UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DE COMPUTAÇÃO

Elisson Lima Gomes da Silva

Multi-Head Attention Classifier Trained on Protein-level for Detecting Viruses Infecting Cassava from RNA-seq Reads

Recife

2024

Elisson Lima Gomes da Silva

Multi-Head Attention Classifier Trained on Protein-level for Detecting Viruses Infecting Cassava from RNA-seq Reads

The dissertation was submitted to the Graduate Program in Computer Science at the Informatics Center of the Federal University of Pernambuco as a partial requirement for obtaining a Master's degree in Computer Science.

Area of Concentration : Bioinformatics

Advisor: Prof. Dr. Stefan Michael Blawid

Recife

2024

.Catalogação de Publicação na Fonte. UFPE - Biblioteca Central

Silva, Elisson Lima Gomes da.
Multi-head attention classifier trained on protein-level for detecting viruses infecting cassava from RNA-seq reads / Elisson Lima Gomes da Silva. - Recife, 2024.
116 f.: il.

Dissertação (Mestrado) - Universidade Federal de Pernambuco, Centro de Informática, Programa de Pós-Graduação em Ciência da Computação, 2024.
Orientação: Stefan Michael Blawid.
Inclui referências e apêndices.

1. Detecção de vírus; 2. Dados de RNA-seq; 3. Classificação de leituras de sequenciamento; 4. Aprendizado profundo; 5. Métodos livres de alinhamento. I. Blawid, Stefan Michael. II. Título.

UFPE-Biblioteca Central

Elisson Lima Gomes da Silva

“Multi-Head Attention Classifier Trained on Protein-level for Detecting Viruses Infecting Cassava from Short Sequencing Reads”

Dissertação de mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Doutor em Ciência da Computação. Área de Concentração: Inteligência Computacional

Aprovado em: 13/09/2024.

BANCA EXAMINADORA

_____

Prof. Dr. Cleber Zanchettin
Centro de Informática / UFPE

_____

Prof. Dr. Ana Maria Benko Iseppon
Departamento de Genética / UFPE

_____

Prof. Dr. Stefan Michael Blawid
Centro de Informática / UFPE
(**orientador**)

I dedicate this dissertation to my wife, Monalisa.

AGRADECIMENTOS

First of all, I would like to thank God for giving me opportunities, guidance, and bless- ings to overcome the challenges and to be successful in this part of my life's journey so far. I'm extremely grateful to my wife and family for their upbringing, tireless efforts, and support on every path I take to achieve my dreams and goals. I would like to express my deepest gratitude to my advisor, Prof. Stefan Blawid, for sharing his knowledge and guidance from the beginning until the completion of my final research thesis. I would like to extend my sincere thanks to the professors in the Phytopathology Department at UFRPE, especially Prof. Rosana Blawid, for their support and partnership in the research that was developed. I am also thankful for all the support and knowledge offered by the professors from the Center of Informatics at UFPE and for the opportunities this department offered to me.

RESUMO

Este estudo aplica redes neurais artificiais para classificar leituras de dados de sequenciamento de alto rendimento (HTS), com foco específico na detecção de vírus em plantas de mandioca (*Manihot esculenta*). Doenças virais representam ameaças significativas à saúde das culturas e à produção de alimentos, e a mandioca, uma cultura crucial para a segurança alimentar e aplicações industriais no Brasil e globalmente, não é exceção. As pipelines tradicionais de bioinformática para a descoberta de vírus baseiam-se principalmente em métodos de alinhamento, que se tornam cada vez mais caros em termos computacionais à medida que o volume de dados genômicos de referência cresce. Metodologias sem alinhamento (AF), especialmente aquelas baseadas na análise de k-mers, oferecem uma alternativa promissora, mas muitas vezes enfrentam desafios relacionados à interpretabilidade e à demanda por memória.

Para enfrentar esses desafios, propomos um modelo de classificador de atenção *multi-head* projetado para detectar infecções virais em dados de sequenciamento de RNA obtidos de amostras de plantas e traduzido para o nível proteico. Este modelo, treinado para uma planta hospedeira específica, aproveita o mecanismo de atenção para melhorar a extração de características das distribuições de k-mers. Essa abordagem permite uma codificação mais dependente do contexto das leituras de sequenciamento, melhorando a classificação das sequências genéticas curtas típicas dos dados de HTS. Além disso, implementamos uma pipeline fitossanitária de última geração na *cloud* da Amazon Web Services para avaliar o desempenho do modelo proposto.

O modelo alcançou 99% de precisão durante o treinamento, filtrando efetivamente milhões de leituras do hospedeiro e de outros organismos, retendo apenas leituras virais. Essa redução substancial na demanda computacional para a identificação de novos vírus destaca a eficiência da nossa abordagem. Nossos resultados demonstram que modelos de *deep learning*, particularmente aqueles que empregam o mecanismo de atenção, podem classificar eficientemente sequências virais em leituras curtas, reduzindo significativamente os custos computacionais associados aos métodos tradicionais de AF. Este trabalho avança na análise genética e na bioinformática, oferecendo um método mais preciso e eficiente para a classificação de leituras de HTS na descoberta de patógenos em plantas.

Palavras-chaves: Detecção de vírus, Dados de RNA-seq, Classificação de leituras de sequenciamento, Aprendizado profundo, Métodos livres de alinhamento.

ABSTRACT

This work applies artificial neural networks for classifying reads from high-throughput sequencing (HTS) data, with a particular focus on detecting plant viruses in cassava (*Manihot esculenta*). Viral diseases pose significant threats to crop health and food production, and cassava, a crucial crop for food security and industrial applications in Brazil and globally is no exception. Traditional bioinformatics pipelines for virus discovery primarily rely on alignment-based methods, which become increasingly computationally expensive as the volume of genomic reference data grows. Alignment-free (AF) methodologies, especially those based on k-mer analysis, offer a promising alternative but often face challenges related to interpretability and memory demands.

To address these challenges, we propose a multi-headed attention classifier model designed to detect viral presence in RNA sequencing data obtained from plant samples and translated to the protein level. This model, trained for a specific host plant, leverages the attention mechanism to enhance feature extraction from k-mer distributions. This approach enables a more context-dependent encoding of sequencing reads, thereby improving the classification of the short genetic sequences typical of HTS data. Additionally, we implemented a cutting-edge phytosanitary pipeline on the Amazon Web Services Cloud to evaluate the performance of our proposed model.

The model achieved 99% accuracy during training, effectively filtering out millions of reads from the host and other organisms, and retaining only viral reads. This substantial reduction in computational demand for identifying new viruses underscores the efficiency of our approach. Our findings demonstrate that deep learning models, particularly those employing the attention mechanism, can efficiently classify viral sequences in short reads, significantly lowering the computational costs associated with traditional AF methods. This work advances genetic analysis and bioinformatics, providing a more accurate and efficient method for classifying HTS reads in plant pathogen discovery.

Keywords: Virus detection, RNA-seq data, Sequencing reads classification, Deep learning, Alignment-free method.

LIST OF FIGURES

LIST OF FRAMES

LIST OF TABLES

LISTA DE ABREVIATURAS E SIGLAS

ASCII	American Standard Code for Information Interchange
AUROC	Area Under the Receiver Operating Characteristic Curve
AWS	Amazon Web Services
BiLSTM	Bi-directional Long Short-term Memory
CIn	Centro de Informática
CIiFiPe	Clínica Fitossanitária de Pernambuco
CNN	Convolution Neural Networks
CPU	Central Processing Unit
DL	Deep Learning
DNA	Deoxyribonucleic Acid
DSMZ	Deutsche Sammlung von Mikroorganismen und Zellkulturen
EBS	Elastic Block Storage
EC2	Elastic Compute Cloud
Embrapa	Empresa Brasileira de Pesquisa Agropecuária
FAO	Food and Agriculture Organization
HMM	Hidden Markov Models
HPC	High-Performance Computing
HPV	Human Papillomavirus
HTML	Hypertext Markup Language
HTS	High-throughput sequencing
IaaC	Infrastructure as a Code
IDF	Inverse Document Frequency
LCA	Lowest Common Ancestor
LSTM	Long Short-term Memory
MB	Megabytes

ML	Machine Learning
mRNA	Messsneger RNA
NCBI	National Center for Biotechnology Information
ncRNA	Non-coding RNA
NGS	Next-generation Sequencing
NLP	Natural Language Processing
nt	Nucleotide
ORFs	Open Reading Frames
PCR	Polymerase Chain Reaction
QUAST	Quality Assessment Tool for Genome Assemblies
RAM	Random Access Memory
RefSeq	Reference Sequence
ReLU	Rectified Linear Unit
RNA	Ribonucleic Acid
RNA-seq	RNA sequencing
RNN	Recurrent Neural Network
rRNA	Ribosomal RNA
RSync	Remote Synchronization
S3	Amazon Simple Storage Service
SSD	Solid State Drive
SSH	Secure Shell
SVM	Support Vector Machines
TF	Term Frequency
TF-IDF	Term Frequency-Inverse Document Frequency
UFPE	Universidade Federal de Pernambuco
UFRPE	Universidade Federal Rural de Pernambuco

CONTENTS

1 INTRODUCTION

1.1 MOTIVATION

According to the United Nations Food and Agriculture Organization (FAO), cassava ranks among the most significant crops globally, alongside rice, corn, and sweet potato (RYBICKI, 2015). This root vegetable is a crucial food source for millions of people, particularly in tropical regions. As of 2009, over 700 million people depended on cassava as a major part of their diet, with Africa accounting for half of this population (PATIL, 2009). This highlights cassava's critical role in ensuring food security and nutrition in many developing countries.

Brazil stands out as a major producer of cassava, holding the position of the fifth-largest producer worldwide. In 2022, Brazil produced more than 17 million tons of cassava, accounting for 13.46% of global production. This significant contribution underscores Brazil's pivotal role in the cassava supply chain and its importance to the country's agricultural economy. The extensive cassava cultivation in Brazil supports local consumption and contributes to various industrial applications (GOMES, 2024; IBGE, 2024).

The widespread cultivation and utilization of cassava underscore its global impor-

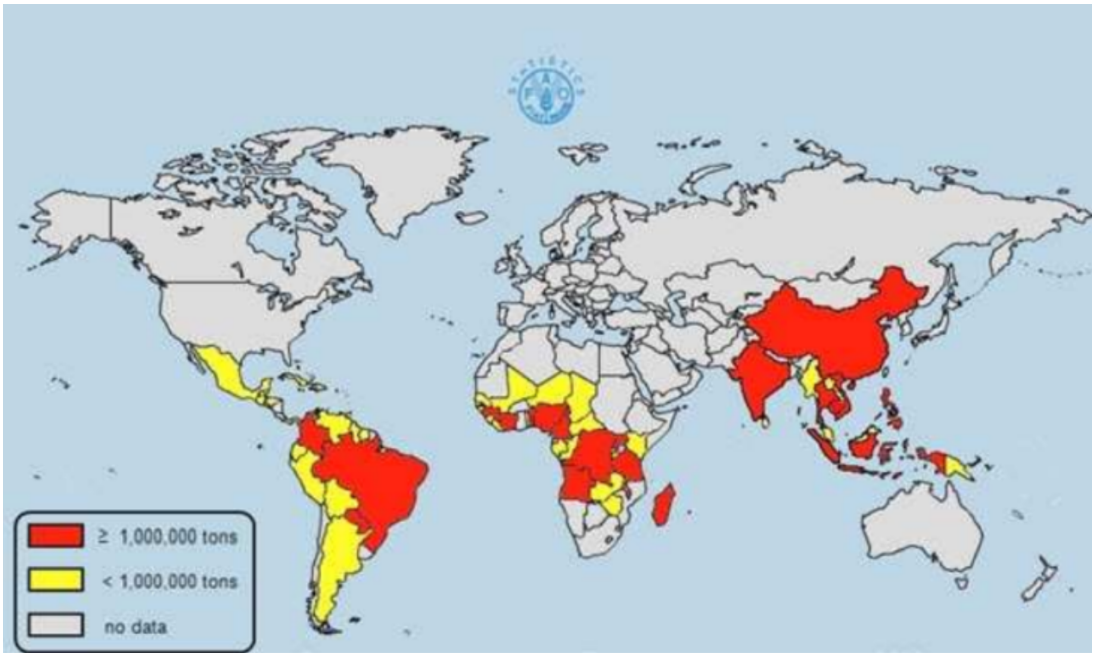


Figure 1 – Distribution of cassava production in the world. (KIM et al., 2017)

tance. Cassava is consumed in its natural form and processed into diverse products. These include biofuel, bioplastic, medicines, and other industrial products, showcasing the crop's versatility and economic value. The numbers and trends in cassava production and consumption illustrate its critical role in food security and industrial applications, making it a vital crop for millions of people worldwide (PATIL, 2009; GOMES, 2024). Figure 1 depicts the worldwide distribution of cassava production in the year 2017.

