Chapter 16

LMPTMSite: A Platform for PTM Site Prediction in Proteins Leveraging Transformer-Based Protein Language Models

Pawel Pratyush, Suresh Pokharel, Hamid D. Ismail, Soufia Bahmani, and Dukka B. KC

Abstract

Protein post-translational modifications (PTMs) introduce new functionalities and play a critical role in the regulation of protein functions. Characterizing these modifications, especially PTM sites, is essential for unraveling complex biological systems. However, traditional experimental approaches, such as mass spectrometry, are time-consuming and expensive. Machine learning and deep learning techniques offer promising alternatives for predicting PTM sites. In this chapter, we introduce our LMPTMSite (language model-based post-translational modification site predictor) platform, which emphasizes two transformer-based protein language model (pLM) approaches: pLMSNOSite and LMSuccSite, for the prediction of S-nitrosylation sites and succinylation sites in proteins, respectively. We highlight the various methods of using pLM-based sequence encoding, explain the underlying deep learning architectures, and discuss the superior efficacy of these tools compared to other state-of-the-art tools. Subsequently, we present an analysis of runtime and memory usage for pLMSNOSite, with a focus on CPU and RAM usage as the input sequence length is scaled up. Finally, we showcase a case study predicting succinylation sites in proteins active within the tricarboxylic acid (TCA) cycle pathway using LMSuccSite, demonstrating its potential utility and efficiency in real-world biological contexts. The LMPTMSite platform, inclusive of pLMSNOSite and LMSuccSite, is freely available both as a web server (http://kcdukkalab.org/pLMSNOSite/ and http://kcdukkalab.org/LMSuccSite/) and as standalone packages (https://github.com/KCLabMTU/pLMSNOSite and https://github.com/KCLabMTU/LMSuccSite), providing valuable tools for researchers in the field.

Key words Post-translational modification (PTM), Protein language models, Transformer, Natural language processing, Deep learning, CPU runtime, RAM usage, Succinylation, S-nitrosylation, Tricarboxylic acid (TCA) cycle

1 Introduction

Proteins, one of the most critical organic compounds produced within living organisms' cells, are manufactured when specific genes are first transcribed into mRNA and then later translated into chains of amino acids through the process called as translation.

Andrzej Kloczkowski et al. (eds.), *Prediction of Protein Secondary Structure*, Methods in Molecular Biology, vol. 2867,
https://doi.org/10.1007/978-1-0716-4196-5_16,

The translation process occurs within ribosomes and results in the assembly of proteins from amino acids. Post these stages, proteins may undergo various biochemical modifications to expand their functional roles within biological activities. These modifications, known as post-translational modifications (PTMs), involve the severance of bonds between specific amino acids in proteins and the subsequent addition of functional biochemical groups such as methyl, phosphoryl, acetyl, glycosyl, nitrosyl, succinyl, or ubiquitin. Once synthesized, most proteins are transported to the endoplasmic reticulum (ER), where PTMs activate them. These well-folded and functional proteins are then moved to the Golgi body and eventually secreted through exocytosis, performing critical roles as secretory or plasma membrane proteins. Other proteins, such as histones found in the cytosol and nucleus, may also undergo PTMs to maintain cellular functions.

PTMs were initially studied for their role in the enzymatic activity of kinases. However, they are now known to be involved in nearly all cellular activities, including gene regulation, localization, signal transduction, and interaction with other biochemical molecules such as proteins, nucleic acids, and lipids. Consequently, PTMs are implicated in the pathogenesis of numerous diseases, including cancers and cardiovascular diseases, and in a wide range of cellular-level biological activities. Research into PTMs and the identification of modification sites is a burgeoning field, with both biologists and computational scientists working in tandem to identify these sites. Although around 400 PTMs exist [1]; some of the well-studied PTMs include phosphorylation, succinylation, S-nitrosylation, acetylation, ubiquitination, methylation, glycosylation, and SUMOylation. Owing to their roles in virtually all cellular biological activities, both normal and abnormal, PTMs are a heavily researched field in biology. The advent of immunoprecipitation, mass spectrometry, and high-throughput sequencing technologies has facilitated large-scale detection of PTMs and data collection. This data is stored in databases for experimentally validated PTM sites, such as Swiss-Prot [2], PhosphoELM [3], O-GLYCBASE [4], dbPTM [5], PTMcode [6], PTMCuration [1], PhosphoSitePlus [7], etc. The dbPTM [5] database alone has amassed more than two million experimentally validated PTM substrate sites.

This ever-increasing accumulation of PTM data has spurred the development of computational methods, both in terms of algorithms and tools, to handle and analyze this wealth of information [8]. Laboratory detection of PTM sites via mass spectrometry is not only costly but also labor-intensive and time-consuming. Computational methods have consistently proven their efficacy in detecting various types of PTM sites and their functional significance, enabling researchers to prioritize targets for functional validation. Machine learning tools for PTM site detection rely on features extracted from the targeted protein sequences. The first generation

of computational tools in this domain focused on deriving physicochemical and structural properties of amino acids from protein sequences and applying shallow machine learning algorithms such as support vector machines (SVM) and random forest [9]. These properties may encompass a broad spectrum of information, including physicochemical, evolutionary, structural, autocorrelational, and compositional data, all contributing to patterns that distinguish between positive sites (where modifications occur) and negative sites (where modifications do not occur). These features may include amino acid composition (AAC); pseudo-amino acid composition (PseAAC); composition, transition, and distribution (CTD); solvent accessibility of amino acids; evolutionary features based on entropy and position-specific scoring matrix (PSSM); and more.

One of the significant challenges in PTM site prediction is the site-specific nature of modifications. For instance, phosphorylation only affects serine (S), threonine (T), and tyrosine (Y). However, not all occurrences of these three amino acid residues in a protein sequence are always affected, but only at specific positions, which complicates prediction. This challenge also extends to other PTM sites, making their prediction difficult. Machine learning-based approaches typically face numerous challenges centered around feature extraction that provides sufficient discriminatory information and the algorithms that synthesize this information for knowledge learning and prediction. The probabilistic nature of machine learning prediction sparks debate regarding the efficacy of the prediction method. Nevertheless, researchers strive to minimize false discovery errors by using a large number of experimentally verified instances, eliminating biases, applying validation metrics of varying interpretations, and leveraging powerful algorithms.

Recently, the field of PTM site prediction has evolved with the advent of emerging variants of deep learning and language models [8], including transformer-based models. Additionally, the field of natural language processing (NLP) has seen remarkable advancements with sequential models such as recurrent neural network (RNN) [10], long short-term memory (LSTM) [11], and convolutional neural network (CNN) [12]. These advanced algorithms have revolutionized the ability of machines to understand and generate human language. RNNs and LSTMs are adept at capturing temporal dependencies, enabling tasks like language modeling and sentiment analysis. CNNs, originally designed for computer vision, have demonstrated success in analyzing sequential data, particularly for text classification and sentiment analysis. Transformers, with their attention mechanisms, have reshaped the NLP landscape, bringing about breakthroughs in language understanding, question answering, and text classification.

The application of natural language processing (NLP) to the life sciences has garnered significant attention and recognition within the scientific community. Given the sequential nature common to nucleotide sequences and natural languages, extensive efforts have been put into developing frameworks that utilize shared mechanisms between these domains. As a result, substantial advancements have been made in developing NLP-based encoding techniques for representing protein sequences. The use of sequence encoding techniques derived from natural language processing, such as one-hot encoding, embedding (including Word2Vec [13] and GloVe [14], BERT [15]), and other generative models, has increased for protein sequences. These methods offer potential solutions for a wide range of downstream protein tasks by capturing semantic relationships and contextual information. Whether representing amino acids as binary vectors, mapping them to continuous vector spaces, or leveraging transformer architectures, these techniques enable protein analysis and understanding, with implications for drug discovery, protein engineering, and systems biology research. Please refer to Pokharel et al. [16] for the in-depth details of various NLP-based encoding techniques of protein sequences for the PTM site prediction.

The applications of these emerging NLP technologies for PTM site prediction are elevated by the resemblance of biological sequences to text. Since the first transformer model [17], various families of transformer-based model have been developed. More recently, language models like GPT (Generative Pre-trained Transformer) [18], BERT (Bidirectional Encoder Representations from Transformers [15], and T5 (Text-to-Text Transfer Transformer) [19] are examples of transformer-based models that have attained state-of-the-art performance in various NLP tasks. These models are pre-trained on large text corpora, allowing them to learn general language representations, which can then be fine-tuned for specific tasks. The principles behind attention mechanisms and transformers have been applied to the field of protein language modeling to predict and analyze protein sequences, structures, and functions. Analogous to how natural language models learn the grammar and context of human language, protein language models aim to learn the "grammar" of amino acid sequences that form proteins, capturing patterns and relationships within these sequences. These models work by applying techniques from natural language processing and deep learning to learn the underlying patterns and relationships within protein sequences. These models are trained on large databases of protein sequences, treating amino acids as analogous to words in human language. By employing techniques such as positional embeddings, multi-head attention mechanisms, and masked language modelling (MLM), protein language models can effectively capture both local and global

sequence features that contribute to the structure and function of proteins.

Building on this foundation, pre-trained protein language models can be fine-tuned to predict post-translational modifications (PTMs) in protein sequences by leveraging transfer learning. The general knowledge of protein sequences acquired during pre-training enables these models to capture the context and patterns of amino acid sequences surrounding the modification sites. This ability to recognize local and global sequence features, along with the use of task-specific loss functions, aids in the identification of PTM sites with higher accuracy.

The recent emergence of these large protein language models and the potential of these models to distill information from the sequences unleash various opportunities to use them as feature extractors in subsequent downstream prediction tasks. Motivated by the development of these language models and their ability to distill global contextual information, we developed various tools [20–23] for predicting different types of PTM sites. We refer to this platform of various PTM site prediction tools as LMPTMsite (language model-based PTM site predictor). This chapter primarily focuses on two tools within the LMPTMSite platform: pLMSNOSite [20] and LMSuccSite [21], tools that have demonstrated superior performance in predicting S-nitrosylation and succinylation sites, respectively. We discuss these tools in depth, focusing on the following aspects: the benchmark datasets used (Subheading 2.2), sequence encoding based on language models (Subheading 2.3), the underlying deep learning (DL) architectures (Subheading 2.4), and the protocols for model evaluation and training (Subheadings 2.5 and 2.6). Subsequently, we recap the comparative performance results from the tools' literature, emphasizing the comparison of different DL architectures, various pLMs, and other widely available tools (Subheading 3.1). We also provide comprehensive instructions for using the web servers and standalone programs, as well as interpreting their outputs (Subheadings 3.2 and 3.3). In addition, we conduct a new study in this chapter, reporting on the tools' resource utilization, including CPU and RAM usage as a function of protein sequence length (Subheading 3.4). We conclude with a case study where LMSuccSite is used to predict succinylation sites in the tricarboxylic acid (TCA) cycle pathway (Subheading 3.5) and a discussion on the limitations of our study and potential future work (Subheading 3.6). Accompanying the detailed methodology, we provide helpful notes to guide readers in developing their own machine learning-based approaches for bioinformatics problems.

