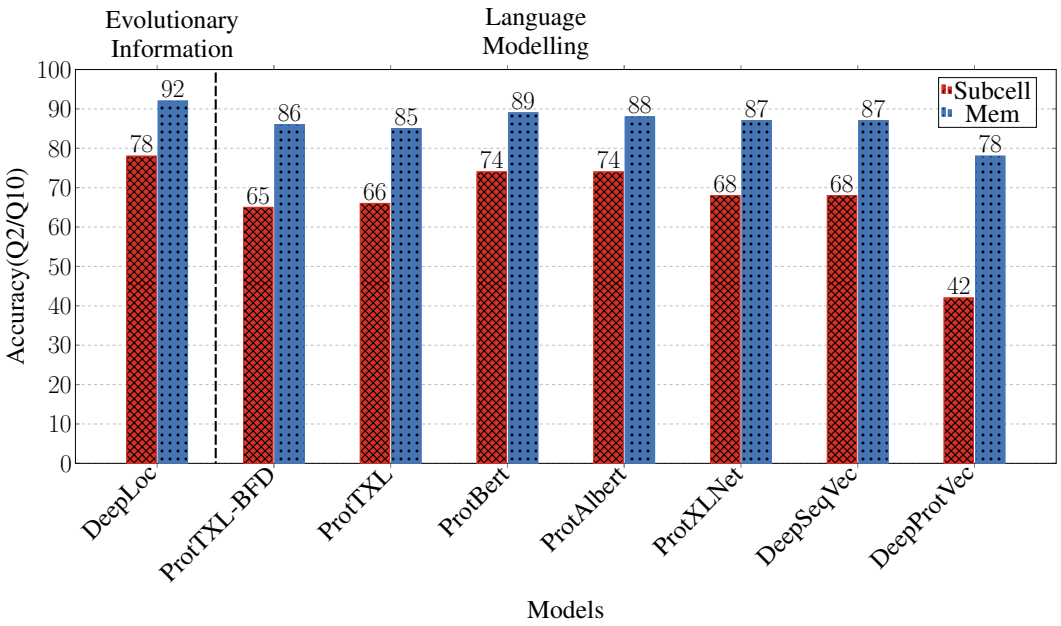


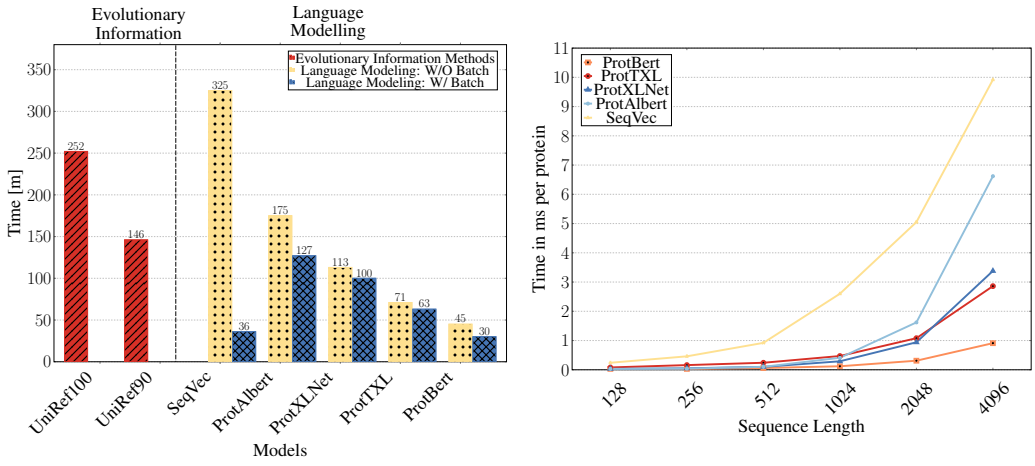
Figure 9: **Performance comparison on protein-level supervised tasks:** the protein LMs trained here (ProtTXL, ProtBert, ProtAlbert, ProtXLNet) were compared on the prediction of subcellular localization in 10-states as well as on classifying proteins into membrane-bound and soluble using the dataset of an existing approach, i.e. DeepLoc [26]). A simple two-layer neural network is trained on top of fixed-size representations for each protein which were derived by averaging over the length dimension of embeddings extracted from the last layer of the language models. The performance of all our LMs falls short when being compared to an existing approach which uses evolutionary information (DeepLoc). However, transformer-based protein LMs introduced here outperform previously published LSTM-based protein LM approaches (DeepSeqVec) as well as uncontextualized approaches using word2vec (DeepProtVec).

(Fig. 9), specifically ProtBert reached a value only a few percentage points below the current state-of-the-art using evolutionary information (Q2(ProtBert-Deeploc)=-3, Q10(ProtBert-DeepLoc)=-4).

3.3 Fast predictions from embeddings

Although embedding-based predictions were less accurate than those using evolutionary information, one crucial advantage of representations derived from protein LMs is their speed-up compared to database searches required to generate evolutionary information. This speed-up was quantified by comparing the time required to generate representations for each protein in the human proteome (20.353 proteins with a median sequence length of 415 residues) using our protein LMs or mmseqs2 [61], the fastest tool to gather evolutionary information from protein sequence databases at the moment. The same parameters as in NetSurfP-2.0 [25] were used to search with mmseqs2 the human proteome against two large protein sequence database (UniRef90=113M and UniRef100=216M proteins), i.e. the number of iterations was set to two (profile search) and the maximum number of sequences passing the pre-filtering was set to 2.000. For the database search we used an IntelR© XeonR© Scalable Processor "Skylake" Gold 6248 with 40 threads, SSD and 377GB main memory, while protein LMs were run on a single Nvidia P100 with 16GB memory using single sequence prediction (Batch = 1), and dynamic batch size based on the variable sequences length. Using the experimental setup described above, mmseqs2 is around 7- or 4-times slower than the fastest LMs (SeqVec and ProtBert, Fig. 10 (a)) when searching UniRef100 or UniRef90, respectively. The comparison also highlights the increased benefit of higher batch-size for LSTM-based LMs such as SeqVec compared to transformer-based LMs.

When checking the effect of protein sequence length on the inference speed of protein LMs (Fig. 10 (b)), we noticed that SeqVec is the slowest model (9.92s) for long proteins (up to 4096 residues),



(a) Inference Speed for LMs Vs Evolutionary Information

(b) Inference speed for a single sequence using LMs Models

Figure 10: **Inference Speed Comparison:** Panel (a) compares the time required to generate protein representations for the human proteome (20.353 proteins) using either our protein LMs or mmseqs2 (protein sequence search tool [61] used to generate evolutionary information). Here, we used mmseqs2 (red bar) to search each protein in the human proteome against two large protein sequence database (UniRef90 and UniRef100 with 113M and 216M proteins, respectively). Only embedding or search time is reported, i.e. no pre-processing or pre-training was measured. mmseqs2 was run on a Intel® Xeon® Scalable Processor "Skylake" Gold 6248 with 40 threads, SSD and 377GB main memory, while protein LMs were run on a single Nvidia P100 with 16GB memory using batch size of 1 (yellow bar), and dynamic batch size based on sequence length (blue bar). Panel (b) highlights the effect of protein sequence length on the inference time ofthe LMs on a Nvidia Titan V with 12GB memory (batch-size=1).

while ProtBert is the fastest (0.91s). We used only single sequence processing on a Nvidia Titan V with 12GB vRAM.