The extensive cultivation of cassava presents significant challenges, particularly concerning the spread of viral diseases. Plant viruses are a major concern for large-scale crops in developing countries (RYBICKI, 2015). One of the most critical issues affecting cassava production is cassava mosaic disease. This disease is prevalent in regions like the African continent and the Indian subcontinent. Cassava mosaic disease is caused by a family of viruses known as cassava mosaic geminiviruses, which belong to the *Begomovirus* genus. These viruses are highly detrimental to cassava plants, causing symptoms such as leaf distortion, chlorosis, and stunted growth, which can lead to significant yield losses (PATIL, 2009).

In Brazil, viral diseases pose a significant threat to cassava production, with cassava mosaic disease caused by the potexvirus cassava common mosaic virus leading to potential losses of up to 30% in the country's cassava output. Other notable viruses affecting cassava in Brazil include cassava american latent virus, cassava polero-like virus, cassava new alphaflexivirus, and cassava torrado-like virus. Understanding the dynamics of these viral communities is crucial for managing and mitigating their impact on cassava crops. RNA sequencing (RNA-seq) data analysis offers valuable insights into the temporal and spatial patterns of these viral populations, enabling researchers to characterize and study the viruses present in the ecosystem and assess their overall impact.

RNA-seq data in plant samples primarily represent the transcriptome, including mRNA, rRNA, ncRNA, and potentially RNA from associated organisms (e.g., viruses, bacteria, fungi) and organelles (mitochondria and chloroplasts). The specific content depends on the RNA extraction and library preparation protocols used. The libraries used in this study deplete the rRNA content.

Recent advancements in viral detection tools have significantly enhanced our ability to analyze viral sequences in metadata samples. The global pandemic caused by

SARS-CoV-2 has accelerated the development and improvement of these tools, leading to innovations such as the Serratus platform (<https://serratus.io/>). This platform, along with other practical and specific pipelines, has been instrumental in discovering new plant viruses and analyzing various sample types.High-throughput sequencing (HTS) combined with molecular detection techniques have become essential for uncovering viral diversity in plants and are particularly effective for detecting and monitoring viruses in propagative material.

The application of these advanced technologies allows for a more comprehensive understanding of the viral landscape affecting cassava crops. By employing HTS and molecular detection methods, researchers can identify and monitor the presence of multiple viruses, including those that may not have been previously detected. This detailed knowledge is crucial for developing effective strategies to combat viral diseases and minimize their impact on cassava production. It also facilitates the early detection of emerging viral threats, enabling prompt and targeted responses to protect the crops.

Overall, the integration of RNA-seq data analysis and advanced sequencing techniques represents a significant step forward in managing viral diseases in cassava. These approaches not only enhance our understanding of the viruses affecting cassava but also provide the necessary data to inform better agricultural practices and disease management strategies. By leveraging these technologies, Brazil can improve the resilience of its cassava production systems, ensuring the sustainability and productivity of this vital crop in the face of viral challenges.

1.2 PROBLEM

Accurate virus disease identification is crucial for implementing effective treatment strategies targeting the causative agents; new and old virus infection identification plays an important role. By diagnosing diseases correctly, farmers can take timely preharvest prevention and treatment measures, applying the appropriate phytosanitary products and dosages to minimize harm to crops, the environment, and human health. This precision in disease management enhances crop yields and sustainability and reduces the risk of resistance development among pests and pathogens, ultimately supporting a more resilient and productive agricultural system.

Traditionally, the identification of viral infections can be performed using various

tools and methods. One of the methods that has recently gained prominence is identification using RNA-seq data obtained through HTS. This activity aims to match a given sequence to a group of already known sequences that share similar characteristics (FABIJAńSKA; GRABOWSKI, 2019b), or to identify which group a particular sequence belongs to based on its specific content, using a model trained to perform this task (MATEOS et al., 2021). The comparison between two RNA sequences, base by base, is called sequence alignment. However, this technique has shown to be inefficient because genomes can diverge significantly, making this method unreliable or impossible (SIMS et al., 2009).

Historically, some tools have been developed to compare a RNA sequence with a database of known references. Kraken (WOOD; SALZBERG, 2014) and Kaiju (MENZEL; NG; KROGH, 2016) are examples of well-established tools used to identify viral infections by comparing sequences with a reference database. These tools perform this task deterministically, and the biggest challenge faced when attempting to use them is the high computational power required to run them.

As a result, some tools based on machine learning techniques have been proposed to address this problem of identifying viral infections more efficiently (FABIJAńSKA; GRABOWSKI, 2019b; MATEOS et al., 2021). However, it was observed that despite the excellent performance in executing the task, some of these tools lost accuracy when used on sequences about 100nt in length. Another limitation of these tools is the selection of samples used for training the models, which means the model needs to be retrained to address specific cases.

The present work proposes a deep-learning model to identify viral infection from RNA-seq data obtained through HTS where sequences has between 100nt and 300nt size. The model was developed in partnership with experts from the Clínica Fitossanitária de Pernambuco (CliFiPe), or Phytosanitary Clinic of Pernambuco, hosted by the Universidade Federal Rural de Pernambuco (UFRPE). The goal is to provide technical aid in plant disease diagnosis, prevention, and treatment more efficiently. Furthermore, the efficiency gain shall enable the experts from CliFiPe to faster detect and identify new viral infections from cassava RNA-seq data.

1.3 OBJECTIVES

The main objective of this project is to develop and implement a deep learning model capable of identifying the presence of viral particles infecting cassava from sampled RNA-seq data. To achieve this goal, the following specific objectives have been defined:

- Develop a method to generate artificial reads from viral, bacterial, and cassava genome sequences. This step is essential for creating datasets that will be used to train and validate the models.
- Configure and execute a phytosanitary pipeline to serve as a performance baseline for evaluating future models.
- Review and assess models proposed in related works to determine their applicability within the context of this project.
- Design and implement a deep learning model specifically tailored to detect viral infections from RNA-seq data.

1.4 CONTRIBUTIONS

This dissertation advances the detection of viral infections in cassava through the analysis of RNA-seq data obtained via high-throughput sequencing (HTS) and aids in the identification of novel viruses that infect cassava. The key contributions of this work are as follows:

- Developed an algorithm for collecting data from NCBI databases and constructing datasets for training deep learning models.
- Implemented a state-of-the-art phytosanitary pipeline as a cloud service to identify novel viruses.
- Constructed tailored reference databases in the cloud, enabling the execution of bioinformatics tools on resource-limited hardware.
- Provided proof of concept that machine learning algorithms can successfully extract features for virus detection from short sequencing reads at the protein level.

- Introduced the attention mechanism as a viable tool for bioinformatics applications, demonstrating its potential for enhancing viral detection.
- Analyzed RNA-seq data sampled from cassava plants showing disease symptoms.

2 THEORETICAL BACKGROUND

This chapter will explore various concepts to establish the theoretical basis for the research conducted. First, Section 2.1 introduces the use of machine learning to solve Natural Language Processing (NLP) problems. Section 2.2 presents recurrent models. Section 2.3 presents the attention mechanism. Finally, Section 2.4 explains transformer models, the state-of-the-art model for NLP problems.

2.1 MACHINE LEARNING AND NATURAL LANGUAGE PROCESSING

Machine Learning (ML) is a field of study that employs algorithms that allow computers to learn without being explicitly programmed (MAHESH, 2019). There's a variety of ML algorithm types that are applied to different problem categories (MAHESH, 2019), and to solve these problems, all those algorithms execute a step that is called *learning* (SZE et al., 2017). This learning step relies on the data available to execute the learning process, and the choice of the proper ML algorithm depends on the volume, characteristics, and structure of the data (MAHESH, 2019).

Nowadays, Machine Learning is widely used to solve several problems in society. ML algorithms have several applications: image recognition, behavior analytics, predictive analytics, and natural language processing. Figure 2 shows an example of an application of NLP called sentiment analysis. This NLP sub-field identifies and extracts mood within a given text (SARKER, 2021).

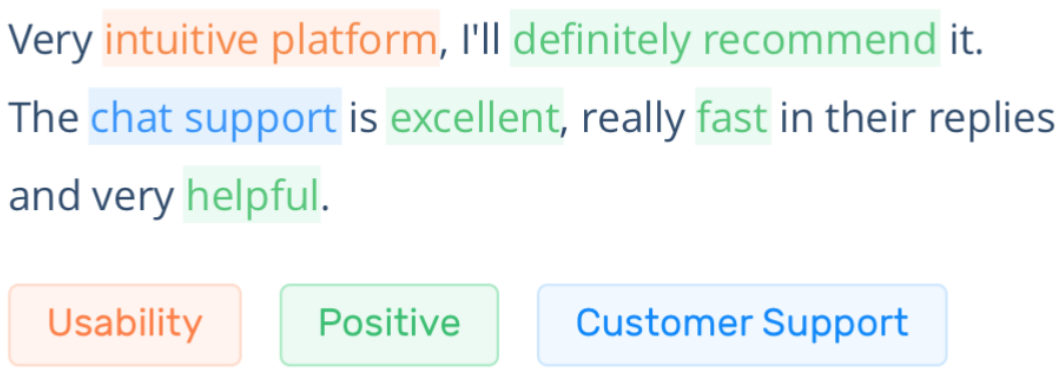


Figure 2 – An application of Natural Language Processing: Sentiment analysis (NATURAL..., 2024).

Natural language processing, or computational linguistics, is the study area focused

on creating computer systems and methods to address real-world challenges in understanding human languages (OTTER; MEDINA; KALITA, 2020). Therefore, NLP enables computers to perform tasks such as reading text, listening to speech, understanding its meaning, analyzing emotions, and determining important elements. Machine learning techniques can be applied in this process. Examples of NLP applications include virtual personal assistants, chatbots, speech recognition, document summarization, and language translation (SARKER, 2021).

Another important application of NLP models is to solve text classification problems. This traditional problem focuses on assigning labels to text segments like sentences, queries, paragraphs, and documents. It has many applications, including question answering, spam detection, sentiment analysis, news categorization, user intent classification, and content moderation (MINAEE et al., 2021). To address this problem, many machine-learning models have been applied through the decades; popular choices include Naïve Bayes, Support Vector Machines (SVM), Hidden Markov Models (HMM), tree-based models, and Neural-based models (MINAEE et al., 2021).

Popular neural-based approaches apply deep learning models to their architectures. Machine-learned embedding, convolutional neural networks, recurrent models, and attention-based models are very popular choices, being the last two of them the state-of-the-art nowadays. All these deep learning concepts and models will be described in more detail in the next sections.

2.2 EMBEDDINGS

When developing machine learning models for NLP problem domains, the first challenge is efficiently representing human language so the model can extract information from text and thus learn from input data. This challenge is formally known as feature (or information) representation, and it has received considerable attention in recent decades. It is relevant to highlight this task as one of the most important steps in NLP models because this enables the model to execute useful operations on texts (e.g., addition, distance measures, etc.) (ALMEIDA; XEXéO, 2023).