2 Materials and Methods

In this section, we describe our recently developed two tools within the LMPTMSite platform: pLMSNOSite [20] and LMSuccSite [21], designed for the prediction of S-nitrosylation and succinylation sites, respectively. We delve into the particulars of the benchmark dataset used, the process of sequence encoding (also referred to as feature extraction), the underlying deep learning architectures, the intricacies of model training, as well as the evaluation strategy and metrics employed.

2.1 S-nitrosylation and Succinylation

S-nitrosylation (SNO) and succinylation are some of the vital protein post-translational modifications (PTMs) that have profound regulatory effects on cellular functions. SNO is characterized by the covalent attachment of a nitric oxide (NO) group to the thiol side chain of cysteine (C) residues within proteins (Fig. 1a). This PTM is of great biological significance as it modulates protein function and plays a significant role in cellular signal transduction. Recently, it has been discovered that the protein SNO can contribute to protein misfolding events that are typically observed in neurodegenerative diseases. Consequently, abnormality in SNO has been linked to numerous neurodegenerative diseases, including Alzheimer's disease and Parkinson's disease, where abnormal protein folding patterns are a common hallmark. This highlights the critical need for continued research into SNO, as it can not only enhance our understanding of the molecular mechanisms underlying these diseases but also aid in the development of potential therapeutic strategies [24].

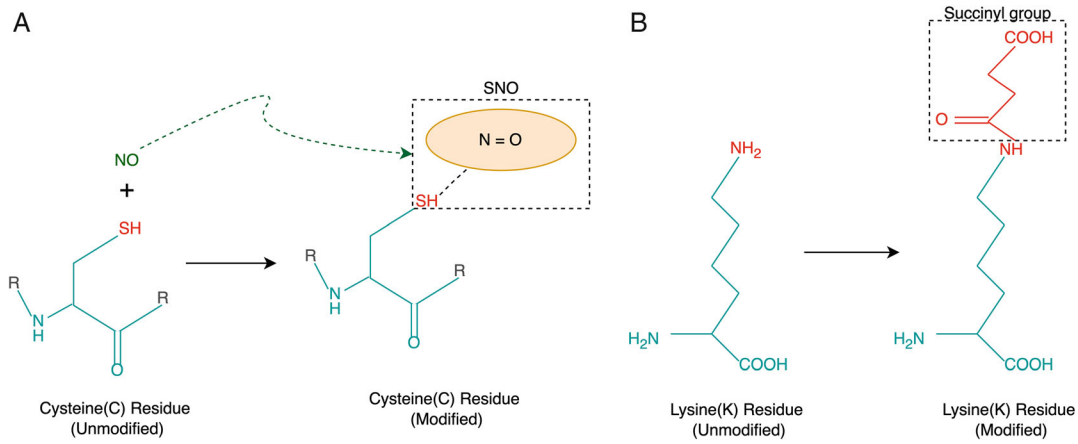

Fig. 1 (**a**) Illustration depicting S-nitrosylation modification on a cysteine (C) residue, (**b**) illustration depicting succinylation modification on a lysine (K) residue

On the other hand, succinylation involves the modification of lysine (K) residues in a protein by attaching a succinyl group (-CO-CH_2-CH_2-CO_2H) (Fig. 1b). This PTM has garnered significant attention due to its integral roles in enzyme function and metabolic pathways. Recent research has illustrated that succinylation can fundamentally alter the enzymatic rates of proteins and influence the pathways they operate in. Predominantly, succinylation targets the mitochondrial metabolic pathways, making it an essential mechanism for the regulation of metabolism and signaling pathways. Additionally, these alterations have implications that extend beyond normal cellular function. Dysregulation in succinylation has been associated with a variety of health conditions and diseases, including, but not limited to, tumors, cardiometabolic diseases, liver metabolic disorders, and various nervous system diseases. Therefore, understanding succinylation and its implications can aid in developing therapeutic strategies for these diseases [25].

Given the crucial role these PTMs play in cellular function and disease, predicting SNO and succinylation sites using computational methods can be immensely beneficial. Such prediction tools can expedite the identification of modification sites, thus accelerating our understanding of their functional implications. Moreover, computational prediction can streamline experimental designs and help prioritize targets for experimental validation, making it a cost-effective strategy. These advantages have the potential to significantly speed up both basic research and the development of novel therapeutic strategies targeting aberrant PTMs in a range of diseases.

Several computational tools have been developed for predicting these sites (*see* **Note 1**). Below, we provide a brief summary of these existing approaches, with a focus on the feature encoding approach and computational method utilized in each case (see Table 1).

2.2 Creation of Benchmark Datasets

The quality and diversity of datasets are of paramount importance in ensuring the reliable performance of machine learning and deep learning models. Thus, it is crucial to have high-quality datasets that accurately represent the problem space, encompassing a wide array of patterns and variations within the dataset. For the creation of a robust dataset geared toward PTM site prediction, robust preprocessing steps should be executed before proceeding to the modeling stage.

The datasets used for building our tools are carefully curated from sources that have been used by several state-of-the-art tools. These datasets have undergone rigorous preprocessing, following the steps outlined in the notes (*see* **Notes 2**–**5**), to ensure their quality and reliability. Subsequently, homologous sequences have been removed using the CD-HIT algorithm [26], which is a greedy incremental clustering algorithm that groups sequences based on a predetermined sequence identity threshold. This helps to reduce

Table 1
Summary of existing tools for the prediction of S-nitrosylation and succinylation sites

Existing tools	PTM type	Published year	Encoding method	Computational method
GPS-SNO [40]	S-nitrosylation (SNO)	2010	BLOSUM62	GPS2.0 algorithm
iSNO-PseAAC [41]		2013	PseAAC	Conditional random field (CRF) algorithm
SNOSite [42]		2011	Physicochemical properties, positional weighted matrix	SVM
DeepNitro [28]		2018	One-hot encoding, PSSM, SAC, PFR	ANN
PreSNO [27]		2019	CPA, SAC, PSSM, physicochemical properties	SVM
DeepSuccinylSite [43]	Succinylation	2018	Word embedding	CNN
GPSuc [35]		2018	AAindex, AAC, PSSM, pCKSAAP	RF, logistic regression
SuccineSite2.0 [29]		2017	pbCKSAAP, orthogonal binary features	SVM
pSuc-FFSEA [44]		2022	EBGW, One-Hot, CBOW, CGR, and AAF_DWT	SVM, BLS, LightGBM, LR

biases during model training. Duplicate or contradicting sites and overlapping sequences between the training and independent testing sets have been removed to maintain the distinctness of each set. Lastly, to handle data imbalance, random undersampling (RUS) of the training set is performed, which helps preserve the distribution between the training and testing sets. It is important to note that pLMSNOSite [20] and LMSuccSite [21] also underwent imbalance learning via cost-sensitive (CS) learning (*see* **Note 6**). However, the results were suboptimal, potentially attributable to the presence of false negatives when considering the entire negative set. Please refer to the supplementary materials of the literature of the respective tools for details about the cost-sensitive learning.

2.2.1 S-nitrosylation Dataset

The SNO dataset has been sourced from PreSNO [27], which is derived from the original DeepNitro [28] dataset, a dataset compiled through a rigorous literature search for experimentally validated SNO sites. This includes 4762 sites that have been confirmed experimentally across 3113 protein sequences. The CD-HIT algorithm [26] was employed to eliminate homology, with a similarity cut-off of 30%, resulting in 3734 positive sites. The remaining cysteine (C) residues from the same protein sequences not annotated as positive were designated as negative SNO sites, with a total of 20,548 negative sites (*see* **Note 7**). A further refinement was conducted by removing any negative sites that matched a window

sequence in the positive set, resulting in 20,333 negative sites. An independent test set was then created by randomly selecting 20% of the total sites, with the remaining data utilized to create the training set. The training set consisted of 3383 SNO sites and 17,165 non-SNO sites, while the independent test set consisted of 351 SNO sites and 3168 non-SNO sites. Given the imbalance between positive and negative sites in the training set, random undersampling (RUS) of non-redundant negative sites was employed to mitigate modeling bias, while the independent test was left as it is to mimic the distribution of positive and negative sites in the real-world scenario.

2.2.2 Succinylation Dataset

This dataset was constructed from experimentally verified succinylation sites provided by Hasan et al. [29], which were acquired from the UniProtKB/Swiss-Prot database [30] and the NCBI protein sequence database. Initially, the sequences were subjected to homology removal using a CD-hit algorithm [26] with a similarity cut-off of 30%, resulting in 5009 succinylated sites from 2322 protein sequences. Thereafter, the sequences were randomly partitioned into a training set (2192 protein sequences) and a testing set (124 proteins), containing 4755 and 254 succinylation sites, respectively. Negative sites were subsequently derived from the same protein sequences, with all lysine residues (K) from the same protein sequences that were not annotated as succinylated sites (*see* **Note 7**). This resulted in 50,565 negative succinylation sites in the training set and 2977 negative succinylation sites in the test set. To address the imbalanced training set, Random Under Sampling (RUS) was used on the negative training set to obtain an equal number of negative sites as the positive sites in the training set. To evaluate the performance of the model in a real-world scenario, the independent test set was left as it is obtained from 124 proteins resulting imbalance in the number of positive and negative sites. The two datasets are summarized in Table 2.

2.3 Sequence Encoding aka Feature Extraction

The process of converting amino acid sequences into numerical space representations presents a significant challenge when developing deep learning-based predictors for PTM sites. An effective vector space encoding should capture crucial sequence features while minimizing noise and redundancy. The predictive models' performance and generalizability are directly influenced by the quality of encoding, particularly its capacity to handle variations in sequence length and context. Various techniques for encoding protein sequences have been devised and employed in numerous downstream prediction tasks, including PTM prediction. As discussed in the introduction, traditional manual feature extraction methods, typically based on physicochemical properties, require a significant amount of manual labor and expertise. It is a time-consuming and complex task to select an optimal set of

Table 2
Description of the training and independent test datasets

PTM	Dataset type	Site status	#Proteins	#Sites	Total sites
S-nitrosylation (SNO)	Train	Positive (SNO)	1962	3383	20546 (after balancing:6766)
		Negative (non-SNO)	340	17165 (after balancing: 3383)	
	Independent test	Positive (SNO)	267	351	3519
		Negative (non-SNO)	231	3168	
Succinylation	Train	Positive (succinylated)	2192	4750	55315 (after balancing: 9500)
		Negative (non-Succinylated)		50565 (after balancing: 4750)	
	Independent test	Positive (succinylated)	124	254	3231
		Negative (non-succinylated)		2977	

physicochemical features that can effectively represent the properties and behaviors of amino acids in protein sequences. Additionally, existing tools have relied mainly on peptide sequences, which restrict the feature extraction to only local information, hence potentially missing out on crucial global information that might be embedded in the larger protein sequence context. To overcome these limitations, we adopted two types of embedding techniques for our tools, each addressing different levels of sequence context: the first technique involves employing unsupervised transformer-based protein language models (pLMs) to embed dynamic global contextual information influencing the site of interest, while on the other hand, we utilize a supervised word embedding layer trained on peptide sequences to emphasize the local context of the site of interest. By combining these two encoding techniques, we can capture a wide range of sequence dependencies, from local to global, without the need for manual feature extraction. This hybrid approach enhances the representation of the protein sequences, leading to improved prediction performance as evidenced by the results achieved by our tools, pLMSNOSite and LMSuccSite. The details of these encoding techniques are described below.