We also investigated the cross-effect of sequence length and batch-size (see Table 2) on the inference speed of different protein LMs. When using a single Nvidia Titan V on varying batch-sizes (1,16,32) as well as sequence lengths (128, 256, 512), SeqVec provided the fastest inference with an average of 0.02 seconds per protein when using a batch size of 32, followed by ProtBert (0.03s). However, the batch-size of ProtBert could have been further increased on the same hardware but was limited to allow a direct comparison between all models.

4 Discussion

Supercomputers such as Summit [1] and Google's cloud TPU Pod [2], combined with optimized libraries such as IBM DDL [7] and Horovod [6] set the stage for training LMs with billions of free parameters on large corpora with terabytes of data in hours or days. Increasing model size improves performance for some NLP applications [14], although the massive data challenges the communication between thousands of nodes and divergence between large batches during training. Here, we presented some solutions to overcome these challenges by fully utilizing 20% of Summit for the training of TransformerXL [49], as well as, by using one TPU Pod V3-512 for the training of Bert [47], Albert [48] and XLNet [13] on protein sequences. This translated into the parallel use of 5616 GPUs on Summit or 512 TPU cores on a TPU Pod, while avoiding training divergence with specialized optimizers such as LAMB [52] up to a global batch size of 44K samples (here: proteins). It remains to be tested whether the entire Summit or a complete TPU Pod could be utilized to train LMs on protein sequences.

4.1 HPC challenges for up-scaling protein LMs on Summit

Up-scaling LMs to the enormous sizes of protein databases (our largest data set of BFD contained 112-times the number of words in the English Wikipedia) on Summit threw up six main challenges that we addressed as follows.
(1) Architecture: Summit is based on IBM Power processors, while most libraries and software tools are written for Intel and AMD architectures. This makes finding compatible tools directly from the developers often challenging. However, the IBM Watson Machine Learning Module, included almost all necessary deep learning libraries, for others common package management tools such as Anaconda [62] were available.
(2) Communication overhead: large-scale training increased the communication overhead. After testing several backends, IBM DDL used the least computation time on Summit.
(3) Distributed training: using thousands of GPUs with Tensorflow [3] and Pytorch [4] required the handling of distributed communication between nodes and assigning work loads (tokenized text files) correctly to workers (GPUs) to be extremely efficient. Horovod [6] provided the easiest and most optimized distributed training for both of these frameworks on Summit.
(4) File sharing: parallel writing of files may increase run-time. During training, multiple nodes read from and write to the same files holding model parameters and logs. In order to address multiple writes on a single file separate log copies for each node were used, while only storing a single copy of the model's weights on the master node. Data set files remained shared, as file reading was not impaired.
(5) Pre-processing: pre-processing, especially tokenization, of batches on the fly increased the GPU waiting time and CPU processing time while reducing storage requirements. For small data sets (few GBs), we recommend pre-processing the batches and storing them on disk before training. For large data sets (TBs of data), there is a trade-off between disk space requirements and training time. In our hands, the most efficient solution for pre-processing was to use ORNL's Rhea cluster. It reduced the pre-processing time from 7.5 months to fewer than 2 days, by converting the original sequential script to distributed processing using MPI.
(6) Deep learning library: The integration of LMS into Pytorch (ProtTXL) required adjusting only a few parameters; in contrast, Tensorflow (ProtBert) required more code changes. Tensorflow might compensate for this problem by auto-tuning certain parameters such as the memory usage; however, for our use-case, this failed. The different parameters for Pytorch and Tensorflow resulted in different behaviors with respect to swapping in and out nodes between GPU and CPU. This in turn varied speed and model/batch sizes.

4.2 Comparison of protein LM training on Summit and TPU Pod

Compared to Summit, TPU devices which are highly optimized for neural network training natively supported distributed training and efficient communication between multiple cores. However, during (protein) LM training one of the most important aspects is the sample throughput, i.e. the number of samples processed per second. We compared the throughput of ProtBert on both systems, resulting in an average throughput of 10k for Summit (936 nodes) and 8.25k for a TPU Pod (v3-512 cores). However, we only used a fraction of both systems (20% of Summit and 25% of the TPU-Pod) and tuning of LMS on Summit could have further optimized throughput but was beyond the scope of this work.

4.3 Unsupervised LMs learned rudimentary features of protein biophysics

The information about how proteins are formed, shaped, and function has been learned by the LMs because all models (ProtBert, ProtAlbert, ProtTXL, ProtXLNet) extracted valuable information as revealed by the embeddings. The basic understanding extended from biophysical features of the building blocks of proteins, the amino acids (e.g. hydrophobicity, charge, and size, Fig. 6), over classifications of protein structure (Fig. 7), and protein function (Fig. 14), to the macroscopic level of the domains of life (Fig. 13). Global structural properties (e.g. overall secondary structure content, Fig. 7) and global biochemical properties (e.g. membrane-boundness, Fig. 14) appeared most distinctive. In contrast, local features which rely on specific, short motifs were harder to distinguish (EC-numbers: Fig. 12, localization: Fig. 15).

4.4 Supervised performance: bi-directional better than uni-directional LMs for proteins

The t-SNE and UMAP analyses suggested that the LMs had extracted some level of *understanding of the language of life*. However, any statistical difference has ultimately limited validity and value if it is not predictive. In this sense prediction is the acid test of understanding. To pass this test, we used the embeddings extracting the information learned by the LMs directly as input for methods predicting aspects of protein structure and function, both on the level of the individual residue (per-residue predictions of secondary structure) and the level of entire proteins (per-protein prediction of localization and membrane/non-membrane). Overall, the supervised results confirmed [17] that evolutionary information outperforms LMs not using such information (on all per-residue 8,11 and per-protein tasks 9) with ProtBert reducing the gap from embeddings-only input to those approaches. Newer contextual models improved both over previous LM-based approaches [17] (3-4 percentage points in Q3) and over non-contextualized word2vec-type approaches [63, 64, 65] (12-16 percentage points in Q3). A merger of models using evolutionary information and embeddings might bring the best.

In NLP uni-directional models (auto-regressive) perform *on par* with bi-directional models (auto-encoding) [14, 66]. In contrast, it bi-directional context appeared crucial to model aspects of the language of life. While auto-encoding models such as Albert [48] utilize context to both sides during loss calculation, auto-regressive models such as TransformerXL [49] consider only context to one side. This difference resulted in a substantial performance difference between ProtTXL and ProtXLNet (XLNet extends Transformer-XL to capture bi-directional context), both trained on UniRef100: Q3(ProtXLNet)-Q3(ProtTXL)=3.6, Q8(ProtXLNet)-Q8(ProtTXL)=4.0, Q10(ProtXLNet)-Q10(ProtTXL)=2, Q2(ProtXLNet)-Q2(ProtTXL)=2. This might be compensated for by first pre-train on sequences and their reverse and then concatenating the output of uni-directional LMs applied on both directions. While this does not allow the LM to use bi-directional context during training, it allows supervised networks to combine context derived independently from both sides. One example for an auto-regressive model that makes use of this is ELMo [10] which concatenates the embeddings derived from a forward and a backward LSTM. Interestingly, ELMo trained on protein sequences (SeqVec) performs better than the uni-directional ProtTXL but worse (Q3,Q8) or equal (Q2,Q10) than the bi-directional ProtXLNet: Q3(ProtXLNet)-Q3(SeqVec)=1.0, Q8(ProtXLNet)-Q8(SeqVec)=0.7, Q10(ProtXLNet)-Q10(SeqVec)=0, Q2(ProtXLNet)-Q2(SeqVec)=0. While part of this difference might be explained by the difference in model size (SeqVec=93M vs. ProtXLNet=409M) and training data (SeqVec=30M vs. ProtAlbert=224M), pure uni-directionality as used in TransformerXL seems to be detrimental for modeling protein sequences.