Outlining the technique in a few words: Embedding maps words from a vocabulary to a numerical representation or vector. The simplest approaches to achieve this use statistical functions called frequency-based embeddings; others are called prediction-

based embeddings and leverage neural networks to create the numerical representations.

Frequency-based embedding is a technique that creates vector representations of words based on how frequently they appear in a text corpus. These methods use statistical measures of word occurrence to capture and encode the meanings of words (5. . . , 2024). The most classical method of this type of embedding is the Count Vectorizer, which creates a sparse matrix containing a simple count of the number of occurrences of a feature in the data set (TRIPATHY; AGRAWAL; RATH, 2015). However, this technique reaches its limit when dealing with more complex scenarios that require large datasets, often containing millions or billions of words (MIKOLOV et al., 2013b).

One popular frequency-based embedding technique is Term Frequency-Inverse Document Frequency (TF-IDF). It compares how often a word appears in a specific document to how rare it is across a collection of documents. This calculation helps determine how important a word is to a particular document. Words frequently appearing in just one or a few documents get higher TF-IDF scores, indicating their relevance. In contrast, common words like "the" or "and" have lower TF-IDF scores because they appear in many documents and are less informative (RAMOS, 2003). Figure 3 shows the score calculation for Term Frequency (TF) in blue - this is the frequency of words - and the term frequency multiplied by the Inverse Document Frequency (IDF) in pink. A very interesting application of TF-IDF is to balance the weight between the most frequent or general words and the less commonly used words in a text classification problem (TRIPATHY; AGRAWAL; RATH, 2015).

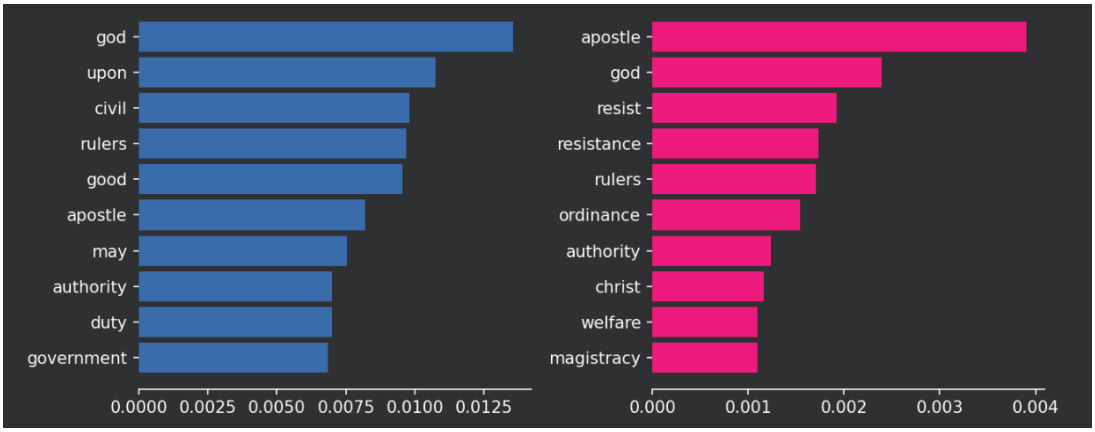


Figure 3 – Visualization of TF-IDF scores showing that the frequency of words and frequency of words across documents can be different (PONNE, 2023).

Prediction-based embeddings have grown in popularity in recent years. They are

created by models that learn to predict a word based on the words around it in sentences. These methods aim to place words that appear in similar contexts near each other in the embedding space. This often produces more detailed word vectors that capture various linguistic relationships (5..., 2024). As there are several ways of designing these types of embedding models, it's possible to split prediction-based embeddings into static and contextualized categories. Static word embedding assigns probabilities to words and represents each word as a vector. These embeddings are learned by training lookup tables that convert words into dense vectors. They are called "static" because, once trained, the vectors do not change based on the context of different sentences, and the embedding tables remain the same across all sentences (BIRUNDA; DEVI, 2021). On the other hand, contextualized embeddings generate a representation of each word based on its context in the sentence. For example, the contextualized representation of the word “bank” would be different in bank deposits and riverbanks (MACAVANEY et al., 2019).

A static prediction-based embedding called Word2Vec initially proposed by Mikolov et al. (MIKOLOV et al., 2013a) was improved by Ge et al. (GE; MOH, 2017). This technique uses a neural network that extracts an optimal word representation learned from a large dataset (5..., 2024). The model is trained to predict the surrounding context words of a given word, and with that, the model can learn the relationship between the words. Figure 4 shows the relationship learned from data and illustrates how the model can organize concepts (MIKOLOV et al., 2013a). Another important aspect of this model is feature reduction. When training an embedding model, one can find that the relationship is stored in the model weights. Thus, the relationship of millions of words is stored compactly, making it easy to embed this knowledge in other models (GE; MOH, 2017).

Matthew et al. (PETERS et al., 2018) propose an embedding model called ELMo that addresses two main challenges: To learn the complex features of the use of words like syntax and semantics and how the use of these words varies across linguistic context (PETERS et al., 2018). This model uses neural networks but applies deep learning to build its deep contextualized word representation. When used with neural networks, ELMo achieved 97% accuracy when classifying texts (BIRUNDA; DEVI, 2021).

Another example of embedding is proposed by Dai et al. (DAI et al., 2017) called Sequence2Vec. The main difference of this model is that it is trained to represent DNA

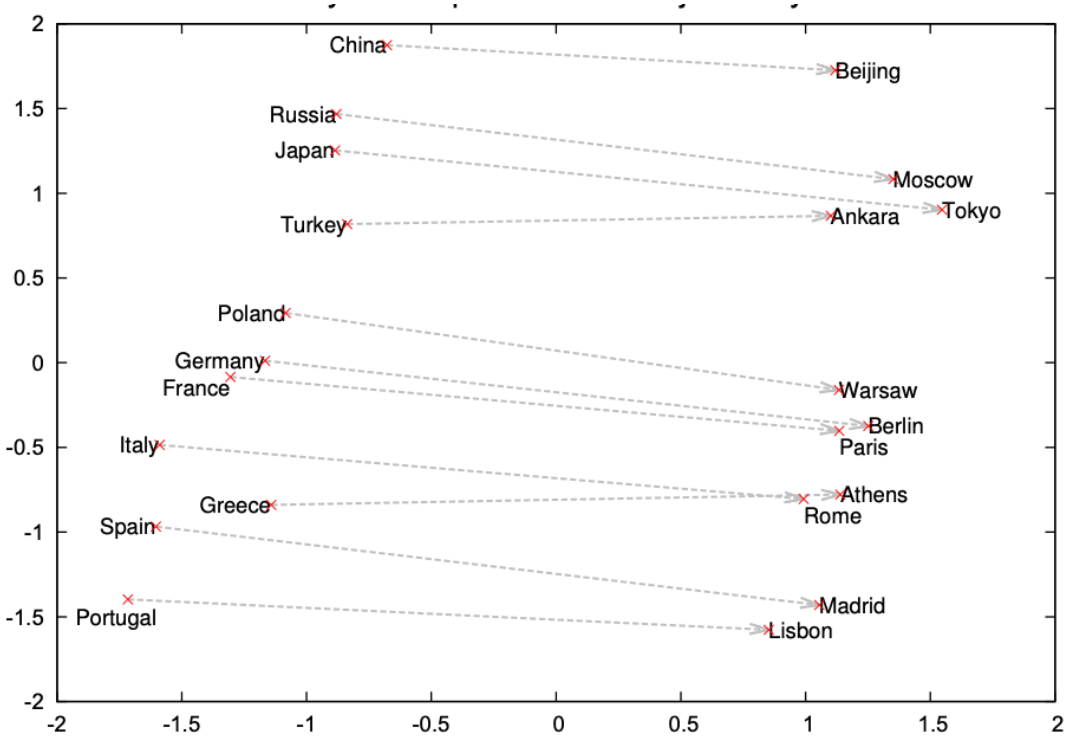


Figure 4 – Relationship between city names and countries learned by a Word2Vec model from context (MIKOLOV et al., 2013a).

sequences instead of human language. The method starts by representing DNA binding sequences as a hidden Markov model, which captures the sequence's position-specific information and long-range dependency. Finally, the model transforms these hidden Markov models into a shared nonlinear feature space and then uses these embedded features to construct a predictive model. Our approach uniquely combines the advantages of probabilistic graphical models, feature space embedding, and deep learning.

2.3 CONVOLUTIONAL NEURAL NETWORKS

Brain-inspired models have driven innovation in machine learning and deep learning fields, and the Convolution Neural Networks (CNN) model is a classic example. This model was initially proposed by LeCun et al. (LECUN et al., 1998) to identify handwritten digits, outperforming several other machine learning algorithms on this task. The inspiration for this type of network is the hierarchical model of the visual nervous system (FUKUSHIMA, 1980). CNNs are usually the most popular choice for image and video-related tasks, like image classification, object recognition, object detection, etc.,

but they also have applications in other tasks like NLP.

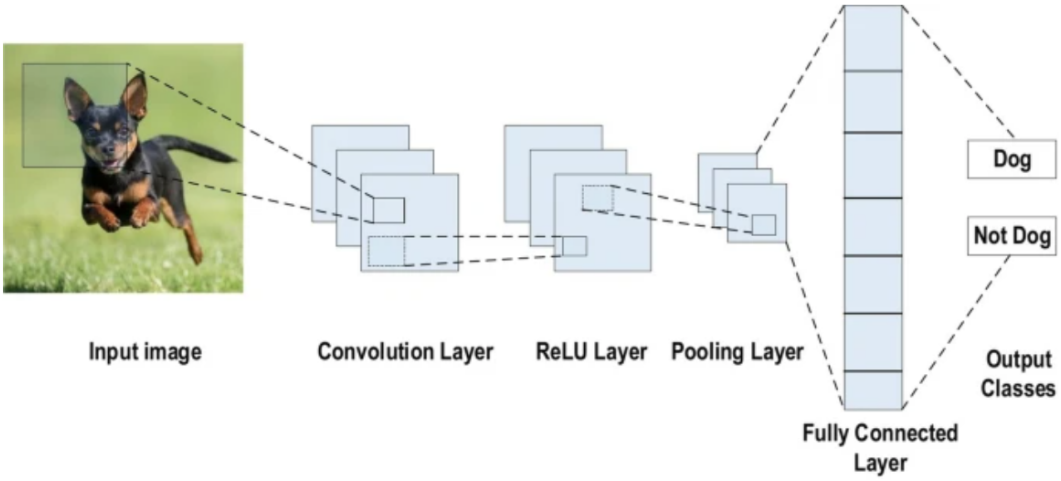


Figure 5 – An example of a simple Convolution Neural Networks (CNN) architecture for image classification (ALZUBAIDI et al., 2021).

Figure 5 shows a simple CNN model to classify whether an image depicts a dog. It is possible to notice that the model has several layers, initially proposed by LeCun et al. (LECUN et al., 1998) for its first CNN network called LeNet. The idea is that each layer is responsible for executing a step on the model task, from feature extraction to classification. Every CNN may be significantly different from others, but a typical model will have the following layers:

- **Convolution**: It is the feature extraction stage of the model. It is responsible for creating a feature map containing information about the input. For image classification example, it will contain borders, contours, etc. (YAMASHITA et al., 2018).
- **Activation Function**: It introduces non-linearity into the model. It enables the model to learn and perform very complex tasks (CONVOLUTIONAL..., 2024). The most common activation functions used are Rectified Linear Unit (ReLU), Sigmoid, and hyperbolic tangent (Tanh) (YAMASHITA et al., 2018).
- **Pooling**: The step executes the sub-sampling of the feature maps. This reduces the size of the feature map, creating smaller maps (ALZUBAIDI et al., 2021).
- **Fully-connected**: This type of layer exists at the end of every CNN model. This step is responsible for flattening the feature maps as a vector. Based on the value of the last vector outputted, it is possible to get the classification result of the model (ALZUBAIDI et al., 2021).