2.3.1 Embeddings from Protein Language Models (pLMs)

As discussed in the Introduction section, language models (LMs) have transformed the landscape of natural language processing (NLP) by learning embeddings directly from expansive, unlabeled datasets of natural language. Unlike traditional uncontextualized word embeddings, which allocate a fixed embedding for a word regardless of its surrounding context, embeddings generated by LMs offer contextualized understanding, adjusting based on the words adjacent to them. This breakthrough in NLP has now found its way into the realm of proteins through the development of protein language models (pLMs).

In pLMs, each amino acid is mapped to a vector of fixed length via an embedding layer. The model also applies position embedding to encode the relative positions of each amino acid within its respective protein sequence. A third type of embedding, segment embedding, is used to differentiate between distinct protein sequences. The integrated use of token (amino acid), position, and segment embeddings allows pLMs to provide more than just a non-contextual mapping of amino acids to the vector space. It also captures the dependencies of amino acids within each protein sequence and the contextual relationships between different protein sequences.

The pLMs have demonstrated remarkable success in various downstream bioinformatics prediction tasks. The wealth of information learned by these models can be transferred to other tasks by generating embeddings from them and subsequently using them as inputs to predict other properties of proteins. This has opened up a whole new avenue of research in the field of bioinformatics, as researchers can now leverage the power of pLMs for a wide range of applications. In our work, we employed three prominent protein language models (pLMs)—ProtT5, ESM-1b, and ProtBert. These pLMs have demonstrated significant success in various downstream tasks, such as protein–protein interaction prediction, protein function prediction, and subcellular localization prediction. The following subsections provide a brief introduction to each of these pLMs.

(a) ProtT5-XL-UniRef50

ProtT5-XL-UniRef50 (also ProT5-XL-U50) [31] (referred to as ProtT5 onwards) has been pre-trained by Rostlab in a self-supervised fashion, utilizing Google's T5 (t5-3b version) transformer architecture. Its operational premise involves the prediction of masked or absent amino acids using a vast collection of protein sequences for training. The Uniref50 dataset, employed for training ProtT5, encompasses 45 million protein sequences, which are composed of 15 billion amino acids. This enormous corpus allows the model to implicitly grasp the structural and functional interconnections among various protein types.

The attention mechanism of ProtT5 is built on a stack of 24 transformer layers. Each of these layer houses 32 attention heads and has a final hidden layer size of 1024 units (thus producing feature vector of size 1024) (*see* **Note 8**). This stacked arrangement allows for each layer to operate sequentially on the output generated by the layer that precedes it. By iteratively blending word embeddings in this way, ProtT5 succeeds in generating rich representations as the input travels through the model's deepest layers.

(b) ESM-1b

Evolutionary Scale Modeling (ESM) [32] is another example of a pre-trained language model for proteins, released by Meta. Among its different versions and variants, ESM-1b utilizes a transformer architecture with 33 layers, featuring 650 million trainable parameters, underscoring its potential for complex pattern recognition. Like ProtT5, ESM-1b is also trained using the UniRef50 dataset, thus inheriting the wide scope of 45 million protein sequences represented therein. Operating on an input of protein sequences, ESM-1b generates feature vectors of 1280 dimensions for each individual amino acid (*see* **Note 8**).

(c) ProtBERT-UniRef100

ProtBERT-UniRef100 [31] is a protein language model based on the BERT architecture, trained by Rostlab using a self-supervised approach on the expansive Uniref100 dataset, which includes 217 million protein sequences. Its architecture comprises 24 attention layers that generate an output embedding of 1024 dimensions from the final layer (*see* **Note 8**).

We summarize these three pLMs based on the architecture, number of layers, number of attention heads, training strategy, etc. in Table 3.

2.3.2 Various Approaches for Extracting Embeddings from pLMs

As mentioned earlier, these language models take a protein sequence (or peptide) as input and provide contextual per residue embeddings. Depending upon the input to these language models and/or whether only the embeddings of the particular residue or windows of residues is considered, the approaches can be divided into various categories. Based on the input to the pLM, the approaches for extracting embedding can be broadly classified into three categories (Fig. 2): (a) full-sequence-based, (b) window sequence-based, and (c) multiple windows of varying length.

(a) Full Sequence-Based Embeddings

This class of approaches leverages the entire protein sequence as input to the pLM to extract features. These approaches can be further divided into two types based on whether the output features are generated for the site of interest only or the window around the site of interest.

(i) *Full Sequence-Based Per-Residue Embeddings (FSPE)*

In this method, for each amino acid residue, the input to the pLM is the entire protein sequence. Subsequently, the feature vector corresponding to only the residue of interest (target residue) is extracted for constructing classifiers. If the feature vector generated from pLM has a size of L, then the feature-length for each instance (or site) will be represented as $1 \times L$. This approach incorporates global

Table 3
Summary of the three protein language models (pLMs) utilized in this study

pLM	Based transformer architecture	#Parameters	Dataset for pre-training	#Layers	#Attention heads	Training strategy	Embedding dimension per residue *(1×L)*
ProtT5 [31]	T5	3 B	Uniref50	24	32	Masked Language Modelling (MLM) Objective	1 × 1024
ESM-1b [32]	RoBERTa	650 M	Uniref50	33	12	MLM	1 × 1280
ProtBERT [31]	BERT	420 M	Uniref100	30	16	MLM	1 × 1024

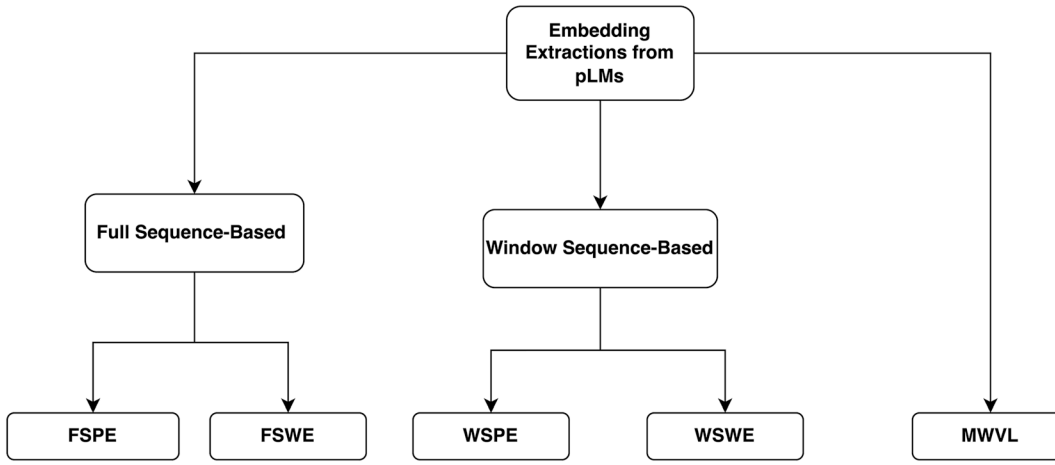

Fig. 2 Various approaches for extracting features from pLMs

contextual information from the protein sequence while focusing on specific sites of interest.

(ii) *Full Sequence-Based Window Embeddings (FSWE)*

This approach bears similarity to FSPE; however, it incorporates a key distinction. Instead of focusing solely on the embedding of the site of interest, a fixed-length window centered around this site is employed. The feature vectors of all residues within this window are extracted (either by concatenation or averaging) for classifier construction. Consequently, the feature-length for each instance (or site) is represented as $W \times L$, where W denotes the window size. The integration of local context information around the site of interest potentially improves the model's performance in capturing residue interactions.

(b) Window Sequence-Based Embedding

This class of approaches diverges from considering the entire protein sequence by focusing on the window sequence fragment centered around the site of interest as input to the pLM.

(i) *Window Sequence-Based Per Residue Embeddings (WSPE)*

Once the fragment is isolated by taking a fixed-size window around the target residue, feature vectors are extracted for each of the residues within it. However, only the feature vector of the site of interest (the central residue) is used for modeling purposes. As a result, the feature-length for each instance remains $1 \times L$. This approach emphasizes local context information, which can be beneficial in capturing residue interactions within the immediate vicinity of the site of interest.

(ii) *Window Sequence-Based Window Embeddings (WSWE)*

In this approach, the focus is on the window sequence fragment centered around the site of interest, similar to WSPE. However, instead of only considering the feature vector of the central residue, the feature vectors for all residues within the fragment are taken into account. These feature vectors collectively represent the local context and interactions around the site of interest. Consequently, the feature length for each instance (or site) becomes W×L, where W denotes the window size.

(c) Multiple Windows of Varying Length (MWVL)

The multiple windows of varying length (MWVL) technique provides a versatile method for PTM prediction by accommodating diverse protein sequences and residue interactions through its adaptability to different scales. Essentially, this approach employs multiple windows of varying lengths centered around the site of interest to capture information at different scales. For each window length, embeddings for all residues within the window are generated and aggregated to create a single representation. The aggregated embeddings from all window sizes are then concatenated to form the final feature vector for the site of interest. The feature-length of each instance (or site) is represented as the total aggregated size of all window embeddings (e.g., $W_1 \times L + W_2 \times L + \cdots + W_n \times L$, where W_i is the size of the ith window and L is the size of the individual residue embeddings).

In both pLMNOSite and LMSuccSite, we utilize the encoder part of the pLMs to extract FSPE-based features (use the entire protein sequence as an input and extract the embeddings only for the site of interest). As a result, each site of interest, whether it is a cysteine (C) in case of SNO or lysine (K) in case of succinylation, is represented using a feature vector of dimension $1 \times L$ (ProtT5, $L = 1024$; ESM-1, $L = 1280$; ProtBERT, $L = 1024$) where L is the length of embedding per residue for each of these pLMs.

2.3.3 Word Embedding

In the architecture of various deep learning models, particularly when dealing with sequential data such as text or, in our case, protein sequences, the Keras embedding layer can be utilized to extract word embedding. In the context of PTM prediction tasks, n-mer window sequences are extracted centered around the site of interest, ensuring an equal number of flanking residues exist on both sides. If there are insufficient residues to form a window, virtual amino acids (X) are padded to maintain sequence length.

Each unique amino acid in these n-mer window sequences is represented by a unique integer in a process known as integer encoding. This integer-encoded representation then serves as the

input to the embedding layer. The crux of the embedding layer's functionality lies in its ability to transform these positive integers (indexes) into fixed-size dense vectors. Each unique index is thereby mapped to a distinct vector, facilitating the encoding of amino acids into a format suitable for ML/DL models.

The initial state of the embedding layer consists of random weights. During the training process, the network adjusts these weights through backpropagation, which aims to minimize the loss function. The outcome is that semantically similar inputs (amino acids in our case) are mapped to proximal points in the high-dimensional vector space. Over time, this process results in a learned dense representation of the input data, where the semantic relationships between different amino acids are encoded in the spatial relationships between their corresponding vectors.

The layer is defined by three salient parameters: input_dim, output_dim, and input_length.