4.5 Bigger data not always better?

LMs were trained on the largest protein database ever used for this purpose, namely BFD [36], which was more than an order of magnitude larger than UniProt [37], the standard in the field. Bigger did not equate better for all 2nd stage predictions. Unfortunately, we could not establish whether this was because the LMs learned all there was to learn from UniProt/UniRef100, or the LMs size was not large enough, or due to some intrinsic aspects of BFD which mostly contains translated metagenomic sequences, i.e. mostly by bacterial proteins for which little is known (not even if they really exist as proteins or constitute just protein fragments). May be BFD added as much noise as signal (not more, because than predictions would have become consistently worse). This might also explain the rather limited improvement in performance with respect to existing LMs [17] (ΔQ3=Q3(ProtBert)-Q3(SeqVec)=3.3%) despite a significant increase in model size (SeqVec=93M vs. ProtBert=420M) and data size (SeqVec=30M vs. ProtBert=216M). Although a ΔQ3 of 2-3 percentage points might imply an improvement that is crucial for the methods using such predictions [67], the value has also to be put into relation to the GPU/TPU hours needed to train those models: while SeqVec needed around 1680 GPU hours, ProtTXL needed 202176 GPU hours and ProtBert needed 116736 TPU core hours.

4.6 Protein LMs reached a ceiling?

Applying techniques from NLP to proteins opens new opportunities to extract information from proteins in a self-supervised, data-driven way. New protein representations may complement existing solutions, most successful when combining evolutionary information and machine learning [68, 69, 34, 70]. The gain in inference speed for protein LMs compared to traditional models using evolutionary

information is so significant that some analyses might prefer *much faster and slightly less accurate* to *better but much slower*, for instance, when time or resources for *much slower* are amiss. Nevertheless, given the experiments described here and in previous work [17, 18, 19, 20, 21, 22, 24], we might expect an upper limit for what protein LMs can learn when using auto-regressive or auto-encoding exclusively. Although this work explicitly addressed the possibility of reaching that limit, we could only conclude: 1) increasing training corpus size without accounting for redundancy or noise as well as LMs size will unlikely improve LMs 2) bi-directional models appeared superior over uni-directional models. This did not imply that larger databases will never help. Answers to the following questions might advance from the status-quo. **(1)** Why do LSTM-based approaches require fewer parameters and resources while performing similarly at downstream prediction tasks (Q3(ProtBert)-Q3(SeqVec)=3.3%) compared to Transformer-based approaches? **(2)** Could redundancy- and noise-reduction of today's largest data sets bring substantial changes? **(3)** Would the addition of auxiliary tasks such as next-sentence or sentence-order prediction offered by BERT or Albert suit protein sequences? A suggestion might be the usage of structure information [71] or evolutionary relationship [20]. **(4)** Addressing model vs. data parallelism: Were the large models introduced here still too small to capture all data? Unfortunately, this brings up training efficiency as recently investigated by sparse Transformers [72] or attention optimized with locality-sensitive hashing (LSH) [73] as introduced recently by the Reformer model [74]. **(5)** Might full precision training stabilize training and speed up convergence by leveraging 32-bit floats? Mixed precision training, employed in this evaluation, uses 16 Bit as well as 32 Bit vectors; this made it more difficult for the model to converge during training. Training the models presented here in full precision might stabilize training and thus provide more informative representations. Overall, our results established that the combination of HPC solutions for building protein LMs and subsequent training of supervised prediction methods scaled up to the largest data sets ever used in the field.

Acknowledgments

The authors thank primarily Tim Karl (TUM) and Jian Kong (TUM) for invaluable help with hardware and software; Inga Weise and Aline Schmidt (both TUM) for support with many other aspects of this work; Florian Matthes (TUM) for his invaluable support and encourage for us. Thanks for invaluable support and feedback from NVIDIA, in particular to to Ulrich Michaelis, Ada Sedova, Geetika Gupta, Axel Koehler, Frederic Pariente, Jonathan Lefman, and Thomas Bradley. No aspect of this work could have been realized without strong support from many at ORNL: thanks; these include John Gounley,Hong-Jun Yoon, Georgia Tourassi, Bill, Brian, Junqi, Graham and Verónica for helping us on fixing issues that occurred while training on Summit. Furthermore, special thanks to Jack Wells for giving us the opportunity to access and work with Summit. From IBM, we would like to thank Nicolas Castet and Bryant Nelson for their help to fix issues and enhance the performance of IBM PowerAI. From Google, we would like to deeply thank Jamie Kinney, Alex Schroeder, Nicole DeSantis, Andrew Stein, Vishal Mishra, Eleazar Ortiz, Nora Limbourg, Cristian Mezzanotte and all TFRC Team for their invaluable support to setup our project on Google Cloud and solve all the related Google TPU and servers issues. Last, not least, thanks to all those who deposit their experimental data in public databases, and to those who maintain these databases.

This work was supported by a grant from Software Campus through the German Ministry for Research and Education (BMBF: Bundesministerium fuer Bildung und Forschung), a grant from the Alexander von Humboldt foundation through the German Ministry for Research and Education (BMBF: Bundesministerium fuer Bildung und Forschung),and by a grant from the Deutsche Forschungsgemeinschaft (DFG–GZ: RO1320/4–1). We gratefully acknowledge the support of NVIDIA Corporation with the donation of two Titan GPU used for this research development phase. We also want to thank LRZ (Leibniz Rechenzentrum) for providing us access to DGX-1(V100) for the testing phase.

Finally and most importantly, this research used resources of the Oak Ridge Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC05-00OR22725, and resources of TPU pods under TensorFlow Research Cloud grant. Furthermore, Rostlab acknowledge support from Google Cloud and Google Cloud Research Credits program to fund this project under Covid19 HPC Consortium grant.

References

[1] Jack Wells, Buddy Bland, Jeff Nichols, Jim Hack, Fernanda Foertter, Gaute Hagen, Thomas Maier, Moetasim Ashfaq, Bronson Messer, and Suzanne Parete-Koon. Announcing Supercomputer Summit. Technical report, Oak Ridge National Lab. (ORNL), Oak Ridge, TN (United States), June 2016.

[2] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Daniel Killebrew, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. In-Datacenter Performance Analysis of a Tensor Processing Unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ISCA '17, pages 1–12, Toronto, ON, Canada, June 2017. Association for Computing Machinery.

[3] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mane, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viegas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. *arXiv:1603.04467 [cs]*, March 2016.

[4] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d\textquotesingle Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8026–8037. Curran Associates, Inc., 2019.

[5] David Kirk. NVIDIA cuda software and gpu parallel computing architecture. In *Proceedings of the 6th International Symposium on Memory Management*, ISMM '07, pages 103–104, Montreal, Quebec, Canada, October 2007. Association for Computing Machinery.

[6] Alexander Sergeev and Mike Del Balso. Horovod: Fast and easy distributed deep learning in TensorFlow. *arXiv:1802.05799 [cs, stat]*, February 2018.

[7] Minsik Cho, Ulrich Finkler, Sameer Kumar, David Kung, Vaibhav Saxena, and Dheeraj Sreedhar. PowerAI DDL. *arXiv:1708.02188 [cs]*, August 2017.

[8] Fujitsu Limited. Press release announcing Supercomputer Fugaku. Technical report, RIKEN, December 2019. _eprint: `https://www.fujitsu.com/global/about/resources/news/press-releases/2019/1202-01.html?_fsi=q8QhaYU5`.