It is important to mention that, as deep learning evolved over the years, CNN architecture became increasingly complex. A typical CNN model has several convolution, activation, and pooling layers blocks. Each block is responsible for increasing the abstraction level of the feature map. For example, the first block will be responsible for extracting borders, the second block will be responsible for some basic shapes, and finally, the third will be responsible for some objects.

Nowadays, CNN is not only used to perform image classification, object detection, and object recognition but also on NLP tasks (YIN et al., 2017). Min et al. (MIN; LEE; YOON, 2016) shows that CNN also has application in many bioinformatics tasks, for example, DNA and RNA classification and function prediction.

2.4 RECURRENT NEURAL NETWORKS

Recurrent Neural Network (RNN) are a type of model that is very popular for addressing tasks that require processing sequential data. This data sequence can include documents, social media posts, genomic data, numerals, etc. It's possible to create models for various problems like language modeling, image or video captioning, speech recognition, text generation, etc. The main difference between recurrent and non-recurrent networks is the way information is propagated through the network; a recurrent model processes data in cycles (SCHMIDT, 2019).

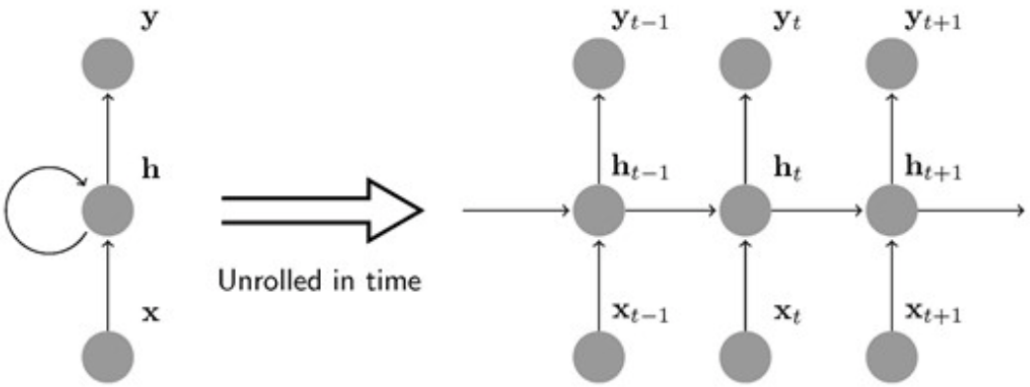


Figure 6 – Basic structure of a Recurrent Neural Network (RNN) (MIN; LEE; YOON, 2016).

Figure 6 shows the basic structure of recurrent neural networks. This diagram also shows how the RNN processes sequential data: A cyclic connection is present, meaning the hidden unit gets inputs from both the hidden unit's state at the previous time

step and the input unit at the current time step (MIN; LEE; YOON, 2016). RNN models are great for capturing dependencies between elements of sequential input. These dependencies can be easily interpreted in NLP problems. Figure 7 indicates relationships between words in a simple sentence. RNNs models are very good in recognizing the relationships between words in close vicinity, e.g., in capturing the year Gwathmey was born. However, as the model tries to capture longer dependencies, e.g., Rosalie's son's name, the model will begin to suffer from "vanishing gradients" and "exploding gradients" issues (SHERSTINSKY, 2020).

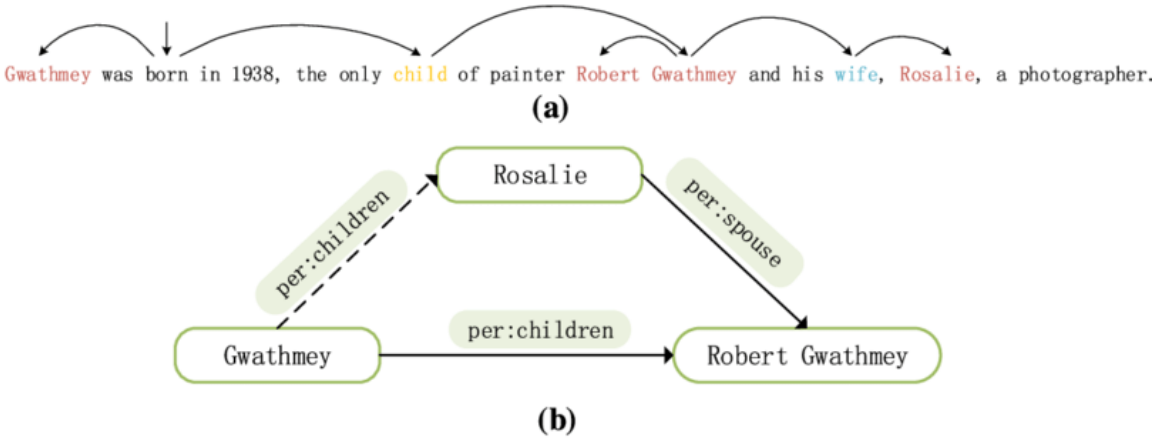


Figure 7 – (a) A simple sentence describing the relationship between three people; (b) The dependencies between terms (the subjects) need to be extracted, a prevalent problem in NLP tasks (LI et al., 2021).

Schmidt (SCHMIDT, 2019) defines the problem of vanishing and exploding gradients. The issue will emerge for too long (exploding gradients) or too short (vanishing gradients) input sentences. In the case of a vanishing gradient, the model does not give enough attention to critical elements for comprehension of the input sequence. For exploding gradients, the model assigns an exaggeratedly high weight value to some elements. Those issues motivated the introduction of a novel model type addressing these challenges (SHERSTINSKY, 2020) and outperforming traditional RNNs on a variety of tasks (SCHMIDT, 2019).

Long Short-term Memory (LSTM) is a recurrent neural network designed to address the vanishing and exploding gradients problem. While traditional RNNs did not scale to long-time dependencies, the LSTM models performs well on capturing long-term temporal dependencies (GREFF et al., 2017). The main difference between conventional RNN and LSTM models is the idea of memory cells. LSTM cells have three gates—input, forget, and output gates—that modify a cell state vector, which is repeat-

edly updated to capture long-term dependencies. This controlled flow of information within the cell allows the network to remember various time dependencies with different properties (LINDEMANN et al., 2021).

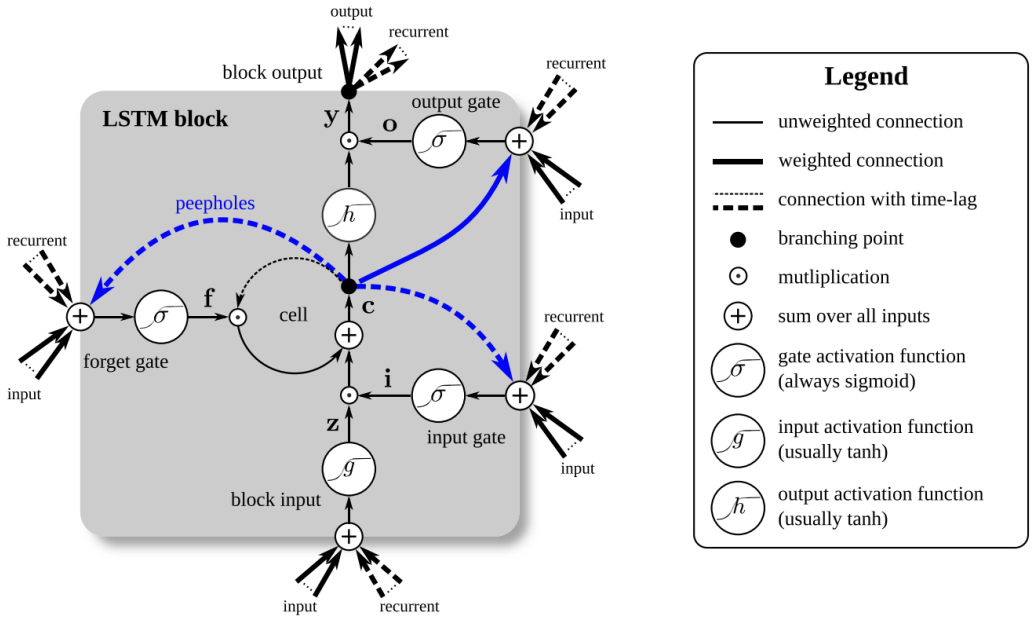


Figure 8 – Detailed schema of an LSTM block (GREFF et al., 2017).

Figure 8 shows the gates, activation function, and information flow of an LSTM block. The output gate displays the entire cell, the input gate reads data into the cell, and the forget gate resets the cell's content. Basic LSTM setups have been used to address a variety of tasks that require sequential data processing. However, Greff et al. (GREFF et al., 2017), and Lindemann et al. (LINDEMANN et al., 2021) show that several variations of this model were proposed to address specific tasks, e.g., bidirectional LSTM, LSTM autoencoder, etc.

2.5 ATTENTION MECHANISM

Attention is a crucial cognitive function that allows humans to selectively focus on specific parts of information when needed, rather than processing everything at once. This selective focus improves the efficiency and accuracy of perceptual information processing. Human attention mechanisms are divided into two types: bottom-up unconscious attention (saliency-based), driven by external stimuli, and top-down conscious attention (focused), driven by specific tasks and goals. In deep learning, most

attention mechanisms are designed as focused attention to enhancing task-specific performance (NIU; ZHONG; YU, 2021).

Following this inspiration, Bahdanau et al. (BAHDANAU; CHO; BENGIO, 2016) proposed an RNN model to address a machine translation task. This task belongs to the NLP universe and is related to translating a sentence from one idiom to another. The RNN model incorporated an attention mechanism to determine which parts of the sentence to focus on. This method enables information to be distributed across the sequence of annotations, and the decoder can selectively access the relevant information as needed. This new approach outperformed other state-of-the-art recurrent models on the task of English-to-French translation (BAHDANAU; CHO; BENGIO, 2016). After that, the attention mechanism approach obtained a lot of focus and was used in several other domains.

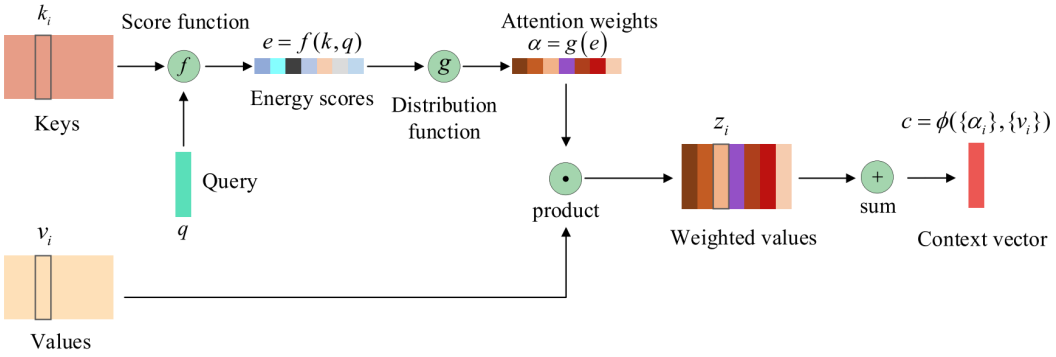


Figure 9 – Attention model diagram (NIU; ZHONG; YU, 2021).

Defining a general attention mechanism that can be applied to any sequence-to-sequence task is possible. This general form is depcited in Figure 9 and has three main components (CRISTINA, 2023):

- The **Query** (q) symbolizes the request for information (BRAUWERS; FRASINCAR, 2023). Each element in the input sequence may generate a q-vector that is compared against all available information.
- The **Keys** (k_j) describe the various pieces of information in the input data and determine which parts of the data are relevant to the Query. Keys can be represented differently depending on the specific tasks and neural architectures. For example, keys could be the features of a particular region in an image, word embeddings in a document, or the hidden states of RNNs (NIU; ZHONG; YU, 2021).

- The **Values** (v_j) represents the actual input information that needs to be aggregated or attended to. Each element in the input sequence is associated with a v-vector that is used to generate the final output based on the attention score obtained from the Query and Keys.