- *input_dim:* This is the size of the vocabulary, essentially the number of unique amino acids (including canonical, non-canonical, and virtual amino acids) in the input dataset.
- *output_dim:* This is the size of the dense vector space into which the inputs are embedded. The optimal value for output_dim can vary and is typically determined through methods like *k*-fold cross-validation. Larger values can capture more detailed representations at the cost of increased computational complexity.
- *input_length:* This is the maximum permissible length of input sequences, in our case the n-mer window size centered around the site of interest in the protein sequence. Again, the optimal value for input_length is determined empirically.

The output of the embedding layer is a 2D vector of dimension $W \times D$, where W represents the sequence (window size) and the second dimension D is the learned dense vector (embedding dimension) representing each unique amino acid for the particular task (*see* **Note 9**). The optimal values of vocabulary size (input_dim), embedding dimension (output_dim), and Window size (input_length) obtained from *k*-fold cross-validation (coupled with exhaustive grid search) in pLMNOSite and LMSuccSite are provided below in Table 4.

2.4 Deep Learning Architectures

Given that only window sequences are fed into the word embedding layer, it may primarily capture the local contextual information surrounding the site of interest. Leveraging the representation obtained by pLMs seems logical as they consider the entire protein sequence to generate embeddings, thereby possessing the capacity to convey the global contextual information that influences the function of the sites of interest. Consequently, the proposed architecture of our tools (pLMNSNOSite and LMSuccSite) is rooted in

Table 4
Various parameters of the word embedding layer selected for pLMSNOSite and LMSuccSite

Tool	Embedding module architecture	Vocabulary size (V)	Embedding dimension (D)	N-mer window sequence length (W)	Output dimension of the embedding layer (W × D)
pLMSNOSite	2D-CNN	23	4	37	37 × 4
LMSuccSite	2D-CNN	21	21	33	33 × 21

an ensemble approach (via stacking). It is designed to integrate the representational capabilities of the protein language model pLM and the supervised word embedding layer, thereby merging both local and global contextual information related to the site of interest.

The overall architecture can be divided into three modules, with each module aiming to learn a specific representation. The first module, known as the *Embedding Layer Module (ELM)*, takes in an n-mer window sequence around the site of interest (followed by integer encoding of the window sequence). The deep learning model underlying this module learns the embeddings each of size $W\,x\,D$, where W is the size of the window and D is the embedding dimension, generated by the supervised word embedding layer, the details of which are discussed in Subheading 2.3.3. Since we are considering a window sequence, we establish a spatial correlation between proximal amino acids surrounding the site of interest within the window. To capture this local interaction of amino acids (spatial correlation), we utilize a two-dimensional convolutional neural network (2D-CNN). There are two key reasons for choosing the CNN architecture over other sequence-based models such as RNN, LSTM, BiLSTM (bidirectional LSTM), etc. Firstly, the computational efficiency of CNN models surpasses that of sequence-based models. Secondly, experiments from *k*-fold cross-validation showed superior performance of the CNN model compared to ANN, LSTM, BiLSTM, and ConvLSTM (convolutional LSTM). The convolution layer in the CNN extracts feature maps from the spatial interaction of the amino acids, while the max-pooling layer captures the most prominent features from these maps, reducing their dimensionality with minimal information loss. Finally, the fully connected (FC) layer employs a feed-forward network to produce a classification inference for the site of interest.

The second module, known as the *pLM Module (PLM)*, processes FSPE-based feature vectors of the site of interest ("C" in the case of pLMSNOSite and "K" in the case of LMSuccSite) obtained from the encoder side of the pre-trained ProtT5 in half-precision mode (*see* **Note 10**). These features have a dimension of $1 \times L$,

where L is 1024 (the length of the feature vector per residue produced by ProtT5). As only the embeddings of the site of interest are considered, a simple artificial neural network (ANN) is utilized to learn these contextualized embeddings. This approach aligns with the findings presented by Villegas-Morcillo et al. [33], which suggest that learning embeddings from pLMs may not necessitate complex architectural designs.

The third and final module, known as the *Stacked Generalization Module (SGM)*, consists of a meta-classifier that performs a stacking ensemble of the embedding layer and ProtT5 modules. A stacking ensemble can be implemented either through intermediate fusion or decision-level fusion (also known as late fusion). In our tools, we utilize an intermediate fusion-based stacking to learn the marginal (or shared) representation of the other two modules. Detailed discussions on various fusion strategies and their drawbacks are described in the Notes (*see* **Notes 11**–**14**). To train the meta-classifier, designed on an ANN architecture, we freeze the 2D-CNN model within the ELM and the ANN model in the PLM. This effectively locks the internal parameters of these models and prevents them from updating during the training of the meta-classifier. Subsequently, predictions are extracted from the final hidden layer of each model. In the case of pLMSNOSite, this amounts to a 4×1 feature vector from the PLM and a 16×1 feature vector from the ELM. Similarly, for LMSuccSite, the corresponding feature vectors are of length 128×1 from the PLM and 16×1 from the ELM. These extracted predictions are then concatenated together to form a unified feature representation. These intermediate predictions, or what we refer to as learned representations, are then utilized to train the meta-classifier. Therefore, the input to the meta-classifier is of length 20×1 in pLMSNOSite and 144×1 in LMSuccSite.

The determination of the optimal architecture for each module was achieved through the utilization of k-fold cross-validation in conjunction with grid search. For detailed information regarding the hyperparameter space and the optimal architecture of various models, please refer to the supplementary materials provided in the individual literature of pLMSNOSite and LMSuccSite. Table 5 provides a comprehensive breakdown of the final architecture specifics, including the type of layers, dimensions of each layer, and the employed activation functions in ELM, PLM, and SGM modules of pLMSNOSite and LMSuccSite.

It should be noted that pLMSNOSite and LMSuccSite utilize intermediate fusion for two primary reasons: (i) we are employing two distinct representations, namely, ProtT5 pLM, which utilizes the entire sequence context (global sequence context), and the embedding layer, which operates on the n-mer window sequence (local sequence context). Early fusion might struggle to capture the differing characteristics of these two modalities. (ii) Merging at the score level may be insufficient in effectively capturing the correlation between these two representations.

Table 5
Architectural details of different modules in pLMSNOSite and LMSuccSite

Module	Architecture	Tool	Input shape	Description	Activation function
ELM	2D-CNN	pLMSNOSite	37×4	Embedding (37×4) - Convolution (64×19×1) - Dropout (0.3) - Max Pooling (5×2) - Flatten (384) - Dense (16) - Output (1)	Rectified Linear Unit (ReLU) activation function across all hidden layers and sigmoid activation in output layers
		LMSuccSite	33×21	Embedding (33×21) - Convolution (32×17×3) - Dropout (0.2) - Max Pooling (2×2) - Flatten (2304) - Dense (16) - Dropout (0.2) - Output (1)	
PLM	ANN	pLMSNOSite	1024×1	Dense (128) - Dropout (0.4) - Dense (16) - Dropout (0.2) - Dense (4) - Output (1)	
		LMSuccSite		Dense (256) - Dropout (0.2) - Dense (128) - Dropout (0.2) - Output (1)	
SGM	ANN	pLMSNOSite	20×1	Dense (8) - Output (1)	
		LMSuccSite	144×1	Dense (16) - Dense (4) - Output (1)	

The symbol "-" denotes the sequence of layers or operations in the model, and values inside the brackets represent the dimensions or shape of specific layers

2.5 Model Evaluation and Performance Measures

A stratified k-fold cross-validation (pLMSNOSite $k = 5$ and LMSuccSite $k = 10$) coupled with a grid-search strategy was implemented for the optimal selection of models and hyperparameters (*see* **Note 15**). Additionally, independent testing was conducted to assess the models on unseen data and benchmark them against other existing methodologies. Performance metrics used for evaluating the accuracy of model predictions against ground truth labels are outlined in Table 6.

2.5.1 Dealing with Class Imbalance

The data distribution in PTM prediction tasks is often skewed toward the negative class, potentially biasing performance estimates toward the negative set. As such, metrics like ACC, SN, FPR, FNR, and AUROC may be significantly impacted and should not be solely relied upon. In our case, the training sets in pLMSNOSite

Table 6
Various performance metrics used for evaluating pLMSNOSite and LMSuccSite

Metric name	Definition	Range[a]
True positive (TP)	Count of correctly predicted PTM sites	$[0, +\infty)$
True negative (TN)	Count of correctly predicted non-PTM sites	
False positive (FP)	Count of incorrectly predicted PTM sites	
False negative (FN)	Count of incorrectly predicted non-PTM sites	
Accuracy (ACC)	$\mathrm{ACC} = \frac{\mathrm{TP}+\mathrm{TN}}{\mathrm{TP}+\mathrm{FN}+\mathrm{FP}+\mathrm{FN}}$	[0, 1]
Sensitivity (SN)/recall/true positive rate (TPR)	$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}}$	
Specificity (SP)/true negative rate (TNR)	$\mathrm{TNR} = \frac{\mathrm{TN}}{\mathrm{FP}+\mathrm{TN}}$	
Fall-out/false alarm rate/type I error/false positive rate (FPR)	$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{FP}+\mathrm{TN}}$	
Miss rate/type II error/false negative rate (FNR)	$\mathrm{FNR} = \frac{\mathrm{FN}}{\mathrm{TP}+\mathrm{FN}}$	
g-mean	$\mathrm{gmean} = \sqrt[2]{\mathrm{SN} \times \mathrm{SP}}$	
Matthews correlation coefficient (MCC)	$\mathrm{MCC} = \frac{(\mathrm{TP} \times \mathrm{TN}) - (\mathrm{FP} \times \mathrm{FN})}{\sqrt[2]{(\mathrm{TP}+\mathrm{FP})(\mathrm{TP}+\mathrm{FN})(\mathrm{TN}+\mathrm{FP})(\mathrm{TP}+\mathrm{FN})}}$	$[-1, 1]$
Area under receiver operating characteristic curve (AUROC)	Area under the curve of sensitivity (TPR) plotted against 1 - specificity (FPR) at various decision threshold cut-offs (range, 0.0–1.0)	[0, 1]
Area under precision–recall curve (AUPR/PrAUC)	Area under the curve of precision plotted against recall (SN) at various decision thresholds cut-offs (range, 0.0–1.0)	

[a]Note: The ranges provided within brackets are general guidelines, but they may vary depending on the specific implementation and context of the metrics

and LMSuccSite were balanced using random undersampling, mitigating the potential effects of class imbalance on cross-validation evaluations. However, the independent test datasets employed in both these tasks were considerably imbalanced (majority-to-minority ratios, ~9 for pLMSNOSite, ~12 for LMSuccSite). Consequently, we focused on performance measures less affected by class imbalance, such as MCC, g-mean, and AUPR. We considered all these metrics in conjunction to gain a comprehensive view of our model's performance on the independent test set.