[9] Nicolay Hammer, Ferdinand Jamitzky, Helmut Satzger, Momme Allalen, Alexander Block, Anupam Karmakar, Matthias Brehm, Reinhold Bader, Luigi Iapichino, Antonio Ragagnin, Vasilios Karakasis, Dieter Kranzlmüller, Arndt Bode, Herbert Huber, Martin Kühn, Rui Machado, Daniel Grünewald, Philipp V. F. Edelmann, Friedrich K. Röpke, Markus Wittmann, Thomas Zeiser, Gerhard Wellein, Gerald Mathias, Magnus Schwörer, Konstantin Lorenzen, Christoph Federrath, Ralf Klessen, Karl-Ulrich Bamberg, Hartmut Ruhl, Florian Schornbaum, Martin Bauer, Anand Nikhil, Jiaxing Qi, Harald Klimach, Hinnerk Stüben, Abhishek Deshmukh, Tobias Falkenstein, Klaus Dolag, and Margarita Petkova. Extreme Scale-out SuperMUC Phase 2 - lessons learned. *arXiv:1609.01507 [astro-ph, physics:physics]*, September 2016.

[10] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. *arXiv:1802.05365 [cs]*, March 2018.

[11] Jeremy Howard and Sebastian Ruder. Universal Language Model Fine-tuning for Text Classification. *arXiv:1801.06146 [cs, stat]*, May 2018.

[12] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, \Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.

[13] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. XLNet: Generalized Autoregressive Pretraining for Language Understanding. *arXiv:1906.08237 [cs]*, January 2020.

[14] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism. *arXiv:1909.08053 [cs]*, March 2020.

[15] Burkhard Rost and Chris Sander. Bridging the protein sequence-structure gap by structure predictions. *Annual Review of Biophysics and Biomolecular Structure*, 25:113–136, 1996.

[16] John Ingraham, Vikas Garg, Regina Barzilay, and Tommi Jaakkola. Generative Models for Graph-Based Protein Design. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d\textquotesingle Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 15820–15831. Curran Associates, Inc., 2019.

[17] Michael Heinzinger, Ahmed Elnaggar, Yu Wang, Christian Dallago, Dmitrii Nechaev, Florian Matthes, and Burkhard Rost. Modeling aspects of the language of life through transfer-learning protein sequences. *BMC Bioinformatics*, 20(1):723, December 2019.

[18] Ethan C. Alley, Grigory Khimulya, Surojit Biswas, Mohammed AlQuraishi, and George M. Church. Unified rational protein engineering with sequence-based deep representation learning. *Nature Methods*, 16(12):1315–1322, December 2019.

[19] Ali Madani, Bryan McCann, Nikhil Naik, Nitish Shirish Keskar, Namrata Anand, Raphael R. Eguchi, Po-Ssu Huang, and Richard Socher. ProGen: Language Modeling for Protein Generation. *bioRxiv*, page 2020.03.07.982272, March 2020.

[20] Seonwoo Min, Seunghyun Park, Siwon Kim, Hyun-Soo Choi, and Sungroh Yoon. Pre-Training of Deep Bidirectional Protein Sequence Representations with Structural Information. *arXiv:1912.05625 [cs, q-bio, stat]*, February 2020.

[21] Roshan Rao, Nicholas Bhattacharya, Neil Thomas, Yan Duan, Peter Chen, John Canny, Pieter Abbeel, and Yun Song. Evaluating Protein Transfer Learning with TAPE. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d\textquotesingle Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 9689–9701. Curran Associates, Inc., 2019.

[22] Jose Juan Almagro Armenteros, Alexander Rosenberg Johansen, Ole Winther, and Henrik Nielsen. Language modelling for biological sequences – curated datasets and baselines. *bioRxiv*, page 2020.03.09.983585, March 2020.

[23] Mohammed AlQuraishi. End-to-End Differentiable Learning of Protein Structure. *Cell Systems*, 8(4):292–301.e3, April 2019.

[24] Alexander Rives, Siddharth Goyal, Joshua Meier, Demi Guo, Myle Ott, C. Lawrence Zitnick, Jerry Ma, and Rob Fergus. Biological structure and function emerge from scaling unsupervised learning to 250 million protein sequences. *bioRxiv*, page 622803, May 2019.

[25] Michael Schantz Klausen, Martin Closter Jespersen, Henrik Nielsen, Kamilla Kjærgaard Jensen, Vanessa Isabell Jurtz, Casper Kaae Sønderby, Morten Otto Alexander Sommer, Ole Winther, Morten Nielsen, Bent Petersen, and Paolo Marcatili. NetSurfP-2.0: Improved prediction of protein structural features by integrated deep learning.

Proteins: Structure, Function, and Bioinformatics, 87(6):520–527, 2019. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/prot.25674.

[26] José Juan Almagro Armenteros, Casper Kaae Sønderby, Søren Kaae Sønderby, Henrik Nielsen, and Ole Winther. DeepLoc: Prediction of protein subcellular localization using deep learning. *Bioinformatics*, 33(21):3387–3395, November 2017.

[27] Jianyi Yang, Ivan Anishchenko, Hahnbeom Park, Zhenling Peng, Sergey Ovchinnikov, and David Baker. Improved protein structure prediction using predicted interresidue orientations. *Proceedings of the National Academy of Sciences*, 117(3):1496–1503, January 2020.

[28] A. Kulandaisamy, Jan Zaucha, Ramasamy Sakthivel, Dmitrij Frishman, and M. Michael Gromiha. Pred-MutHTP: Prediction of disease-causing and neutral mutations in human transmembrane proteins. *Human Mutation*, 41(3):581–590, 2020. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/humu.23961.

[29] Maria Schelling, Thomas A. Hopf, and Burkhard Rost. Evolutionary couplings and sequence variation effect predict protein binding sites. *Proteins: Structure, Function, and Bioinformatics*, 86(10):1064–1074, 2018. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/prot.25585.

[30] Michael Bernhofer, Edda Kloppmann, Jonas Reeb, and Burkhard Rost. TMSEG: Novel prediction of transmembrane helices. *Proteins: Structure, Function, and Bioinformatics*, 84(11):1706–1716, 2016. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/prot.25155.

[31] Debora S. Marks, Lucy J. Colwell, Robert Sheridan, Thomas A. Hopf, Andrea Pagnani, Riccardo Zecchina, and Chris Sander. Protein 3D Structure Computed from Evolutionary Sequence Variation. *PLOS ONE*, 6(12):e28766, December 2011.

[32] Predrag Radivojac, Zoran Obradovic, David K. Smith, Guang Zhu, Slobodan Vucetic, Celeste J. Brown, J. David Lawson, and A. Keith Dunker. Protein flexibility and intrinsic disorder. *Protein Science*, 13(1):71–80, 2004. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1110/ps.03128904.

[33] Nelson Perdigão, Julian Heinrich, Christian Stolte, Kenneth S. Sabir, Michael J. Buckley, Bruce Tabor, Beth Signal, Brian S. Gloss, Christopher J. Hammang, and Burkhard Rost. Unexpected features of the dark proteome. *Proceedings of the National Academy of Sciences*, 112(52):15898–15903, 2015.

[34] Burkhard Rost and Chris Sander. Combining evolutionary information and neural networks to predict protein secondary structure. *Proteins: Structure, Function, and Genetics*, 19:55–72, 1994.

[35] Baris E. Suzek, Yuqi Wang, Hongzhan Huang, Peter B. McGarvey, and Cathy H. Wu. UniRef clusters: A comprehensive and scalable alternative for improving sequence similarity searches. *Bioinformatics*, 31(6):926–932, March 2015.