Figure 9 shows in detail the attention mechanism depicting the query, keys, and values as input for the entire model. The model outputs a context vector for each element (e.g., a single word) to be analyzed taking into account it's relationship with all other elements of the input sequence (NIU; ZHONG; YU, 2021). Finally, attention-based implementations must perform additional computation on the context vector to execute some tasks. Usually, another model - called the output model - runs predictions using the values of the context vector.

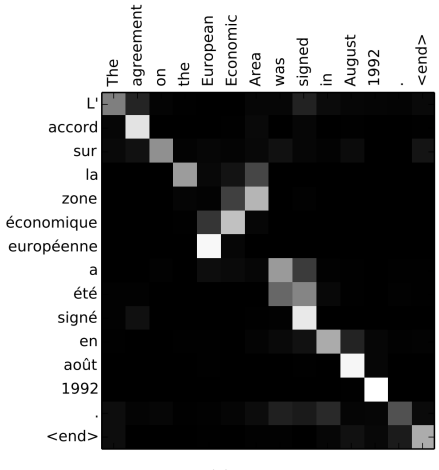


Figure 10 – Visualization of attention weight values for each token on an English-French translation task (BAHDANAU; CHO; BENGIO, 2016).

An essential inner component of the attention module is the matrix of attention weights. The attention weights offer a clear interpretation of the attention module. Each weight shows how important a specific feature vector is compared to the others for a given problem. Visualizing the attention weights matrix helps to better understand how the model behaves and to recognize the connections between the model inputs and outputs (BRAUWERS; FRASINCAR, 2023). Figure 10 shows the values for the attention weights matrix on the English-French translation task. It's possible to notice that the model gives higher weights to the words with the same meaning in both languages.

We hypothesize that the attention mechanism is especially well suited to elucidate k-mer relations in short reads obtained from RNA-seq data and likely will outperform

other approaches like CNNs and RNNs.

2.6 RNA SEQUENCE CLASSIFICATION USING KRAKEN2

When executing virus identification on RNA-seq samples, one popular deterministic and traditional approach is to use the “Kraken”. This tool was initially developed by Wood et al. (WOOD; SALZBERG, 2014) and further improved in a second version (WOOD; LU; LANGMEAD, 2019). Understanding how the first version works gives a better picture of how the second version works. Currently, Kraken2 is a widely used tool by bioinformaticians and researchers (LU; SALZBERG, 2020) and is also an integral part of the phytosanitary pipeline employed in this work.

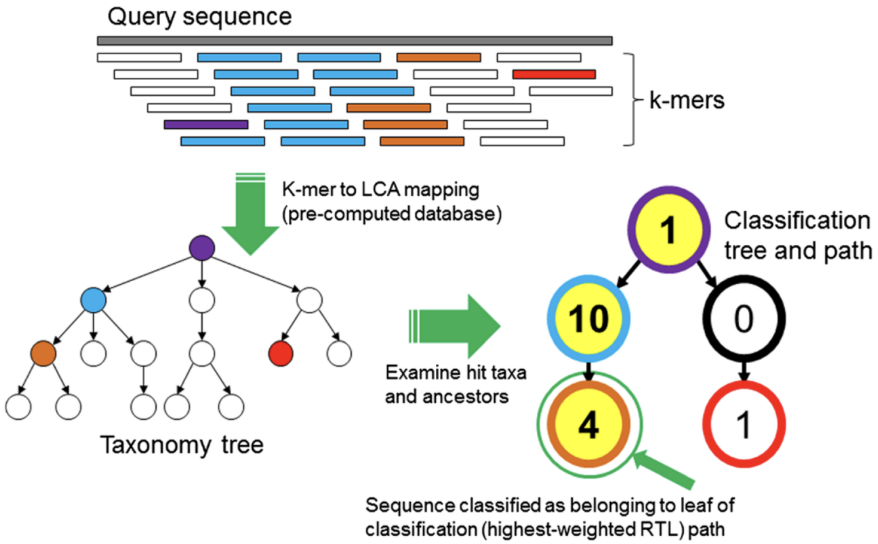


Figure 11 – The classification algorithm executed by the first version of Kraken (WOOD; SALZBERG, 2014).

As depcited in Figure 11, Kraken’s algorithm classifies sequences by mapping each k-mer from an input sequence to the Lowest Common Ancestor (LCA) of all reference sequences that contain that k-mer. This mapping is done using a pre-built database. The data for this database are sourced from the National Center for Biotechnology Information (NCBI). By default, Kraken uses complete microbial genomes, such as those from archaea, viruses, and bacteria, available in the NCBI Reference Sequence (RefSeq) collection. Alternatively, genomes from the broader Nucleotide (nt) collection can be used. Users can further customize the library by adding specific sequences as required. Once the base reference data are selected, Kraken utilizes the Jellyfish k-mer counting algorithm to generate a database that includes every distinct 31-mer

present in the reference library. Each k-mer is then associated with a taxonomic ID that represents the LCA of all RNA sequences containing that k-mer. This taxonomic information is also retrieved from the NCBI database (WOOD; SALZBERG, 2014).

Once the database is created, sequences can be classified. Figure 11 shows how the classification of sequences is executed as implemented by the first version of Kraken. The initial stage shown in the figure is the assignment of LCAs to each k-mer of the query sequence employing the pre-built database. Then, a classification tree is created from the taxonomic relations formed by the taxa associated with the sequence's k-mers and the taxa's ancestors. The numbers on the nodes in the classification tree represents weights equal to the number of k-mers in the sequence associated with the node's taxon. The next step is to create a classification path given by the root-to-leaf path with the maximum weight sum. The *leaf* of this classification path is the output class for the query sequence (WOOD; SALZBERG, 2014).

The second version of Kraken implemented some changes to improve the query performance. The main change is executing a hash function on the database's k-mers. This hash code is associated with the LCA information composing a database record. The Kraken2 algorithm achieved lower memory usage and higher processing speed during database queries. However, Kraken2 requires many computational resources to create and store the database, primarily if the complete NCBI "nt" data are used. The computation of the Kraken2 database usually takes several days to finish for large collections of reference genomes.

2.7 TAXONOMIC CLASSIFICATION OF RNA-SEQ DATA WITH KAIJU

Another traditional and deterministic tool used to classify RNA sequences is Kaiju proposed by Menzel et al. (MENZEL; NG; KROGH, 2016). The main difference between Kraken and Kaiju is that the second one is conceived to work mainly on the protein level, while Kraken is most used on the nucleotide level.

Figure 12 illustrates the steps executed by the Kaiju algorithm during sequence classification. The first step in Kaiju's workflow involves translating the nucleotide sequence into amino acid sequences (referred to as the protein level) by considering all six possible reading frames. These sequences are then split into fragments at stop codons. Depending on the execution mode, these fragments are either sorted by their

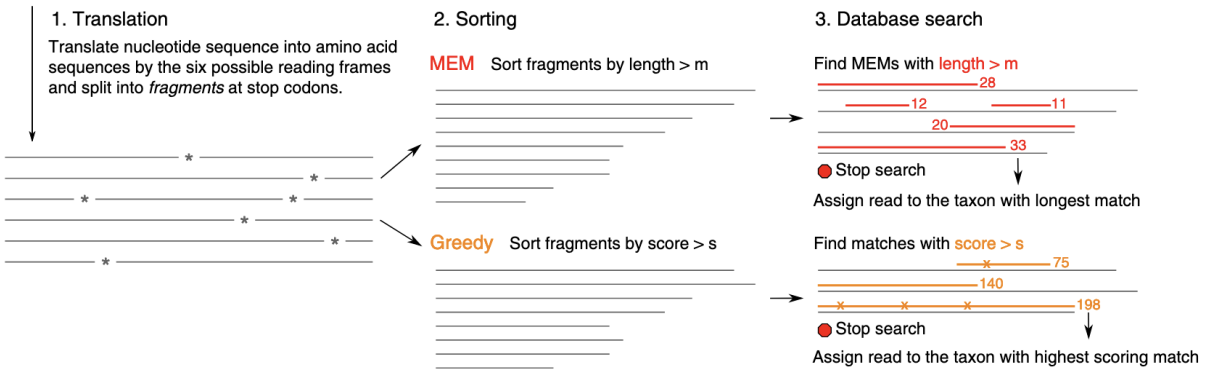


Figure 12 – The classification algorithm of Kaiju (MENZEL; NG; KROGH, 2016).

length (in MEM mode) or by their BLOSUM62 score (in Greedy mode). Once sorted, the list of fragments is queried against a reference database of microbial proteins (MENZEL; NG; KROGH, 2016). The classification of the queried sequence is based on the taxon of the best match found in the database.

In MEM mode, only exact matches are considered, focusing on the longest possible fragment matches. In contrast, Greedy mode analyzes approximate matches by extending exact matches at their left end, allowing for amino acid substitutions based on the BLOSUM62 substitution matrix. This sorting of fragments offers a significant advantage in speeding up the query process. By ranking the fragments before the database search, Kaiju can stop the search early when it is unlikely that a better match will be found. Specifically, in MEM mode, the search can be terminated once the remaining fragments are shorter than the length of the best match already identified. In Greedy mode, the search stops when the remaining fragments have a lower BLOSUM62 score than the best match found so far.

The BLOSUM62 score is used to evaluate how closely related two protein sequences are based on the likelihood of amino acid substitutions observed in evolutionarily related proteins. In Greedy mode, the score for each fragment can be computed before querying the database by applying the BLOSUM62 substitution matrix. This involves comparing the original amino acid sequence with modified sequences, allowing for substitutions to account for likely evolutionary variations. Higher scores indicate a greater likelihood that the fragment corresponds to a real protein sequence found in the database.

Kaiju also has a database build step that needs to be executed before any sequence classification can be performed. Like Kraken, it is possible to use the complete NCBI

database or its subsets, but all on the protein level. Kaiju runs the Burrows-Wheeler transform, converting the original reference sequence into an easily searchable representation (MENZEL; NG; KROGH, 2016). Also similar to Kraken, Kaiju requires high computational resources for database creation and storage, and building the Kaiju reference database usually takes several days.

2.8 BIOINFORMATICS CONCEPTS

This section presents some bioinformatics concepts that will help the reader to better understand the solution proposed by this work. Advanced DNA sequencing technologies, such as Next-generation Sequencing (NGS) and the more recent third-generation sequencing, provide an efficient and cost-effective way to explore the gene sequences of living organisms. These methods have also been adapted for RNA sequencing (RNA-seq), allowing researchers to detect and measure the expression levels of different RNA populations, including mRNA and total RNA. RNA-seq has revolutionized biomedical research by significantly enhancing the ability to study and interpret complex biological data (DESHPANDE et al., 2023).

2.8.1 Reads and Contigs

In NGS, a “read” is the nucleotide sequence describing a single fragment of RNA. Typically, NGS technologies break the RNA molecules into smaller fragments before sequencing, and each fragment corresponds to a read. The length and number of reads depend on the size of the fragments and the specific technology used. Since these RNA fragments usually overlap, the reads can be assembled to reconstruct the original molecules. However, some NGS methods do not involve fragmenting; these are known as long-read sequencing techniques, as they produce much longer continuous reads (READ, 2019).

When one assembles many reads together, the resulting sequence is called a “contig”. By definition, contig refers to a collection of overlapping RNA sequences that together form a continuous segment of a genomic region. The term “contig” comes from “contiguous”, reflecting how these sequences align to represent a stretch of RNA without gaps. For instance, a clone contig offers a physical map of overlapping cloned RNA

segments across a particular genomic region, while a sequence contig provides the complete RNA sequence for that region (CONTIG, 2024).

2.8.2 K-mers

K-mers are short sequences of nucleotides of length k that are derived from a longer DNA or RNA sequence. For example, if k = 3, the possible 3-mers from the sequence "AGCTGAC" would be "AGC", "GCT", "CTG", "TGA", and "GAC". The concept of k-mers is fundamental in bioinformatics, as these small sequences can be used to represent and analyze longer genetic sequences. K-mers are valuable because they capture information about the local structure and composition of a RNA sequence, which can be critical for various analyses such as sequence alignment, genome assembly, and motif finding.