2.5.2 Dealing with Varying Decision Threshold

In binary classification tasks, a critical process involves determining an appropriate decision threshold cut-off. This threshold transforms the model's raw probability outputs into definitive categorical labels. Depending on the selected threshold cut-off, the value of

computed metrics can vary significantly. The decision of selecting an optimal cut-off value typically relies on the specific use case. In the context of pLMSNOSite and LMSuccSite, we consistently utilized a widely adopted cut-off value of 0.5. However, we emphasized on threshold-independent metrics like AUROC and AUPR when evaluating our models. These metrics are particularly handy in situations where the optimal cut-off point is not known a priori. In addition, our web servers give users the flexibility to select a desired cut-off value within a range of 0.1–0.9 (refer to Subheading 3.3). Similarly, in our standalone version, we have facilitated an easy process to modify the cut-off value prior to running the prediction (see Subheading 3.2).

2.5.3 Dealing with Data Leakage

In order to address the issue of potential target information leakage from base models to the meta-classifier in stacked generalization (*see* **Note 16**), which can result in overestimated cross-validation performance, we have implemented Wolpert's stacking algorithm with *k*-fold cross-validation [34]. This strategy starts with random partitioning of the total training data into *k* folds. Then the base models are trained on *k*-1 of these folds and validated on the remaining one fold. This procedure is performed *k* times, rotating the validation fold each time, and the predictions from base models are gathered as new features. The gathered predictions from each base model train the higher-level meta-classifier, ensuring it is trained on a non-overlapping dataset, thus effectively preventing data leakage and preserving the model selection process' integrity.

2.6 Deep Learning Model Training

All deep learning models in pLMSNOSite and LMSuccSite were trained with the objective of minimizing the binary cross-entropy loss, also known as log loss, represented by the following equation:

$$-\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log y_i' + (1-y_i)\log(1-y_i')\right] \quad (1)$$

In Eq. (1), y_i and y_i' denote the actual and predicted probabilities for the ith instance out of N instances, respectively.

To minimize this loss function, the Adam stochastic optimization method was utilized. In addition, an early stopping mechanism was implemented with a patience value of P_L (range, 0–total number of epochs) halting the training process if there's no loss improvement detected after P_L consecutive epochs. This helps avoid unnecessary computational expenditure and overfitting. The progress of training was continuously scrutinized by monitoring the accuracy/loss curves to further prevent overfitting. The values of major parameters chosen for the Adam optimizer in pLMSNOSite and LMSuccSite are detailed in Table 7 below. This includes learning rate, decay rates, number of epochs, batch size, and patience (P_L) for early stopping, all optimized via a *k*-fold cross-validation strategy with grid search for maximum model performance.

Table 7
Various model training parameters chosen for pLMSNOSite and LMSuccSite

Tool	AMSgrad variant?	Adaptive learning rate	Decay rate		Batch size	No. of epochs	Patience (P_L)
			First moment	Second moment			
pLMSNOSite	Yes	0.001	0.900	0.999	128	200	5
LMSuccSite	No	0.001	0.900	0.999	256	100	7

3 Results

In the subsequent sections, we present the results for our recently developed SNO site predictor, pLMSNOSite [20], and succinylation site predictor, LMSuccSite [21]. In Subheading 3.1, we summarize the *k*-fold cross-validation and independent testing results from the published works [20] [21] on these predictors. In Subheadings 3.2 and 3.3, we provide a comprehensive guide to using the standalone and web server versions of the tools, respectively. In Subheading 3.4, we describe an additional experiments performed in this chapter, where we evaluated the runtime and memory usage of pLMSNOSite as a function of the input sequence length. Finally, in Subheading 3.5, we outline a case study showcasing the application of LMSuccSite in identifying succinylated sites within proteins participating in the TCA pathway.

3.1 k-Fold Cross-validation and Independent Testing Summary

The predictive performance of our approach was assessed in greater details in the original manuscripts [20, 21]. Here, we briefly summarize the results of the k-fold cross-validation experiment, as detailed in the published literature for pLMSNOSite [20] and LMSuccSite [21]. This encompasses all modules (ELM, PLM, and SGM) for both tools, with $k = 5$ for pLMSNOSite and $k = 10$ for LMSuccSite, as shown in Table 8. In the ELM module of both tools, the 2D-CNN emerged as the optimal model, providing a mean MCC of 0.382 ± 0.03 for pLMSNOSite and 0.43 ± 0.02 for LMSuccSite, outshining other architectures such as ANN, LSTM, BiLSTM, and ConvLSTM. In the PLM module, the ANN proved to be the optimal model for both tools, delivering a mean MCC of 0.42 ± 0.03 for pLMSNOSite and 0.47 ± 0.02 for LMSuccSite, outperforming other architectures like 1D-CNN, SVM (support vector machine), RF (random forest), XGBoost (extreme gradient boosting), and AdaBoost (adaptive boosting). As for the SGM module, ANN again surfaced as the optimal model, demonstrating a mean MCC of 0.46 ± 0.03 for pLMSNOSite and 0.56 ± 0.02 for LMSuccSite, superior to architectures like LR (logistic regression), SVM, RF, and XGBoost.

Table 8
***k*-fold cross-validation comparison of various machine learning/deep learning architectures in the modules of the architecture**

Module	Tool	Optimal architecture	ACC	MCC	SN	SP	ROAUC	Compared against[a]
Embedding Layer (ELM)	pLMSNOSite	2D-CNN	0.69	0.38	0.76	0.62	0.73	ANN, LSTM, BiLSTM, ConvLST
	LMSuccSite		0.73	0.47	0.76	0.70	0.79	
ProtT5 (PLM)	pLMSNOSite	ANN	0.71	0.42	0.75	0.67	0.73	1D-CNN, SVM, RF, XGBoost, AdaBoost
	LMSuccSite		0.74	0.47	0.76	0.71	0.80	
Stacked Generalization (SGM)	pLMSNOSite	ANN	0.73	0.46	0.77	0.69	0.80	LR, SVM, RF, XGBoost
	LMSuccSite		0.77	0.56	0.80	0.76	0.85	

Adopted from pLMSNOSite [20] and LMSuccSite [21]
[a]For detailed architectural specifics of each model, please refer to the supplementary materials in the literature corresponding to each respective tool

Table 9
Independent test comparison of various protein language models (pLMs) on the pLMSNOSite dataset

pLM	MCC	Sensitivity	Specificity	g-mean	ROAUC
ProtBERT[31]	0.20	0.48	0.80	0.62	0.64
ESM-1 [33]	0.25	0.56	0.79	0.67	0.69
ProtT5[31]	**0.29**	**0.60**	**0.81**	**0.70**	**0.71**

Adopted from pLMSNOSite [20]

Next, we summarize the comparative study from the original manuscripts [20, 21] where three protein language models (pLMs), ProtBERT, ESM-1, and ProtT5 were used to extract the features. These findings indicated that ProtT5 consistently outperformed the other two pLMs in both pLMSNOSite and LMSuccSite, leading to its choice as a feature extractor in both instances. Table 9 demonstrates the performance differences among these three pLMs in pLMSNOSite on the independent test set. A similar trend was observed in the case of LMSuccSite, with ProtT5 surpassing the other pLMs in performance.

Additionally, we also summarize from the original manuscripts [20, 21] the comparison of pLMSNOSite and LMSuccSite against other existing tools, and these results are presented in Table 10. It can be seen from the table that both these tools perform quite well in comparison to current state-of-the-art tools. Notably, pLMSNOSite demonstrated an impressive 35.0% increase in MCC and a 10.6% improvement in g-mean compared to its closest

Table 10
Comparison of pLMSNOSite and LMSuccSite against other existing tools

PTM	Predictor	ACC	MCC	SN	SP	g-mean
S-nitrosylation	GPS-SNO [40]	0.693	0.014	0.281	0.739	0.455
	iSNO-PseAAC [41]	0.710	0.031	0.287	0.757	0.466
	SNOSite [42]	0.469	0.069	0.668	0.447	0.546
	DeepNitro [28]	0.737	0.222	0.578	0.737	0.653
	pLMSNOSite [20]	0.769	0.340	0.735	0.773	0.754
Succinylation	SuccineSite2.0 [29]	0.85	0.26	0.40	0.88	0.63
	GPSuc [35]	0.85	0.30	0.50	0.88	0.66
	PSuccE [45]	0.85	0.20	0.38	0.89	0.58
	DeepSuccinylSite [43]	0.70	0.27	0.79	0.69	0.74
	LMSuccSite [21]	0.79	0.36	0.79	0.79	0.79

Adopted from pLMSNOSite [20] and LMSuccSite [21]

competitor, PreSNO [27]. Importantly, the pLM-based model alone was able to outperform PreSNO, showing a significant 16.3% increase in MCC. It is worth highlighting that the pLM-based model alone was able to outperform PreSNO with an approximately 16.3% increment in MCC. This underlines the capability of pLM-based embeddings in predicting PTM sites. In a similar vein, LMSuccSite exhibited a 20.0% improvement in MCC and approximately a 19.7% increase in g-mean when compared to GPSuc [35], the next best predictor. These results affirm that our novel approach, which combines global contextual information using pLM and local contextual information using a word embedding layer, is indeed a robust predictor of PTM sites such as S-nitrosylation and succinylation.

3.2 pLMSNOSite and LMSuccSite: Standalone Version

The standalone version of pLMSNOSite is provided on GitHub and can be accessed freely via this link: http://kcdukkalab.org/pLMSNOSite/. The source code and all its dependencies fall under the open-source Apache-2.0 license, providing the freedom to modify and distribute the software.

In the repository, a *requirements.txt* file is provided which details all the necessary libraries with their appropriate versions required to run pLMSNOSite (*see* **Notes 17** and **18**). The instructions for executing pLMSNOSite are outlined below.

Use the following commands to install the libraries and dependencies:

1. *pip install -r requirements.txt*
2. *pip install -q SentencePiece transformers*

After installing all dependencies, one can evaluate the proposed model on the independent test set or run your own predictions. To evaluate the model on the independent test set:

1. Navigate to the *data/test/folder* where the test sequences and corresponding ProtT5 features are placed.
2. Run the command *python3 evaluate_model.py.*

To predict using your own sequence:

1. Place the FASTA file, which should contain the protein sequence data, in the directory *input/sequence.fasta*.
2. Run the script for generating predictions with the command *python3 predict.py.*
3. Once the prediction process is complete, the results will be stored in the *output/ folder.* The output file will provide predictions for each cysteine (C) residues present in the input sequence.

The screenshot of pLMSNOSite repository on GitHub with its command line interface (CLI) is shown in Fig. 3. LMSuccSite's setup process on a local system shares a similar set of procedures to those necessary for establishing pLMSNOSite. To begin with, one should clone the LMSuccSite repository from GitHub (https://github.com/KCLabMTU/LMSuccSite) and ensure that one is covered under the Apache-2.0 license. Then, one needs to adhere to the instructions similarly outlined in the pLMSNOSite setup and its implementation steps.

It is important to highlight that pLMSNOSite and LMSuccSite were developed using Python 3.9.7, TensorFlow 2.9.1, Keras 2.9.0, PyTorch 1.11.0, and Transformers 4.18.0. Consequently, to ensure seamless operation, your system should be equipped with corresponding compatible versions (*see* **Note 18**).