[36] Martin Steinegger and Johannes Söding. Clustering huge protein sequence sets in linear time. *Nature Communications*, 9(1):1–8, June 2018.

[37] The UniProt Consortium. UniProt: A worldwide hub of protein knowledge. *Nucleic Acids Research*, 47(D1):D506–D515, January 2019.

[38] Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, Phillipp Koehn, and Tony Robinson. One Billion Word Benchmark for Measuring Progress in Statistical Language Modeling. *arXiv:1312.3005 [cs]*, March 2014.

[39] Guoli Wang and Roland L. Dunbrack Jr. PISCES: A protein sequence culling server. *Bioinformatics*, 19(12):1589–1591, 2003.

[40] Helen M. Berman, John Westbrook, Zukang Feng, Gary Gilliland, T. N. Bhat, Helge Weissig, Ilya N. Shindyalov, and Philip E. Bourne. The Protein Data Bank. *Nucleic Acids Research*, 28(1):235–242, January 2000.

[41] Yuedong Yang, Jianzhao Gao, Jihua Wang, Rhys Heffernan, Jack Hanson, Kuldip Paliwal, and Yaoqi Zhou. Sixty-five years of the long march in protein secondary structure prediction: The final stretch? *Briefings in bioinformatics*, 19(3):482–494, 2018.

[42] James A. Cuff and Geoffrey J. Barton. Evaluation and improvement of multiple sequence methods for protein secondary structure prediction. *Proteins: Structure, Function, and Bioinformatics*, 34(4):508–519, 1999.

[43] Luciano A. Abriata, Giorgio E. Tamò, Bohdan Monastyrskyy, Andriy Kryshtafovych, and Matteo Dal Peraro. Assessment of hard target modeling in CASP12 reveals an emerging role of alignment-based contact prediction methods. *Proteins: Structure, Function, and Bioinformatics*, 86:97–112, 2018.

[44] Laurens van der Maaten and Geoffrey Hinton. Visualizing Data using t-SNE. *Journal of Machine Learning Research*, 9(Nov):2579–2605, 2008.

[45] John-Marc Chandonia, Naomi K. Fox, and Steven E. Brenner. SCOPe: Classification of large macromolecular structures in the structural classification of proteins—extended database. *Nucleic Acids Research*, 47(D1):D475–D481, January 2019.

[46] E. C. Webb. *Enzyme Nomenclature 1992. Recommendations of the Nomenclature committee of the International Union of Biochemistry and Molecular Biology.* Academic Press, New York, 1992 edition, 1992.

[47] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv:1810.04805 [cs]*, May 2019.

[48] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. ALBERT: A Lite BERT for Self-supervised Learning of Language Representations. *arXiv:1909.11942 [cs]*, February 2020.

[49] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V. Le, and Ruslan Salakhutdinov. Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context. *arXiv:1901.02860 [cs, stat]*, June 2019.

[50] Erik Schils and Pieter de Haan. Characteristics of Sentence Length in Running Text. *Literary and Linguistic Computing*, 8(1):20–26, January 1993.

[51] Ananthan Nambiar, Maeve Elizabeth Heflin, Simon Liu, Sergei Maslov, Mark Hopkins, and Anna Ritz. Transforming the language of life: Transformer neural networks for protein prediction tasks. *BioRxiv*, 2020.

[52] Yang You, Jing Li, Sashank Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. Large Batch Optimization for Deep Learning: Training BERT in 76 minutes. In *International Conference on Learning Representations*, September 2019.

[53] Akhilesh Gotmare, Nitish Shirish Keskar, Caiming Xiong, and Richard Socher. A Closer Look at Deep Learning Heuristics: Learning rate restarts, Warmup and Distillation. *arXiv:1810.13243 [cs, stat]*, October 2018.

[54] Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the Variance of the Adaptive Learning Rate and Beyond. *arXiv:1908.03265 [cs, stat]*, March 2020.

[55] Google TPU. https://cloud.google.com/tpu/docs/system-architecture, June 2020.

[56] Nvidia Apex. https://github.com/NVIDIA/apex, March 2020.

[57] Tung D. Le, Haruki Imai, Yasushi Negishi, and Kiyokuni Kawachiya. TFLMS: Large Model Support in TensorFlow by Graph Rewriting. *arXiv:1807.02037 [cs, stat]*, October 2019.

[58] Jianwei Feng and Dong Huang. Optimal Gradient Checkpoint Search for Arbitrary Computation Graphs. *arXiv:1808.00079 [cs, stat]*, September 2019.

[59] Amos Bairoch. The ENZYME database in 2000. *Nucleic acids research*, 28(1):304–305, 2000.

[60] Ehsaneddin Asgari and Mohammad RK Mofrad. Continuous distributed representation of biological sequences for deep proteomics and genomics. *PloS one*, 10(11), 2015.

[61] Martin Steinegger and Johannes Söding. Mmseqs2 enables sensitive protein sequence searching for the analysis of massive data sets. *Nature biotechnology*, 35(11):1026–1028, 2017.

[62] Continuum Analytics et al. Anaconda software distribution. *Computer software. Vers*, pages 2–2, 2016.

[63] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed Representations of Words and Phrases and their Compositionality. *arXiv:1310.4546 [cs, stat]*, October 2013.

[64] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. GloVe: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[65] Armand Joulin, Edouard Grave, Piotr Bojanowski, Matthijs Douze, Hérve Jégou, and Tomas Mikolov. FastText.zip: Compressing text classification models. *arXiv preprint arXiv:1612.03651*, 2016.

[66] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. ZeRO: Memory Optimization Towards Training A Trillion Parameter Models. *arXiv:1910.02054 [cs, stat]*, October 2019.

[67] Dariusz Przybylski and Burkhard Rost. Improving fold recognition without folds. *Journal of Molecular Biology*, 341:255–269, 2004.

[68] Burkhard Rost and Chris Sander. Prediction of protein secondary structure at better than 70% accuracy. *Journal of Molecular Biology*, 232:584–599, 1993.

[69] Burkhard Rost and Chris Sander. Improved prediction of protein secondary structure by use of sequence profiles and neural networks. *Proceedings of the National Academy of Sciences*, 90:7558–7562, 1993.

[70] Burkhard Rost. PHD: predicting one-dimensional protein structure by profile based neural networks. *Methods in Enzymology*, 266:525–539, 1996.

[71] Tristan Bepler and Bonnie Berger. Learning protein sequence embeddings using information from structure. *arXiv:1902.08661 [cs, q-bio, stat]*, October 2019.

[72] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating Long Sequences with Sparse Transformers. *arXiv:1904.10509 [cs, stat]*, April 2019.

[73] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, pages 604–613, 1998.

[74] Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. Reformer: The Efficient Transformer. In *International Conference on Learning Representations*, September 2019.

Supplementary Online Material (SOM)

-1.1 Supervised Learning

In the following we compare the performance of the language models introduced here and methods using evolutionary information on different supervised tasks. The tasks cover predictions classifying single residues within a protein as well as overall properties of proteins.

-1.1.1 8-state secondary structure prediction

On the level of single residues, we also compare our protein LMs on results for secondary structure prediction in 8-states as shown in Fig. 11.