In bioinformatics, k-mers are used extensively in tasks like genome assembly, where they help to piece together short reads of RNA into a complete sequence. By breaking down the reads into k-mers and finding overlaps between them, algorithms can reconstruct the original sequence. K-mers are also used in sequence comparison and searching, where the presence, absence, or frequency of specific k-mers can indicate similarities or differences between sequences. Additionally, k-mers are employed in tasks like error correction in sequencing data and identifying unique genomic signatures, making them a versatile tool in bioinformatics research and applications.

2.8.3 FASTA and FASTQ Files

FASTA and FASTQ are both widely used text-based file formats for storing nucleotide sequences, such as DNA and RNA, and can also store amino acid sequences for proteins in the case of FASTA. While they both serve as repositories for sequence data, they have different purposes and structures.

The FASTA format is simpler and typically used to store plain sequence data without any additional information. Each entry in a FASTA file begins with a single-line description, usually preceded by a ">" symbol, followed by the sequence itself. This format is versatile and is commonly used in various bioinformatics applications, particularly when sequence alignment, searching, or database queries are needed. FASTA

can handle both nucleotide and protein sequences, making it a fundamental format for many types of biological data.

On the other hand, the FASTQ format is more complex and is specifically designed to store nucleotide sequences along with quality scores for each base. This additional information is crucial in Next-generation Sequencing (NGS) workflows, where assessing the accuracy of each base call is important. Each entry in a FASTQ file consists of four lines: a sequence identifier, the nucleotide sequence, a plus sign (which may be followed by the same identifier), and a line of quality scores corresponding to the sequence. These quality scores, usually encoded as ASCII characters, help researchers determine the reliability of the sequencing data, making FASTQ the preferred format in NGS pipelines.

In summary, while both FASTA and FASTQ formats are used to store nucleotide sequences, FASTA is more straightforward and versatile, suitable for storing sequences without quality information. FASTQ, however, is specifically designed for next-generation sequencing data, providing both the sequence and the corresponding quality scores, which are essential for evaluating the accuracy of sequencing results.

2.8.4 Forward and Reverse Reads (R1 and R2)

In paired-ended sequencing, there are two ways of reading the sequence: forward read and reverse read. It improves accuracy by providing information from both ends of the RNA fragment, allowing for better alignment and detecting structural variations, insertions, and deletions. Some FASTA or FASTQ data indicate the read direction in the file name: "sample_1_R1.fastq" or "sample_1_R2.fastq". This example refers to sequences or reads from a single origin but sequenced in a forward way (R1) and in the reverse way (R2).

Using R1 and R2 together helps accurately reconstruct the original RNA sequence, especially in repetitive or complex regions where single-end sequencing might struggle. By overlapping or bridging gaps between the paired reads, researchers can achieve higher confidence in sequence assembly, mapping, and variant calling, making R1 and R2 crucial in high-throughput sequencing workflows.

3 RELATED WORKS

3.1 VIRAL GENOME DEEP CLASSIFIER

Fabijańska et al. (FABIJAńSKA; GRABOWSKI, 2019a) proposed a CNN-based model to execute virus classification from RNA-seq data. The paper details the model architecture, the dataset construction, and how the training process was conducted. The manuscript also compares the proposed model with other well-established tools. The reference provides clues on how to use deep learning models for RNA sequence classification tasks.

The model aims to propose a feature-based method for classifying genomic sequence data, detecting the presence of one of the following virus types: Dengue, Hepatitis B, Hepatitis C, HIV-1, or Influenza A. The first step is to create a dataset that will be used to train and evaluate the model's performance. All sequence data was gathered from NCBI (NCBI, 2024), LANL (LANL, 2024), and HBVdb (HBVDB, 2024) databases. The encoding strategy was to replace the nucleotide symbols - A, C, G, and T - with the corresponding ASCII code - 65, 67, 71, and 84, respectively.

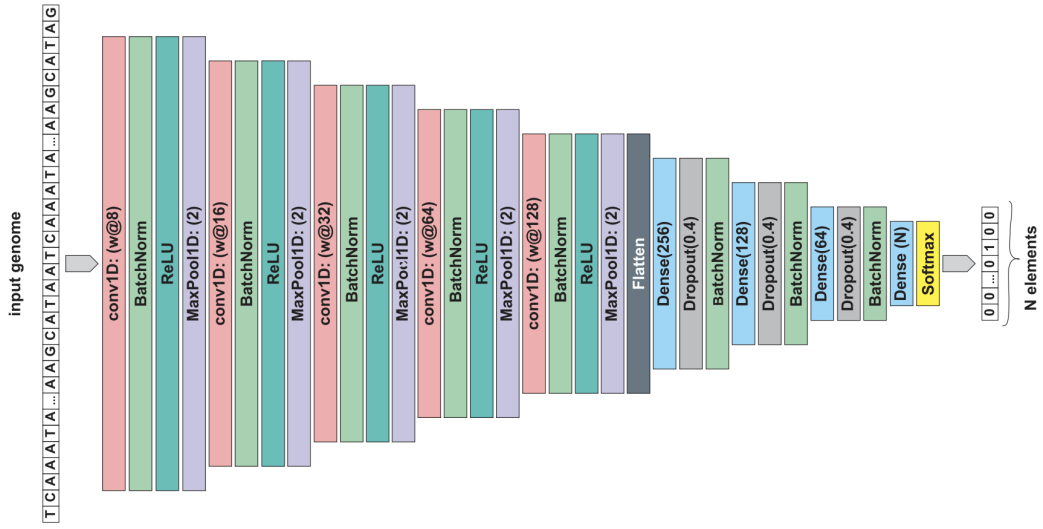


Figure 13 – Architecture proposed to detect virus presence in a genomic sequence (FABIJAńSKA; GRABOWSKI, 2019a).

Figure 13 shows the architecture of the proposed model. Essentially, the model uses uni-dimensional convolution layers and fully connected layers, where the first one is responsible for extracting features from the input sequence, and the second is responsible for using those features to perform the classification. The option for a net-

work with five convolutional layers architecture was due to challenges when classifying longer genomes like Dengue, HIV-1, and Hepatitis C.

Finally, the model was trained using a K-fold cross-validation strategy, where 80% of the dataset was used for training and 20% for validation. Analyzing the performance metrics and performance comparison with state-of-the-art approaches, it was possible to conclude that the model can execute its proposed task with high confidence.

3.2 IDENTIFYING VIRUSES FROM METAGENOMIC DATA USING DEEP LEARNING

Ren et al. (REN et al., 2020) state that the identification of viral sequences from metagenomic samples is a crucial step for the analysis of viruses. This work proposes a deep-learning CNN-based model for predicting viral sequences. The proposed model can be seen as a successor to the VirFinder model, a state-of-the-art model also used to identify viral sequences but not using a deep learning approach.

The model was trained using viral sequences and prokaryotic (bacteria and archaea) sequences obtained from the NCBI database. The viral genomes are fragmented into non-overlapping short sequences to mimic the real metagenomic contigs. The authors employed one-hot encoding as representation strategy with a "N" symbol coded as $[\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}]$. This sepcial symbol is required to represent ambiguous nucleotides.

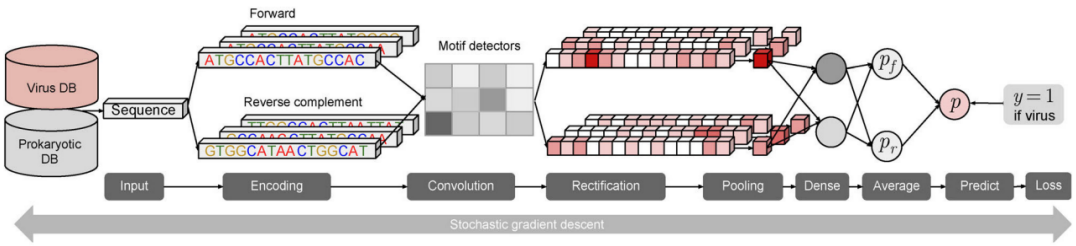


Figure 14 – DeepVirFinder model architecture (REN et al., 2020).

Figure 14 shows the architecture of the model. Basically, it consists of convolutional, max pooling, and fully-connected layers and outputs a prediction score between 0 to 1 for a binary classification between virus and prokaryote. The DeepVirFinder outperformed other state-of-the-art models, and significantly improved the accuracy of virus identification.

3.3 CNN-MGP: CONVOLUTIONAL NEURAL NETWORKS FOR METAGENOMICS GENE PREDICTION

Al-Ajlan et al. (AL-AJLAN; ALLALI, 2018) propose a CNN-based model to classify metagenomic fragments executing a gene prediction task. Metagenomics involves analyzing the genomes found in environmental samples like soil, seawater, and human gut samples. This paper also compares the performance of the proposed model with other machine learning-based methods.

The paper uses the Orphelia (ORPHELIA, 2024) and MGC (ALLALI; ROSE, 2013) datasets to train and evaluate the model's performance. The dataset includes millions of Open Reading Frames (ORFs) extracted from 700nt long fragments and was divided into ten parts to run the training step in a K-fold approach. One-hot encoding was used as a strategy to convert the sequence symbols into a representation that the model can process. Figure 15 shows how the one-hot encoding works. Each nucleotide is represented by a unique one-hot vector: A = 1000, T = 0001, C = 0100 and G = 0010.

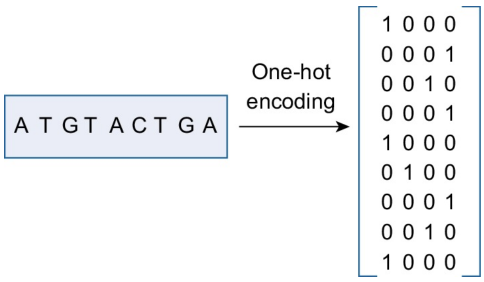


Figure 15 – One-hot encoding for a genomic sequence (AL-AJLAN; ALLALI, 2018).

Since the input is converted from an one-dimensional vector of nucleotides to a two-dimensional matrix, the authors use layers of two-dimensional convolution together with a fully connected layer as model architecture. The performance metrics show that the model was 98% accurate, and the work concludes that the proposed approach performs well compared to state-of-the-art gene prediction programs.

3.4 PACIFIC: A LIGHTWEIGHT DEEP LEARNING CLASSIFIER OF SARS-COV-2 AND CO-INFECTING RNA VIRUSES

Mateos et al. (MATEOS et al., 2021) emphasize that the Polymerase Chain Reaction (PCR) is a suitable COVID-19 identification method but can't identify co-infecting

viruses. Identifying co-infection is essential because it can alter disease severity and modify survival rates. High-throughput RNA sequencing (RNA-seq) offers an unbiased way to collect information about the RNA molecules in a sample, allowing for the systematic detection of SARS-CoV-2 infections and co-infections. To fast and reliably identify in a clinical setting virus infections from patient blood samples, the authors developed PACIFIC, a lightweight deep-learning model that detects SARS-CoV-2 and other common respiratory RNA viruses in RNA-seq data as shown in Figure 16(a).