3.3 pLMSNOSite and LMSuccSite: Web Server

The pLMSNOSite and LMSuccSite web servers within the LMPTMSite platform were developed using the Python framework and served through the Apache web service. These tools operate entirely on the server side, with client queries sent via an Internet browser. The web user interfaces for pLMSNOSite and LMSuccSite are accessible at http://kcdukkalab.org/pLMSNOSite/ and http://kcdukkalab.org/LMSuccSite/, respectively. The screenshot of web server of pLMSNOSite is shown in Fig. 4.

Users have the option to upload a FASTA file containing protein sequences along with their respective accession IDs. An example FASTA file is readily available for download via a link on the

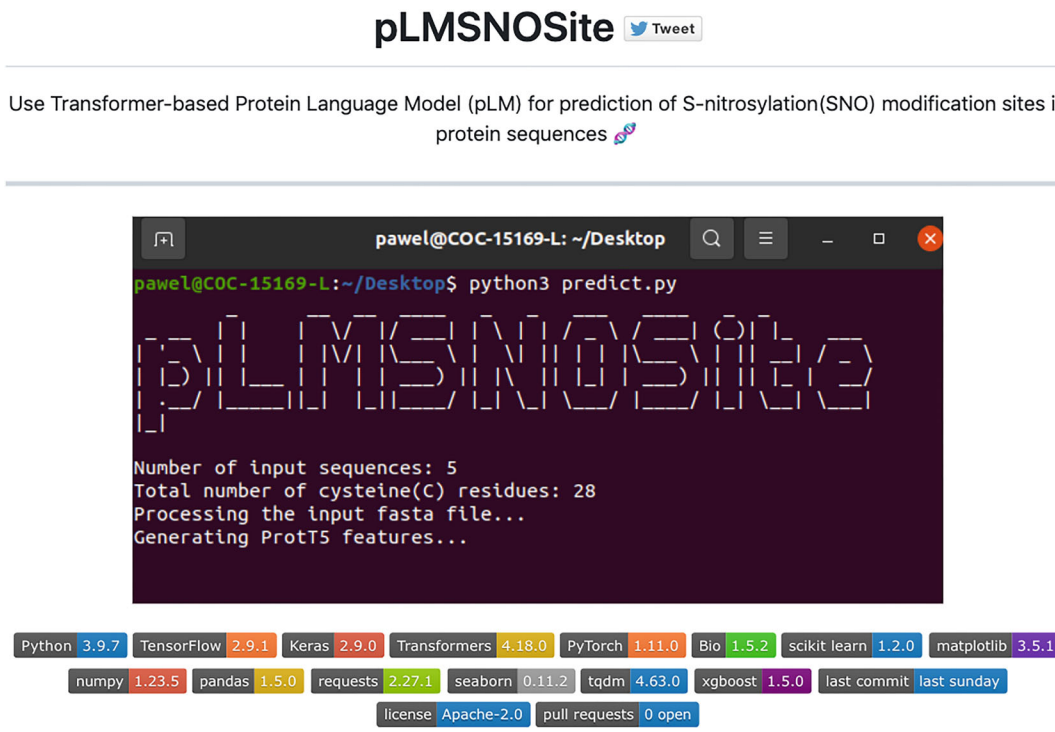

Fig. 3 Screenshot of standalone version of pLMSNOSite along with its CLI hosted at GitHub

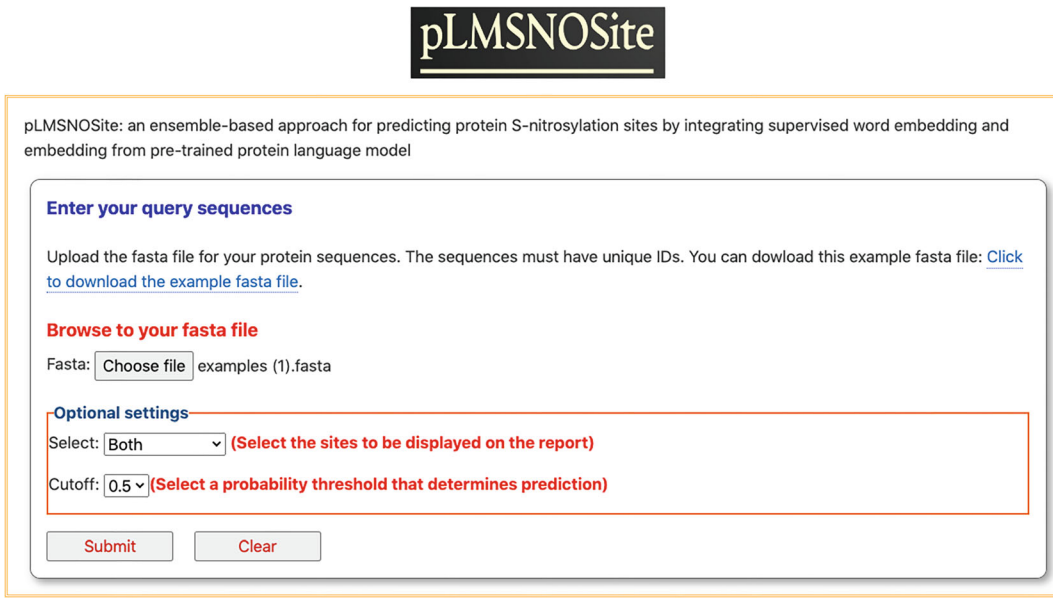

Fig. 4 Screenshot displaying the homepage of the pLMSNOSite web server

webpage (*Click to download the example fasta file*). Additionally, using optional settings, users can further customize their output by selecting whether they prefer to display the results of only positive sites or only negative sites or both. Along with it, users can set their desired decision threshold cut-off, ranging from 0.1 to 0.9, to convert raw probability scores into crisp labels (positive or negative). Please note that the default cut-off for the tools is set at 0.5, as this was the value utilized during the analytical phase of these tools.

Input Once a valid FASTA file is uploaded and the *Submit* button is pressed, the user's request will be sent to the server using an encrypted POST method. The processing time required to display the output will depend on the sequence length, the size of the FASTA file, and the current server load. The uploaded FASTA file will undergo a validity check; if the file is valid, the sequences will be pre-processed and fed into the model for feature extraction and site prediction, as detailed in the Materials and Methods section. An example output from the pLMSNOSite web server is shown in Fig. 5.

The selected probability cutoff: **0.5**
Total number of sites: **5**
The number of positive sites: **3**
The number of negative sites: **2**
The display setting: **Both positive and negative**

Output file download

Use the following links to download your input file and prediction results. This link will be valid for a week. The prediction is based on the probability cutoff chosen by the user upon query submission.

Prediction file	(download)	Your input file	(download)

Prediction report

Index	Protein description	Position	Site residue	Probability	Prediction	Label
1	sp|P0A901	19	C	0.6608795	1	Positive
2	sp|P0A901	131	C	0.34284467	0	Negative
3	sp|P09030	177	C	0.44460586	0	Negative
4	sp|P09030	240	C	0.69272876	1	Positive
5	sp|P09030	241	C	0.65235084	1	Positive

Fig. 5 Screenshot displaying an example output page of the pLMSNOSite web server

Output Upon successful processing of the input FASTA file, users will be directed to the results page, which displays various details such as the selected cut-off threshold, the total count of identified potential sites, and the chosen display setting (whether it is positive only, negative only, or both). The "Download" section provides links to export the results as a *csv* (comma-separated values) file. In addition to this, a "Prediction Report" section displays the prediction results, which include the probability scores and the predicted labels for each site. The results will be retained on the server for 1 week before deletion.

3.4 Runtime and Memory Usage Analysis

Additionally, for this book chapter, we conducted an analysis of CPU runtime and RAM memory usage by computing the time (in seconds) and memory (in megabytes) required for the pLMSNOSite tool to generate predictions across the input sequence (*see* **Note 19**). This process was examined as the length of the sequence was progressively scaled up, ranging from 100 to 4000. To account for variability, these experiments were repeated for ten runs for each length scale, and the results were averaged out. Furthermore, we separately analyzed the average time and average memory usage for the following components: *Environment setup* (which involves importing transformer and other libraries, loading models, defining devices (CPU or GPU), etc.), *ProtT5 encoding (Embedding Generation)*, *Prediction*, and *Encoding+Prediction*.

As the length of the sequence was scaled up (see Fig. 6a), we observed a quadratic increase in the average runtime of Encoding +Prediction (blue curve), which is mainly due to the quadratic nature of generating embeddings from the ProtT5 encoder (green curve). In terms of memory usage (see Fig. 6b), a loosely linear trend was observed for Encoding+Prediction (blue curve). Interestingly, ProtT5 encoding (green curve) did not contribute significantly to this trend—it was primarily the Prediction (red curve) process that demanded the majority of the memory.

It is important to highlight that LMSuccSite is expected to show similar trends regarding time and memory usage, considering it is based on a similar approach to pLMSNOSite. Both these tools rely on extracting feature vectors from protein sequences using ProtT5 and the same underlying architecture for predictions; hence their computational resource consumption would follow a similar pattern. This investigation provides valuable insights into the computational efficiency of our tools and highlights the balance between predictive performance and computational resources.

3.5 Investigating TCA Pathway Proteins: A Case Study

We also performed a case study to demonstrate the utility of our tools pLMSNOSite [20] and LMSuccSite [21]. In this case study, we selected five functional protein sequences originating from the *Homo sapiens*, all significantly involved in the citric acid cycle (also