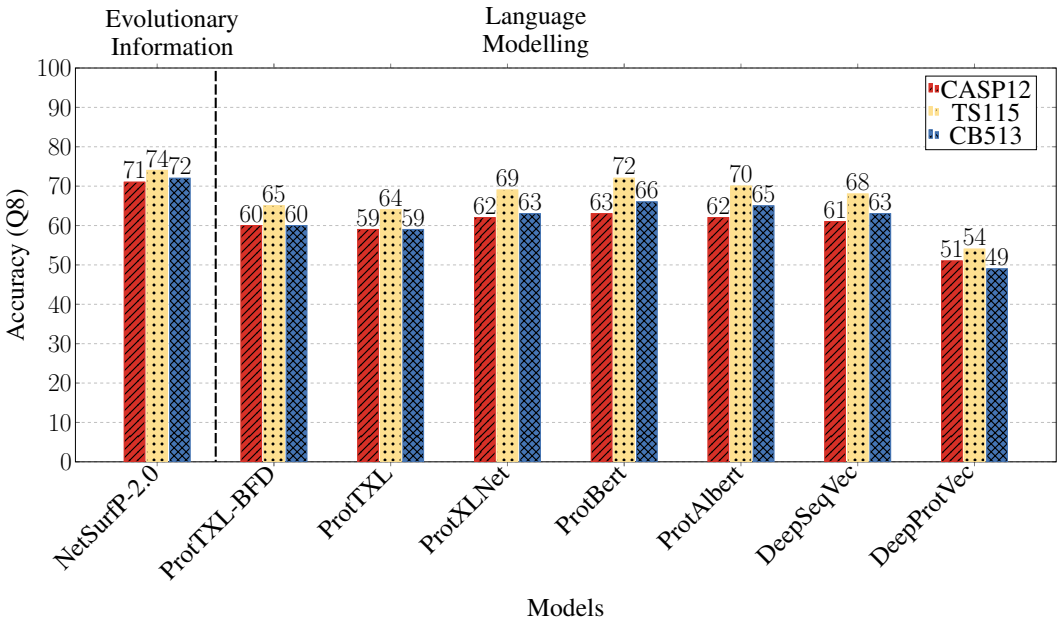


Figure 11: **Performance comparison of Language models on supervised tasks:** similar to the analysis performed for three-state secondary structure (Fig. 8), the features learnt by the proposed Language models (LMs) trained here (ProtBert, ProtAlbert, ProtTXL, ProtXLNet) were also evaluated on eight-state secondary structure prediction (y-axis: Q8). The same datasets (NetSurfP-2.0 [25]), pre-processing steps as well as the same supervised models were used for this analysis, confirming the trend suggested by the three-state secondary structure prediction.

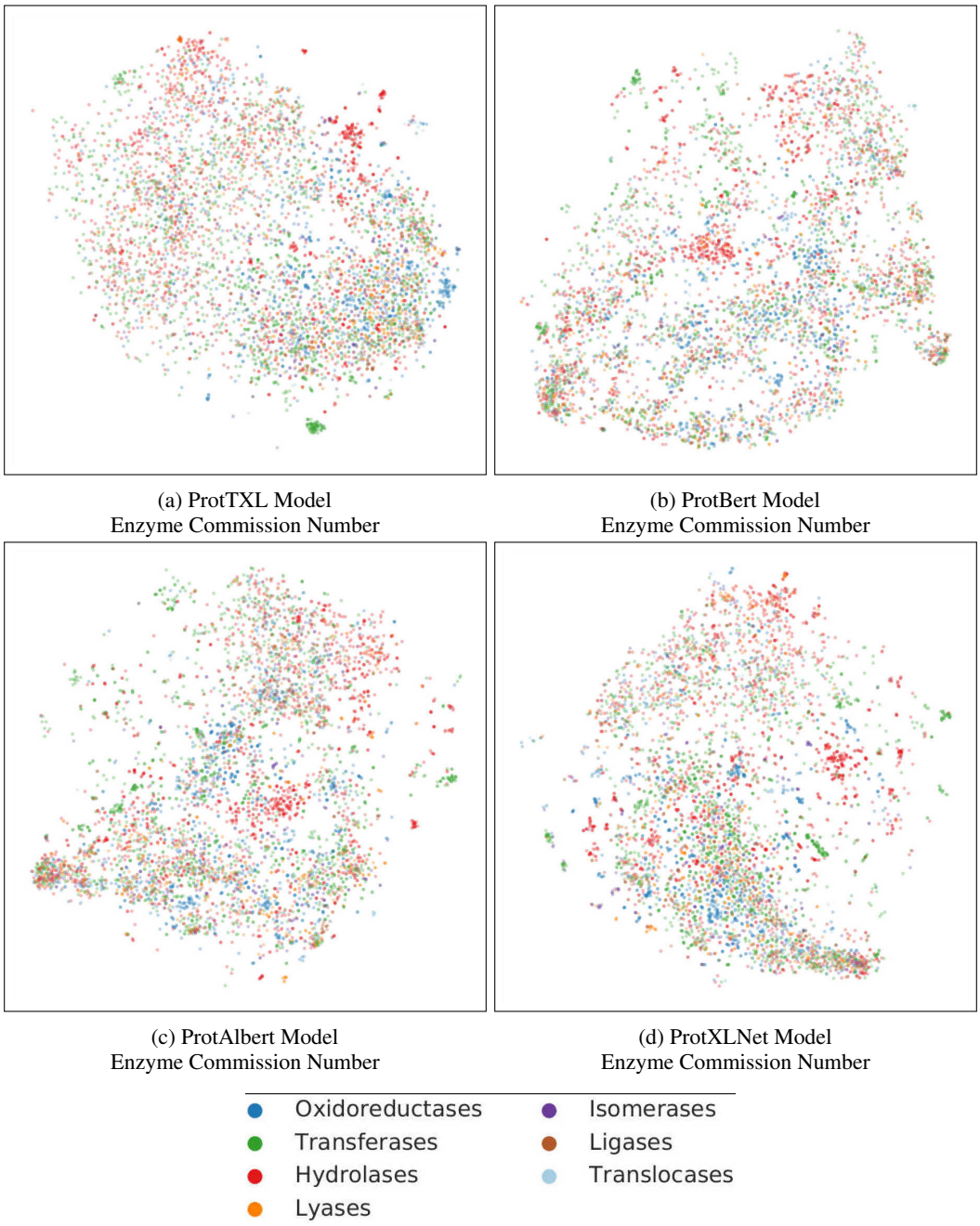
-1.2 Unsupervised Learning

Using t-SNE projections, the information content stored within the novel embeddings was qualitatively assessed on various levels, ranging from different aspects of protein function (E.C. numbers, subcellular localization and membrane-boundness) to the level of kingdoms of life, i.e. Eukaryota, Bacteria and Archaea (for completeness here also including Viruses).

Enzyme Commission - EC - numbers For the analysis of protein function, we used again the SCOPe dataset but replaced annotations on the proteins' structures by functions as defined by EC. Proteins without this annotation were removed from the analysis. Results are shown in 12.

Kingdoms of Life and Viruses Whole protein embeddings were also used to assess higher-order properties. Specifically, kingdoms, i.e. Eukaryota, Bacteria, Archaea and Viruses, were studied as shown in Fig. 13.