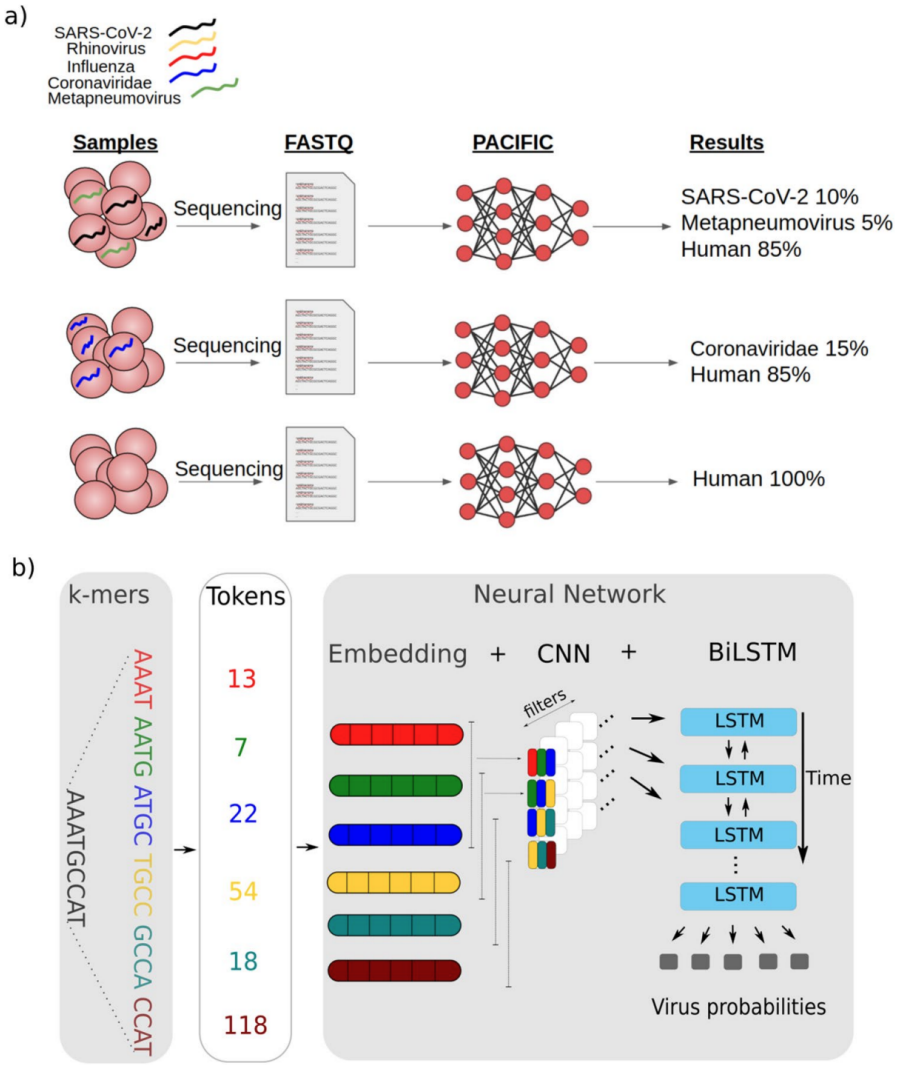


Figure 16 – PACIFIC model overview: (a) Required steps from sample collection to virus identification; (b) The model's architecture (MATEOS et al., 2021).

PACIFIC classifies RNA-seq reads into five respiratory virus classes — SARS-CoV-2, Rhinovirus, Influenza, Coronaviridae, and Metapneumovirus — and a sixth class for reads originating from the human genome. The model architecture is composed of an embedding layer, a CNN, and a Bi-directional Long Short-term Memory (BiLSTM)

network that ends in a fully connected layer. Figure 16(b) depcits the model architecture. An important aspect of this work is the direct encoding of k-mers and the use of an embedding layer. As shown in Figure 16(b), this encoding strategy transforms a group of nucleotides into a numerical representation. The authors state that using an embedding layer improved the model performance compared to a model without this layer.

PACIFIC was trained using 7.9 million 150nt random fragments from 362 viral genome assemblies, categorized into five viral classes (SARS-CoV-2, Influenza, Metapneumovirus, Rhinovirus, and Coronaviridae) along with the human transcriptome. In silico fragments from both strands were generated without errors to support paired-end sequence reads and capture natural genome variation within each class. 90% of the data was allocated for training and 10% for tuning hyperparameters and network architecture. The chosen virus classes aimed to ensure accurate detection of SARS-CoV-2 as a distinct class, distinguish it from other coronaviruses, include viruses recently reported to co-infect with SARS-CoV-2, and focus on viruses listed in the NCBI taxonomy database with humans as hosts. Additionally, the human class was included to prevent the misclassification of human reads into viral classes, as most sample reads are expected to be from human RNA.

The 362 genomes corresponding to the five classes of single-stranded RNA viruses were downloaded from the NCBI assembly database. The separate class for SARS-CoV-2 contained 87 assemblies. The Coronaviridae class contained 12 alpha, beta, gamma, and unclassified coronavirus genomes. The Influenza class included assemblies of influenza A and B viruses. For the Rhinovirus class, assemblies of rhinovirus A, B, C, and unlabelled enterovirus were grouped. Five distinct assemblies for metapneumovirus were grouped into a single class. Human GENCODE47 canonical transcript sequences (downloaded from the Ensembl v99 database) were included as an additional class to distinguish sequencing reads derived from the human transcriptome. A custom script generated between 0.44 and 3.5 million 150nt-long fragments in silico for each class. These training sequences were randomly sampled without base substitutions and were derived from both strands of the genome assemblies.

The authors concluded that PACIFIC can be used as a powerful end-to-end tool to identify virus infection from RNA-seq data with high sensitivity and specificity. The model achieved more than 99% accuracy for virus detection, enabling the systematic

identification of co-infections to improve clinical management and surveillance during the COVID pandemic.

3.5 PPR-META: A TOOL FOR IDENTIFYING PHAGES AND PLASMIDS FROM METAGENOMIC FRAGMENTS USING DEEP LEARNING

Fang et al. (FANG et al., 2019) propose a model to identify phage and plasmid fragments called PPR-Meta. Effectively identifying these elements is still quite challenging. Currently, the fragment assembly performance for both plasmid and phage sequences from high-throughput sequencing data is not as good as for host-derived fragments. This suggests that phage or plasmid sequences are present as many short fragments, making their identification difficult. Despite the difficulty, several tools were proposed to detect phages or plasmids from metagenomic data. Even though related tools have been created, the latest tools for detecting short fragments have not performed satisfactorily. This scenario motivated the proposal of a model for identifying phages, plasmids, or chromosomes using deep learning approaches.

Due to the lack of real metagenome datasets with reliable annotations, simulated datasets with artificial contigs generated from fully sequenced genomes were used to benchmark the study. Complete genomes of prokaryote chromosomes, plasmids, and phages, including prophages predicted by ProphET (PROPHET, 2024), were downloaded from the NCBI (NCBI, 2024) genome database and organized into training and test sets. Additionally, MetaSim (METASIM, 2024) was used to create artificial contigs of varying lengths to simulate different sequencing technologies, and real metagenomic data were used to estimate PPR-Meta's reliability.

The paper states that k-mer frequencies are a popular approach to represent biological sequences, but they perform poorly when used in short sequences. This motivated the authors to propose a different application of one-hot encoding. Each sequence is represented in two ways: A one-hot matrix for the nucleotide base sequence (BOH) and a one-hot matrix for the codon (i.e., a group of three nucleotides) sequence (COH). The first one is a simple one-hot representation of the sequence, giving each nucleotide a unique feature vector, so A, C, G, and T are represented by [0,0,0,1], [0,0,1,0], [0,1,0,0], and [1,0,0,0], respectively. The second representation strategy expands the sequence in codons and assigns to each codon its one-hot vector. Since there are 4^3 codons,

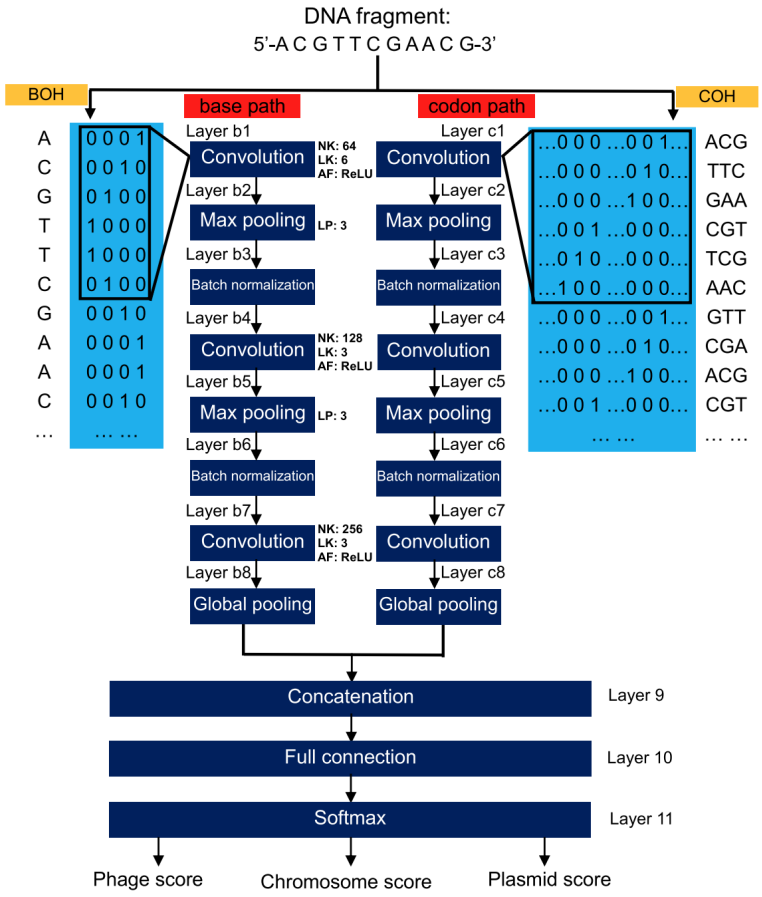


Figure 17 – PPR-Meta model architecture: One model path processes the one-hot matrix for the nucleotide bases and the other one the one-hot matrix for codons (FANG et al., 2019).

the corresponding feature vectors have 64 elements. Figure 17 shows the approximate representation of the following example sequence 5'-ACGTTCGAACG-3'.

The model implements a strategy to process both sequence representations in parallel. The authors call this architecture a BiPathCNN. As shown in Figure 17, the model has two paths of bi-dimensional CNN layers. Both paths have CNN, Max Pooling, and Normalization layers ending in a global pooling layer. Finally, the features extracted by the paths are concatenated and processed by a fully connected layer that will output the classification result. Four models were trained for different sequence lengths, namely, 100-400nt, 400-800nt, 800-1200nt, and 5000-10000nt, to ensure the model can adapt to different sequencing platforms.

The paper concluded that PPR-Meta has shown better performance than similar tools such as VirFinder, VirSorter, PasFlow, and cBar. So, the PPR-Meta tool is expected to meet the demand for metagenomics analysis when considering the microbial community tangled with phages and plasmids. It certainly qualifies as a powerful tool

for the research community.

3.6 XVIR: A TRANSFORMER-BASED ARCHITECTURE FOR IDENTIFYING VIRAL READS FROM CANCER SAMPLES

Consul et al. (CONSUL; ROBERTSON; VIKALO, 2023) propose a model to identify viruses from sequence information extracted from human cancer samples. This work is motivated by the understanding that viruses linked to human tumors produce viral oncoproteins that affect the regulatory processes in host cells, eventually leading to tumor development. High-throughput sequencing of genomic content of tumor cells produces a massive amount of short sequenced reads. The main task of the proposed model is viral identification, i.e., the model tries to identify if those short reads have viral origin or not.

The dataset is built with artificially generated 150nt-long sequences from Human Papillomavirus (HPV) and human genomes. The sequences are partitioned into training (80%), validation (10%), and test (10%) data sets. Sequenced reads are represented as a series of k-mers. The authors varied the length of the k-mers from 3 to 7 nucleotides choosing 6 as the optimal value.