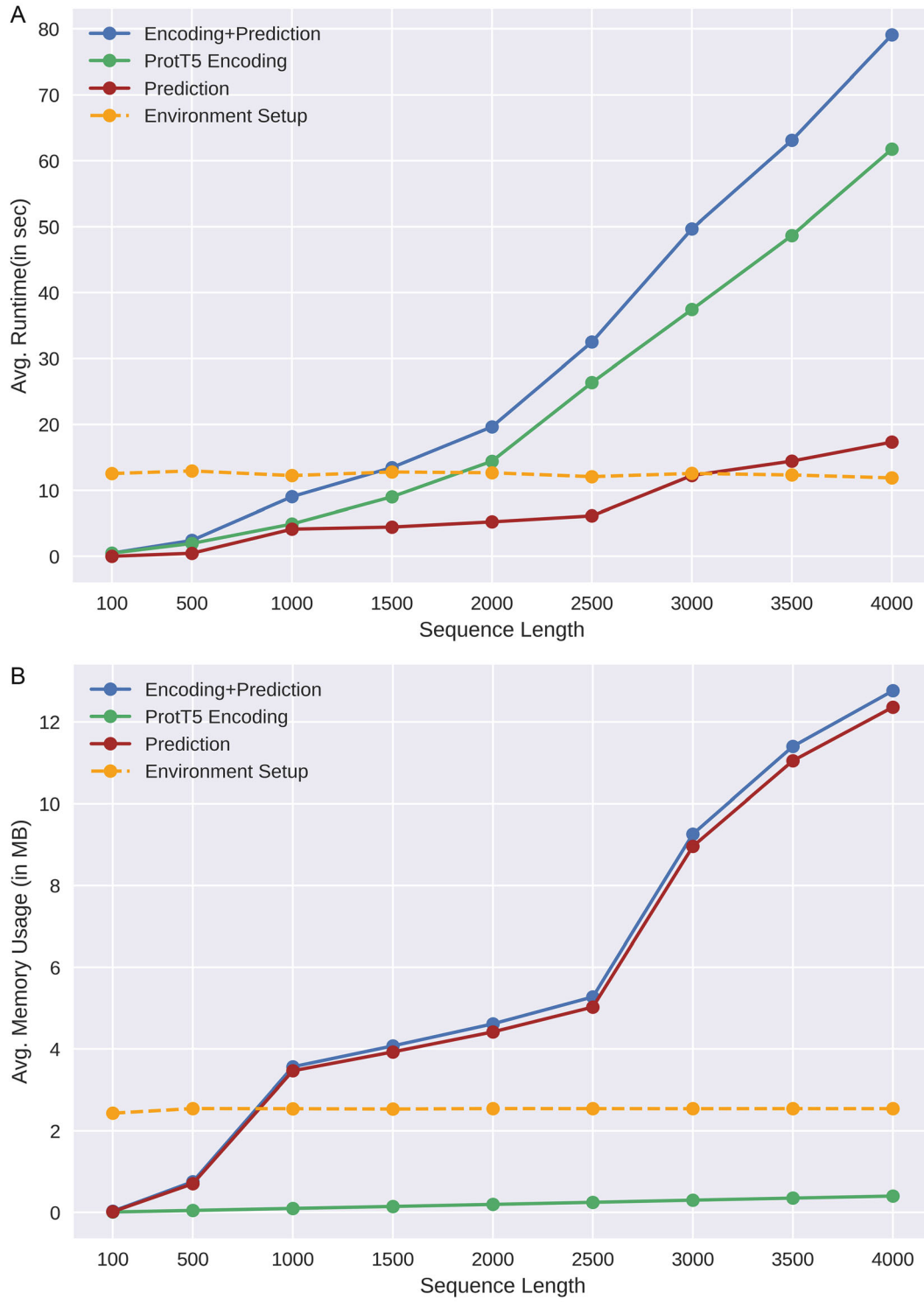

Fig. 6 (**a**) Relationship between average runtime (in seconds/sec) and sequence length and (**b**) average memory usage (in megabytes/MB) and sequence length for pLMSNOSite

known as the tricarboxylic acid cycle or TCA cycle). The TCA cycle is a sequence of chemical reactions that take place in both prokaryotic and eukaryotic cells, generating energy via the oxidation of acetyl-CoA, derived from carbohydrates, fats, and proteins. It is a central metabolic hub and plays a fundamental role in both the generation of energy and the provision of building blocks for the biosynthesis of complex molecules. The regulation of the TCA cycle is driven by a variety of genes that facilitate the process of carbohydrate metabolism. It has been found that proteins produced by these TCA pathway genes are primarily affected by succinylation [36]. For the purposes of our case study, we downloaded the canonical FASTA sequences of the proteins generated by these TCA cycle genes from KEGG pathway database [37]. Detailed information about these sequences along with their location in the eukaryotic cells and their function is reported in Table 11.

Utilizing the web server (or the standalone version) of LMSuccSite [21], we successfully identified 18 out of the 25 known succinylated lysine sites (sourced from the "experimental and putative" site collections in the dbPTM database [5]). This resulted in a TPR (sensitivity) of 72% and a FNR (type II error) of 28%. The substantial proportion of TPR over FNR, as illustrated in Fig. 7, underscores the effectiveness of our tool in predicting sites in sequences that actively participate in vital biological pathways like the TCA cycle.

3.6 Limitation and Future Work

Despite the promising results demonstrated by LMSuccSite and pLMSNOSite tools in predicting PTM sites—achieved by merging the global contextual information from pLMs and the local contextual information using word embedding layer—there remain

Table 11
Protein sequences from the tricarboxylic acid (TCA) pathway with their associated information

Accession ID (protein name)	Gene name	Location	Function
O75390 (CISY_HUMAN)	CS	Mitochondrial	Catalyzes the reaction of acetyl-CoA and oxaloacetate to form citrate and CoA in the citric acid cycle
P08559 (ODPA_HUMAN)	PDHA1	Mitochondrial	Part of the pyruvate dehydrogenase complex that catalyzes the overall conversion of pyruvate to acetyl-CoA and CO_2
P40925 (MDHC_HUMAN)	MDH1	Cytoplasmic	Reversibly converts malate into oxaloacetate using NAD+/NADH in the citric acid cycle
Q02218 (ODO1_HUMAN)	OGDH	Mitochondrial	Part of the 2-oxoglutarate dehydrogenase complex which converts 2-oxoglutarate to succinyl-CoA and CO_2 in the TCA cycle
Q99798 (ACON_HUMAN)	ACO2	Mitochondrial	Catalyzes the isomerization of citrate to isocitrate via cis-aconitate in the TCA cycle

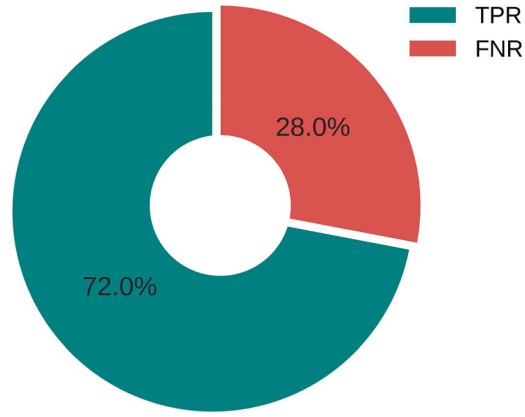

Fig. 7 Donut chart illustrating the distribution of true positive rate (TPR) and false positive rate (FPR) for the prediction of known succinylated sites within proteins involved in the tricarboxylic acid (TCA) cycle pathways, using LMSuccSite

certain limitations and opportunities for enhancement. One limitation is the lack of utilization of structural information. Incorporating structural information can potentially enhance the accuracy and robustness of the predictions, as it provides valuable insights into the spatial arrangement and interactions of amino acids. Furthermore, future work could explore the new pLMs, such as Ankh [38] or ESM-2 [39], which may have improved performance and capture additional contextual information. Additionally, there are alternative methods for obtaining features from pLMs that were not explored in our current approaches. Techniques like FSWE, WSPE, WSWE, and MWVL could be investigated to capture different aspects of protein sequences and potentially improve the predictive performance. Another aspect for future work is the consideration of fine-tuning. While we performed soft fine-tuning (feature-based approach), where we added fully connected layers on top of the pre-trained transformer encoder, hard fine-tuning, which involves jointly training the pLMs and the prediction models, was not explored. Incorporating hard fine-tuning may lead to further improvements in the model's performance. Moreover, an important area of future work includes integrating prediction tools for other types of PTM sites into the LMPTMSite platform.

4 Notes

1. Most existing predictors for S-nitrosylation (SNO) and Succinylation rely on manual or handcrafted feature extraction. Additionally, these predictors operate solely on window sequences, which restricts the computational model to capturing only the local contextual information of the site of interest. In contrast, our approach aims to eliminate these drawbacks while also obtaining the most robust results.

2. It is crucial that positive sites in the dataset are curated from experimentally verified sites to ensure their validity. Redundant sites, which can add noise and cause overfitting, should be eliminated using homology removal algorithms to maintain the uniqueness of the data. Duplicate sites (positive or negative) sharing same protein accession ID must be removed from the training set to avoid potential overfitting and to ensure diverse training set.
3. The dataset should be thoroughly examined for duplicate or contradicting sites. Such inconsistencies can occur when a site is labeled as both positive and negative simultaneously in the training or testing set or if a site's label differs between the training and testing sets. These duplicate or contradictory sites should be identified and removed to maintain the integrity of the dataset.
4. There should be no overlapping sequences between the training and testing sets, i.e., the split between these sets should be based on protein accession IDs. Alternatively, overlapping sequences can be removed from the training set. This is crucial because, especially when using protein language models for feature extraction, similar sequences in the training and testing sets could lead to feature information leakage, thereby inflating performance metrics.
5. Often, the negative set significantly outnumbers the positive set. This imbalance arises because all the sites within the same protein sequences that have not been experimentally verified as positive sites are consequently treated as negative sites. In such cases, data balancing methods should be applied to prevent the model from being biased toward the majority class. However, while addressing imbalance issues, care should be taken to ensure the distributions in the training and testing sets remain consistent and are not distorted, as such changes could inadvertently introduce biases. Furthermore, some sites labeled as negative may be false negatives, meaning they are yet to be experimentally identified as positive sites. This misclassification can result in inaccurate predictions, particularly in situations involving imbalanced learning, such as cost-sensitive learning.
6. In the context of learning from imbalanced datasets, we employ a cost-sensitive (CS) learning approach called class weighting. The strategy counteracts the bias toward the majority class by assigning different weights to each class. These weights serve as a "penalty factor" and aim to influence the learner's decisions. Misclassification of samples of a class with a higher weight will incur a greater penalty, compelling the model to pay more attention to such instances during the learning process. Weight of class i, W_i, is calculated according to the following formula:

$$W_i = \frac{n_s}{n_c \times n_{sc}}$$

where:

n_s refers to the total number of samples

n_c is the total number of unique classes (=2 in binary classification)

n_{sc} denotes the number of samples that belong to a specific class

Please note that the above formula is a common practice and there are variations to it that can also be employed based on the specific requirements of the problem.

7. In the context of post-translational modifications (PTMs), we designate "negative sites" as all potential sites for a given PTM which have not been experimentally annotated as positive. This may appear as a crude approximation, given that it might introduce a degree of false negatives—sites yet to be experimentally confirmed as positive. Although random undersampling could inadvertently eliminate some of these false negatives, it remains an area that requires careful consideration.
8. ESM-1 can accept input sequence length no longer than 1024. In contrast, ProtT5 and ProtBERT do not have this limitation and can handle protein sequences of arbitrary length. This flexibility allows for the analysis of longer protein sequences without the need for truncation or other preprocessing steps.
9. It is worth noting that the learned embeddings from Keras embedding layer are problem-specific. They capture the relationships between amino acids that are relevant to the specific task the model is trained on and may not necessarily be generalizable to other tasks or datasets.
10. We used only the encoder part of ProtT5 in our work, as our experiments showed that the embeddings from the encoder surpassed those from the decoder in performance. To expedite the generation of embeddings, we operated in half-precision mode. Notably, the developers observed no performance degradation in any of their experiments when running in this mode.
11. Within ML/deep learning architectures, three common fusion strategies are employed: early fusion, intermediate fusion, and late fusion. These strategies differ in how they combine input features (or modalities) before inputting them into the final inference model.
12. Early fusion, also known as feature-level fusion, merges input features (or features from the input layer of deep learning architectures) obtained from various representations before inputting them into the final inference model. This method

allows the model to learn joint representations at the input feature level, potentially capturing correlations between different modalities. However, this strategy might be less effective when input features from modalities have differing scales or difficult-to-align characteristics.