Membrane-Bound vs Water-Soluble We used a published dataset [26] to analyse whether our models were able to extract information that allows them to distinguish between membrane-bound and soluble proteins. Our results (Fig. 14) show that all models were able to learn features that allow



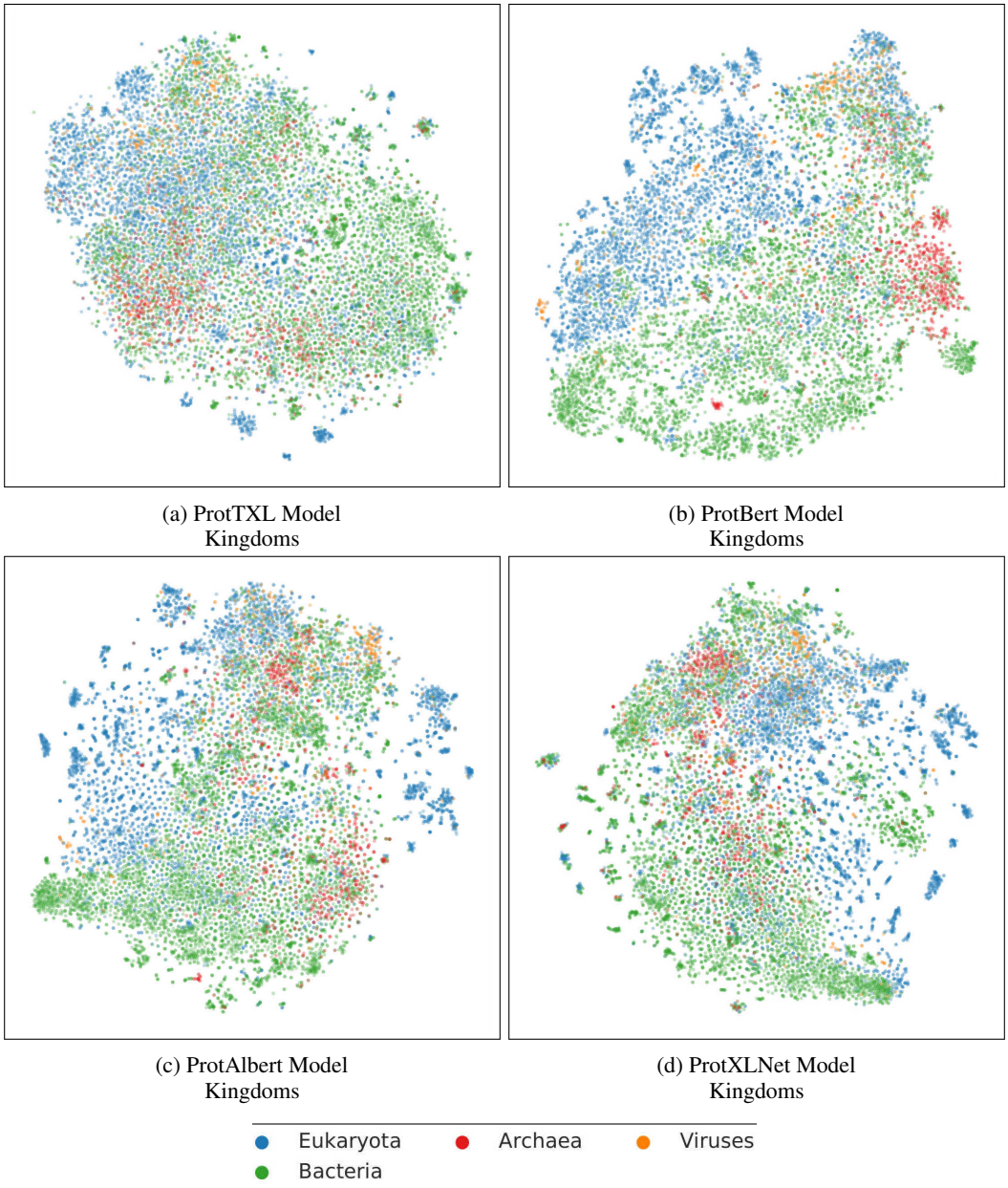
(a) ProtTXL Model
Enzyme Commission Number

(b) ProtBert Model
Enzyme Commission Number

(c) ProtAlbert Model
Enzyme Commission Number

(d) ProtXLNet Model
Enzyme Commission Number

Figure 12: **Unsupervised training captures functional aspects of proteins:** in analogy to the analysis of structural features in Fig. 7, the same dataset (SCOPe reduced at 40% sequence identity) was used to a check whether the language models (LMs) trained here captured functional features of proteins without being explicitly trained on such labels. Therefor, protein functions as defined by the Enzyme Commission Number (E.C.) were used to annotate the proteins in our SCOPe dataset. If a protein had no E.C. annotation it was excluded from this analysis. For the remaining proteins embeddings were generated on the per-protein level again by mean-pooling over the representations extracted from the last layer of each model (average over the length of the protein). Afterwards, 2D t-SNE projections of the high-dimensional representations were computed to visualize the data. Many small, local clusters share function as given by the main classes in the Enzyme Commission number (E.C.).



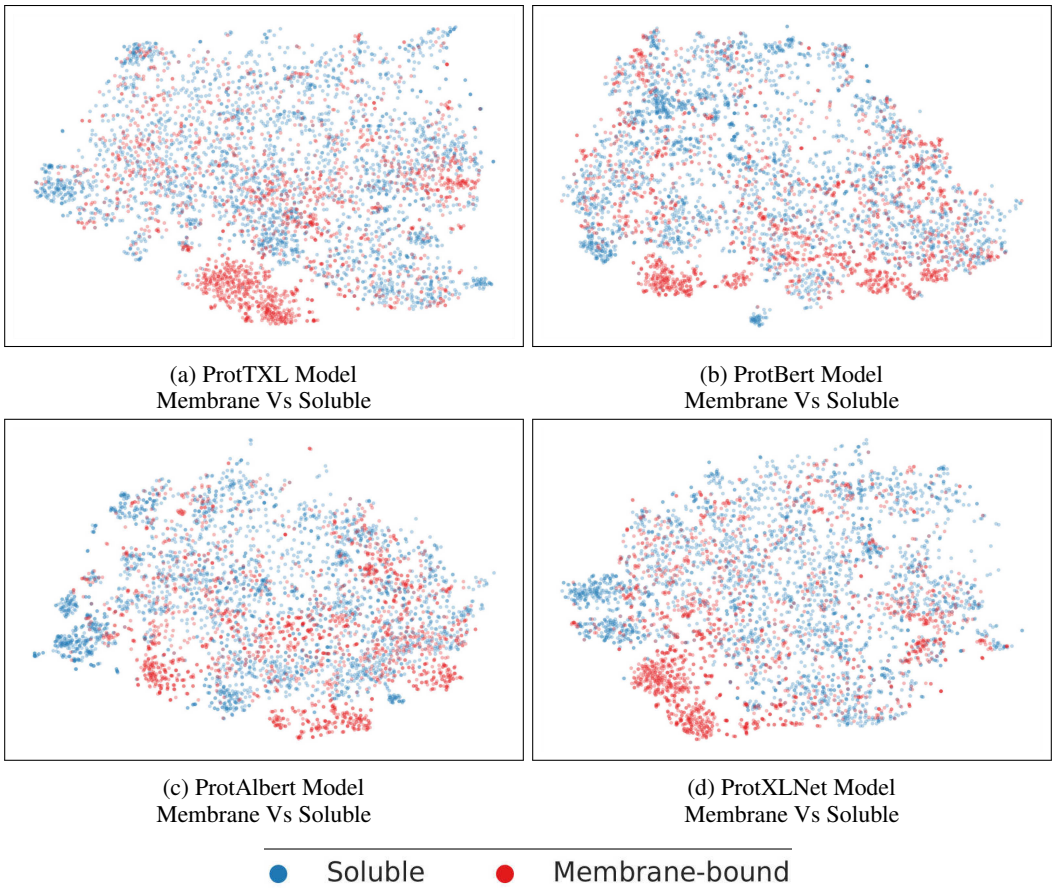
(a) ProtTXL Model Kingdoms

(b) ProtBert Model Kingdoms

(c) ProtAlbert Model Kingdoms

(d) ProtXLNet Model Kingdoms

Figure 13: **Unsupervised training captures aspects specific to the kingdoms of life:** similar to the analysis of structural and functional features of proteins ((Fig. 7 and Fig. 12), the SCOPe dataset redundancy reduced at 40% sequence identity was used to analyse whether the language models (LMs) trained here were able to learn features specific to the different kingdoms of life. If no such annotation was available for a protein, it was excluded from this analysis. Protein representations were again generated by mean-pooling over the representations extracted from the last layer of each model (average over the length of the protein) and t-SNE was used to project the high-dimensional representations down to 2D dimensions. Many small, local clusters share function as given by the main classes in the Enzyme Commission number (E.C.).



(a) ProtTXL Model
Membrane Vs Soluble

(b) ProtBert Model
Membrane Vs Soluble

(c) ProtAlbert Model
Membrane Vs Soluble

(d) ProtXLNet Model
Membrane Vs Soluble

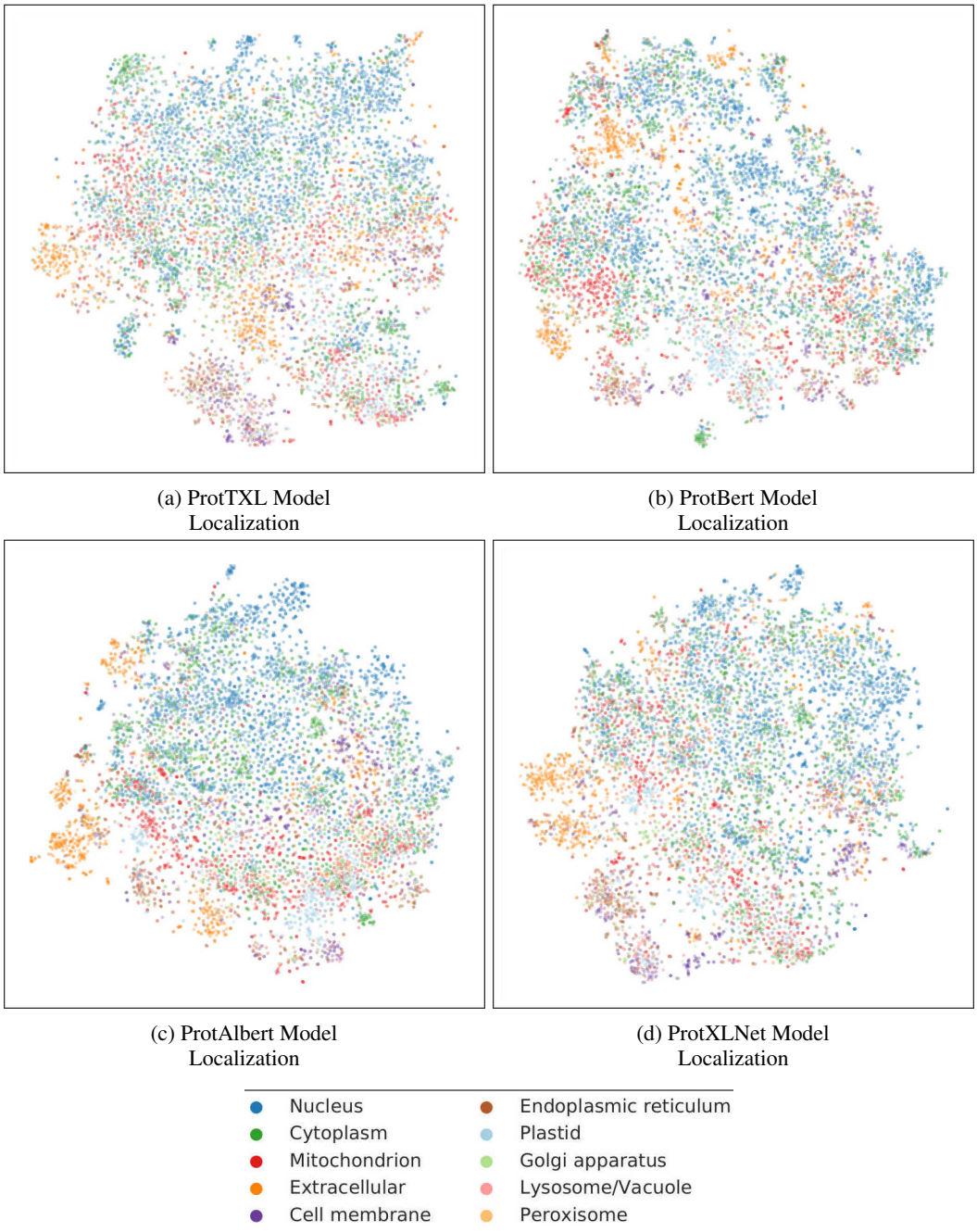
Figure 14: **Language models learn to distinguish between soluble and membrane-bound proteins:** in order to assess whether the language models (LMs) trained here learnt to differentiate between soluble and membrane-bound proteins, the Deeploc dataset [26] was used again. Protein embeddings were again generated by mean-pooling over the representations extracted from the last layer of each LM. T-SNE projections of the high-dimensional representations suggests that embeddings capture aspects of proteins without ever seeing labels of membrane, i.e. without supervised training.

them to pick up the very different biochemical properties of proteins located in (or attached to) a membrane and soluble proteins.

Localization The 2 dimensional t-SNE projection of the deeploc dataset [26] was also used to color proteins according 10 different subcellular compartments. As shown in Fig. 15, among the easiest to distinguish clusters are Peroxisomes, Nucleus and Cytoplasm.

-1.3 Protein LM inference speed

The effect of varying sequence lengths (128, 256, 512) and different batch sizes (1, 16, 32) on the inference time of the protein LMs introduced here is reported in table 2.



(a) ProtTXL Model Localization

(b) ProtBert Model Localization

(c) ProtAlbert Model Localization

(d) ProtXLNet Model Localization

Figure 15: **Language models learn aspects of protein subcellular localization without labelled data:** proteins and localization annotation were taken from the Deeploc dataset [26] to assess whether the language models (LMs) trained here learnt to distinguish proteins depending on the subcellular compartment they occur in. Towards this end, fixed-size protein representations were again derived by mean-pooling over the embeddings extracted from the last layer of each LM. The high-dimensional representations were projected to 2D using t-SNE and proteins were colored according to their localization annotation. The figures suggests that LMs learnt to capture aspects of proteins without ever seeing labels of localization, i.e. without supervised training.

Model		**BioTXL**	**BioBert**	**BioAlbert**	**BioXLNet**	**SeqVec**
Sequence Length	**Batch Size**					
512	1	0.10	0.06	0.24	0.10	0.92
	16	0.11	0.05	0.24	0.14	0.07
	32	0.11	0.05	0.25	0.14	0.04
256	1	0.05	0.03	0.16	0.04	0.46
	16	0.04	0.02	0.11	0.04	0.03
	32	0.04	0.02	0.11	0.05	0.02
128	1	0.03	0.03	0.08	0.03	0.24
	16	0.02	0.01	0.06	0.01	0.02
	32	0.01	0.01	0.06	0.02	0.01
Average	1	0.06	0.04	0.16	0.06	0.54
	16	0.05	0.03	0.14	0.07	0.04
	32	**0.05**	**0.03**	**0.14**	**0.07**	**0.02**

Table 2: **Comparison of inference speed:** The analysis distinguished proteins of different length, as well as different batch sizes (numbers of proteins processed: 1, 16 and 32; cap at 32 due to limitation of GPU memory to 12GB vRAM). For simplicity, no proteins longer than 512 is shown . Each test was repeated 100 times and the average time per protein was reported. The experiment was conducted using a single Nvidia Titan V GPU.