13. Intermediate fusion: This strategy involves input features from representations being initially learned by specific layers within their deep learning architectures independently. The resulting learned marginal representations at the intermediate level are combined through concatenation or addition. This method enables the model to learn individual representations for each input source or modality before merging them, which can be advantageous when the inputs possess distinct characteristics that necessitate separate processing.
14. Late fusion, also referred to as score-level fusion, integrates the outputs of models trained on their particular input representations. Each model generates its prediction, and these predictions are combined using methods such as weighted average, voting, or other aggregation techniques to produce the final inference. The limitation of this strategy is that it may not effectively capture correlations between different sources or modalities as well as early or intermediate fusion.
15. In the process of cross-validation, it is crucial to ensure that each fold is constructed in such a way that the training and validation sets do not contain common proteins. This precaution helps prevent any data leakage from the training set into the validation set, thus maintaining the integrity of the evaluation process.
16. In stacked generalization, or stacking, information leakage from base models to the meta-classifier can occur when the same dataset is used for training both. This can lead to overfitting, overly optimistic performance estimates, and potential biases in the final model prediction. To mitigate these issues, strategies like implementing Wolpert's stacking algorithm, using nested cross-validation, utilizing diverse base models, or separating training data for base models and meta-classifiers can be employed.
17. Both pLMSNOSite and LMSuccSite use a standard probability threshold cut-off of 0.5. The user can easily tailor the cut-off value according to their specific preference by following the steps outlined in the *Readme.md* file of the tool's respective repository.
18. The version of the dependencies includes Bio 1.5.2, Keras 2.9.0, Matplotlib 3.5.1, NumPy 1.23.5, Pandas 1.5.0, Requests 2.27.1, scikit-learn 1.2.0, Seaborn 0.11.2, TensorFlow 2.9.1, Torch 1.11.0, tqdm 4.63.0, transformers 4.18.0, and XGBoost 1.5.0.

19. We conducted the runtime and memory usage analysis on a system equipped with an Intel Core i9-10900X processor featuring ten cores, each running at 3.70 GHz (x86_64 architecture), and 128 GB of RAM. This system was operating on Ubuntu 22.04.2 LTS 64-bit. It is crucial to note that runtime and memory usage can significantly vary across different systems. Therefore, when interpreting our results, the specific hardware and software specifications we used should be taken into consideration.

Acknowledgments

This work was partly supported by the National Science Foundation grants number 2210356, 1901793 and MI-SAPPHIRE grant.

References

1. Khoury GA, Baliban RC, Floudas CA (2011) Proteome-wide post-translational modification statistics: frequency analysis and curation of the swiss-prot database. Sci Rep 1:90. https://doi.org/10.1038/srep00090
2. Boeckmann B, Bairoch A, Apweiler R et al (2003) The SWISS-PROT protein knowledgebase and its supplement TrEMBL in 2003. Nucleic Acids Res 31:365–370. https://doi.org/10.1093/nar/gkg095
3. Dinkel H, Chica C, Via A et al (2011) Phospho.ELM: a database of phosphorylation sites—update 2011. Nucleic Acids Res 39:D261–D267. https://doi.org/10.1093/nar/gkq1104
4. O-GLYCBASE version 4.0: a revised database of O-glycosylated proteins | Nucleic Acids Res | Oxford Academic. https://academic.oup.com/nar/article/27/1/370/1241788. Accessed 1 July 2023
5. dbPTM in 2022: an updated database for exploring regulatory networks and functional associations of protein post-translational modifications | Nucleic Acids Research | Oxford Academic. https://academic.oup.com/nar/article/50/D1/D471/6426061. Accessed 1 July 2023
6. Minguez P, Letunic I, Parca L, Bork P (2013) PTMcode: a database of known and predicted functional associations between post-translational modifications in proteins. Nucleic Acids Res 41:D306–D311. https://doi.org/10.1093/nar/gks1230
7. Hornbeck PV, Kornhauser JM, Tkachev S et al (2012) PhosphoSitePlus: a comprehensive resource for investigating the structure and function of experimentally determined post-translational modifications in man and mouse. Nucleic Acids Res 40:D261–D270. https://doi.org/10.1093/nar/gkr1122
8. Pakhrin SC, Pokharel S, Saigo H, KC DB (2022) Deep learning–based advances in protein posttranslational modification site and protein cleavage prediction. In: KC DB (ed) Computational methods for predicting post-translational modification sites. Springer US, New York, pp 285–322
9. Ismail HD, Jones A, Kim JH et al (2016) RF-Phos: a novel general phosphorylation site prediction tool based on random forest. BioMed Res Int 2016:3281590. https://doi.org/10.1155/2016/3281590
10. Larry RM, Jain LC (2001) Recurrent neural networks. Des Appl
11. Hochreiter S, Jürgen S (1997) Long short-term memory. Neural Comput 9:1735–1780
12. Yamashita R, Nishio M, Do RKG, Togashi K (2018) Convolutional neural networks: an overview and application in radiology. Insights Imaging 9:611–629. https://doi.org/10.1007/s13244-018-0639-9
13. Goldberg Y, Levy O (2014) word2vec explained: deriving Mikolov et al.'s negative-sampling word-embedding method
14. Pennington J, Socher R, Manning CD (2014) Glove: global vectors for word representation. Accessed 1 July 2023
15. Devlin J, Chang M-W, Lee K, Toutanova K (2019) BERT: pre-training of deep bidirectional transformers for language understanding

16. Pokharel S, Sidorov E, Caragea D, B Kc D (2022) NLP-based encoding techniques for prediction of post-translational modification sites and protein functions. In: Machine learning in bioinformatics of protein sequences. World Scientific, pp 81–127
17. Vaswani A, Shazeer N, Parmar N et al (2017) Attention is all you need. In: Advances in neural information processing systems. Curran Associates, Inc.
18. Radford A, Narasimhan K, Salimans T, Sutskever I (2018) Improving language understanding by generative pre-training
19. Exploring the limits of transfer learning with a unified text-to-text transformer | J Machine Learn Res. https://dl.acm.org/doi/abs/10.5555/3455716.3455856. Accessed 1 July 2023
20. Pratyush P, Pokharel S, Saigo H, Kc DB (2023) pLMSNOSite: an ensemble-based approach for predicting protein S-nitrosylation sites by integrating supervised word embedding and embedding from pre-trained protein language model. BMC Bioinformatics 24:41. https://doi.org/10.1186/s12859-023-05164-9
21. Pokharel S, Pratyush P, Heinzinger M et al (2022) Improving protein succinylation sites prediction using embeddings from protein language model. Sci Rep 12:16933. https://doi.org/10.1038/s41598-022-21366-2
22. Pakhrin SC, Pokharel S, Aoki-Kinoshita KF et al (2023) LMNglyPred: prediction of human N-linked glycosylation sites using embeddings from a pre-trained protein language model. Glycobiology 33:411–422. https://doi.org/10.1093/glycob/cwad033
23. Pakhrin S, Pokharel S, Pratyush P et al (2023) LMPhosSite: A deep learning-based approach for general protein phosphorylation site prediction using embeddings from local window sequence and pre-trained Protein Language Model. J Proteome Res
24. Stomberski CT, Hess DT, Stamler JS (2019) Protein S-nitrosylation: determinants of specificity and enzymatic regulation of S-nitrosothiol-based signaling. Antioxid Redox Signal 30:1331–1351. https://doi.org/10.1089/ars.2017.7403
25. Dai X, Zhou Y, Han F, Li J (2022) Succinylation and redox status in cancer cells. Front Oncol 12
26. Cd-hit: a fast program for clustering and comparing large sets of protein or nucleotide sequences | Bioinformatics | Oxford Academic. https://academic.oup.com/bioinformatics/article/22/13/1658/194225. Accessed 28 June 2023
27. Hasan MM, Manavalan B, Khatun MS, Kurata H (2019) Prediction of S-nitrosylation sites by integrating support vector machines and random forest. Mol Omics 15:451–458. https://doi.org/10.1039/C9MO00098D
28. DeepNitro: prediction of protein nitration and nitrosylation sites by deep learning – ScienceDirect. https://www.sciencedirect.com/science/article/pii/S1672022918303474. Accessed 28 June 2023
29. Hasan MM, Khatun MS, Mollah MNH et al (2017) A systematic identification of species-specific protein succinylation sites using joint element features information. Int J Nanomedicine 12:6303–6315. https://doi.org/10.2147/IJN.S140875
30. The UniProt Consortium (2015) UniProt: a hub for protein information. Nucleic Acids Res 43:D204–D212. https://doi.org/10.1093/nar/gku989
31. Elnaggar A, Heinzinger M, Dallago C et al (2021) ProtTrans: towards cracking the language of Lifes code through self-supervised deep learning and high performance computing. IEEE Trans Pattern Anal Mach Intell PP. https://doi.org/10.1109/TPAMI.2021.3095381
32. Rao R, Meier J, Sercu T, et al (2020) Transformer protein language models are unsupervised structure learners. 2020.12.15.422761
33. Villegas-Morcillo A, Makrodimitris S, van Ham RCHJ et al (2021) Unsupervised protein embeddings outperform hand-crafted sequence and structure features at predicting molecular function. Bioinformatics 37:162–170. https://doi.org/10.1093/bioinformatics/btaa701
34. Wolpert DH (1992) Stacked generalization. Neural Netw 5:241–259. https://doi.org/10.1016/S0893-6080(05)80023-1
35. Hasan MM, Kurata H (2018) GPSuc: global prediction of generic and species-specific succinylation sites by aggregating multiple sequence features. PLOS One 13:e0200283. https://doi.org/10.1371/journal.pone.0200283
36. Yang Y, Gibson GE (2019) Succinylation links metabolism to protein functions. Neurochem Res 44:2346–2359. https://doi.org/10.1007/s11064-019-02780-x
37. Kanehisa M (2002) The KEGG database. In: 'In Silico' simulation of biological processes. Wiley, pp 91–103
38. Elnaggar A, Essam H, Salah-Eldin W, et al (2023) Ankh : optimized protein language model unlocks general-purpose modelling. 2023.01.16.524265

39. Lin Z, Akin H, Rao R, et al (2022) Language models of protein sequences at the scale of evolution enable accurate structure prediction. 2022.07.20.500902
40. Xue Y, Liu Z, Gao X et al (2010) GPS-SNO: Computational Prediction of Protein S-Nitrosylation Sites with a Modified GPS Algorithm. PLOS One 5:e11290. https://doi.org/10.1371/journal.pone.0011290
41. Xu Y, Ding J, Wu L-Y, Chou K-C (2013) iSNO-PseAAC: predict cysteine S-nitrosylation sites in proteins by incorporating position specific amino acid propensity into pseudo amino acid composition. PloS One 8: e55844. https://doi.org/10.1371/journal.pone.0055844
42. SNOSite: exploiting maximal dependence decomposition to identify cysteine S-nitrosylation with substrate site specificity | PLOS One. https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0021849. Accessed 28 June 2023
43. Thapa N, Chaudhari M, McManus S et al (2020) DeepSuccinylSite: a deep learning based approach for protein succinylation site prediction. BMC Bioinformatics 21:63. https://doi.org/10.1186/s12859-020-3342-z
44. pSuc-FFSEA: predicting lysine succinylation sites in proteins based on feature fusion and stacking ensemble algorithm – PubMed. https://pubmed.ncbi.nlm.nih.gov/35686053/. Accessed 28 Jun 2023
45. Ning Q, Zhao X, Bao L et al (2018) Detecting succinylation sites from protein sequences using ensemble support vector machine. BMC Bioinformatics 19:237. https://doi.org/10.1186/s12859-018-2249-4