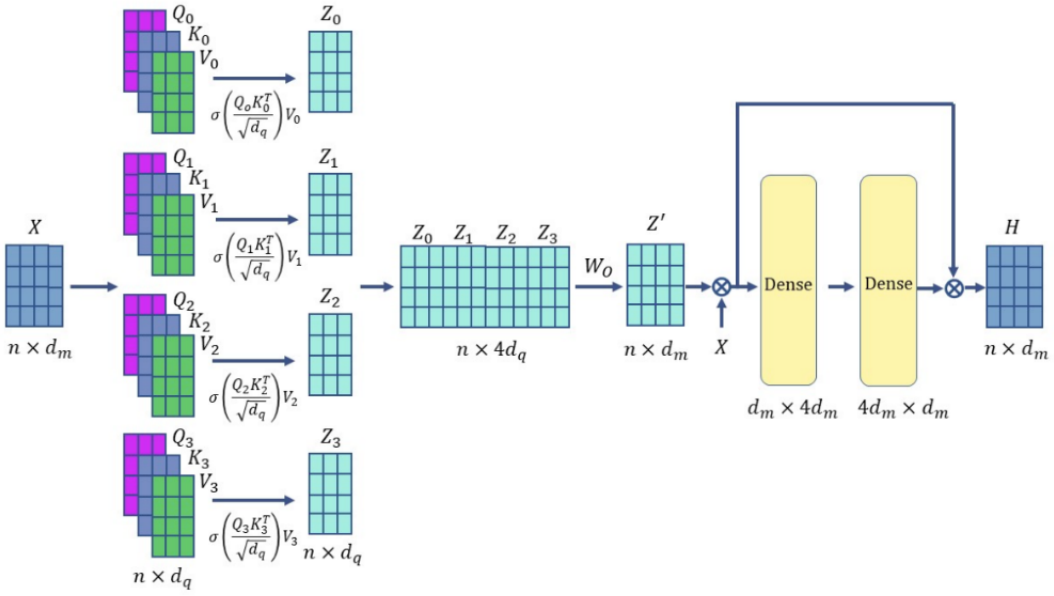


Figure 18 – XVir model architecture (CONSUL; ROBERTSON; VIKALO, 2023).

Figure 18 shows the architecture of the model. The first step is k-mer mapping. The embedding includes positional information. After this step, the encode step em-

ploys a multi-headed self-attention mechanism that leverages learned relationships of k-mers in viruses. Every matrix produced by each of the attention heads is concatenated, condensed, and, finally, passed to a feed-forward network that will output the predicted probability for the read having a viral origin. The authors showed that the model achieved performance comparable with others optimal models like DeepVirFinder but having 25% less parameters.

3.7 VIRHUNTER: A DEEP LEARNING-BASED METHOD FOR DETECTION OF NOVEL RNA VIRUSES IN PLANT SEQUENCING DATA

Sukhorukov et al. (SUKHORUKOV et al., 2022) propose a model to detect (known and novel) RNA viruses in plant sequence data. The deep-learning model architecture is based on CNNs trained to identify viruses in *assembled* sequences. The authors compared the performance of VirHunter with traditional alignment-based tools like translated BLAST (tBLASTx, alignment of non-redundant nucleotide sequences based on the encoded proteins) (TBLASTX, 2024) and with other state-of-the-art deep-learning sequence classification models like DeepVirFinder.

The paper downloaded extensive training data from several NCBI databases (NCBI, 2024), using different approaches for the various classes. Figure 19 gives an overview about the model. The authors developed a model with three classes - Bacteria, Virus, and Plant. Model building started with downloading all viruses that infect plants, i.e., viral sequences with a host taxonomic ID equaling *Viridiplantae*. The reference genomes for the following plant species were added: *Prunus persica* (peach), *Vitis vinifera* (grapevine), *Beta vulgaris* (sugar beet), and *Oryza sativa* (rice). Finally, the authors downloaded all representative bacterial reference genomes from NCBI RefSeq. As part of testing the performance of the model, 12 novel virus sequences were assembled from RNA-seq data extracted from peach, grapevine, and sugar beet samples. All downloaded sequences were artificially fragmented into 500bp and 1000nt. The model uses one-hot encoding as the input's representation strategy.

Figure 19(b) shows the architecture of the model. The proposed approach uses a two-path CNN model. One path processes the fragmented raw sequence, and the other processes its reverse complement. Each path implements a simple pattern of a single convolutional layer with a pooling layer. Then, both paths are flattened, concate-

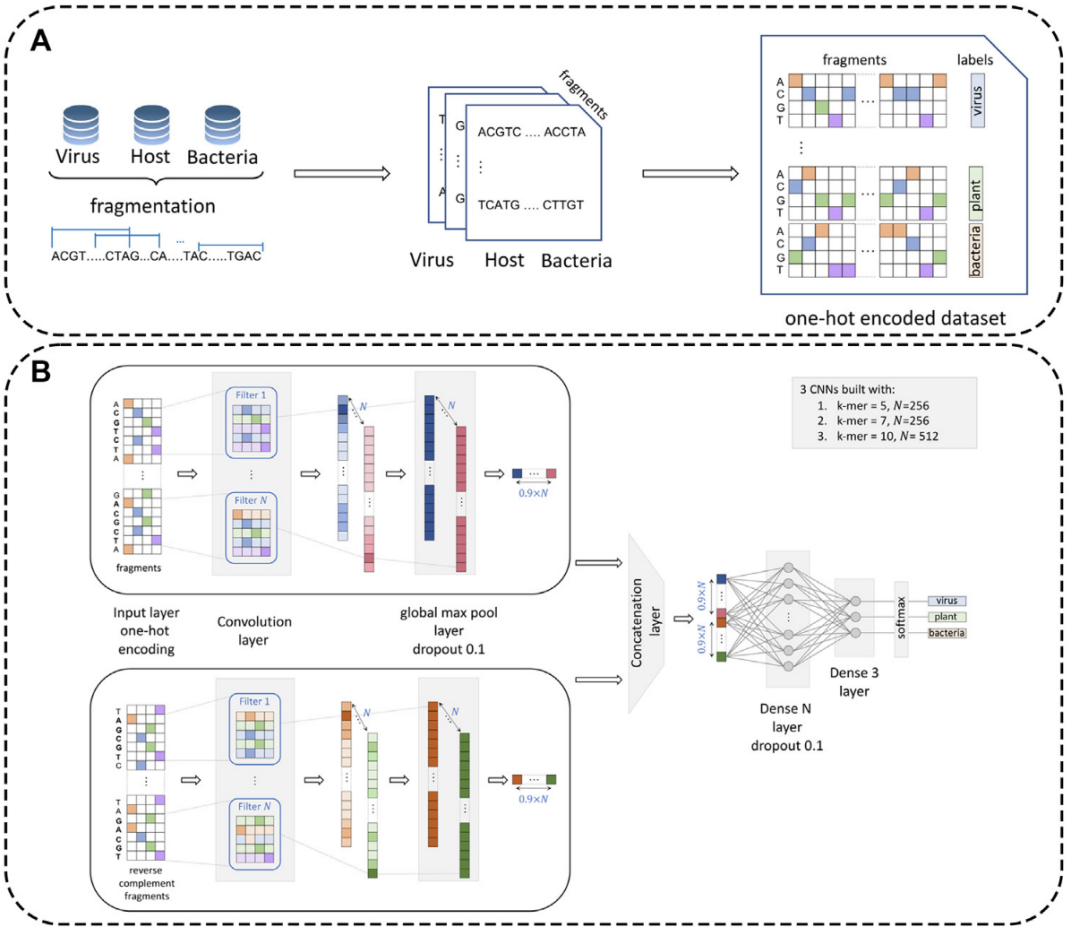


Figure 19 – The preprocessing of the reference data for training the VirHunter model (a) and the architecture of the model (b) (SUKHORUKOV et al., 2022).

nated, and passed as input to a fully connected layer to compute the output probability distribution. The authors used one-hot encoding to convert the input text into a numerical representation but also interpreted the CNN filters as a k-mer processing. In other words, when the CNN filter is set to have a size equal to 5, it works similar to 5-mer extraction. When using CNNs, the convolutional step extracts the features. Therefore, in the context of processing RNA sequences, convoluting the nucleotide string of symbols generate nt groups that serve as characteristic features of the sequence.

Figure 20 shows how the classification works. The prediction module is composed of three CNN models trained for the following filter sizes: 5, 7, and 10. In addition to the three CNN models, the model also contains a random forest classifier to assemble the outputs of the CNNs and predict the final output using a majority vote strategy.

Figure 20 also illustrates how all parts of the prediction module were trained. All training steps were executed for each plant host species (peach, grapevine, and sugar beet) and fragment sizes of 500nt and 1,000nt. The CNN models were initially trained

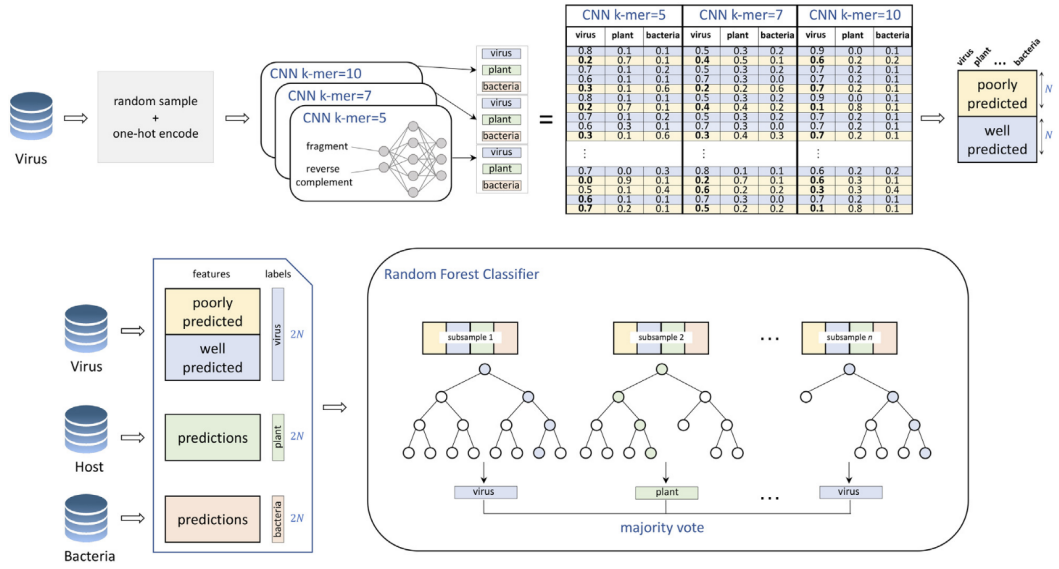


Figure 20 – The scheme for the entire prediction module. Three models were trained for different CNN filter sizes performaing feature extraction for distinct k-mer sizes (SUKHORUKOV et al., 2022).

separately, one for each filter size. The data for training the random forest model were selected as follows. First, 100,000 fragments were randomly selected from the output of the trained CNN models and split into two groups based on the prediction value for the viral class; the first group has values greater than or equal to 0.8, and the second group has values lower than 0.8. 10,000 fragments were randomly selected from each group to train and test the random forest model.

VirHunter's ability to detect novel viruses was validated using 12 newly acquired RNA-seq datasets from peach, grapevine, and sugar beet. In these datasets, at least 90% of all expert-annotated viral contigs were detected by VirHunter, compared to 73% by DeepVirFinder and 26% by VirSorter2. In seven of the datasets VirHunter achieved even 100% detection. Additionally, VirHunter produced a low rate of false positives, resulting in only 19 to 277 contigs per dataset needing expert inspection. These findings demonstrate that the use of VirHunter effectively reduces the number of contigs that require manual expert curation.

3.8 DIFFERENCES BETWEEN RELATED WORKS

This chapter presented different models to address similar challenges in virus detection in RNA-seq data. Although they share the same objective—to identify the presence of viruses in sequence data—the models differ in their deep learning approach,

maximum and minimum input size, supported hosts, etc.

Frame 1 shows a comparative analysis between the works presented in this chapter. Most solutions are CNN-based models trained for detecting viruses in human or prokaryote hosts. The models are trained to detect different virus species. An essential aspect of the proposed solutions is the input size; some models are trained using different sequence sizes, and others use sequences of fixed lengths. However, most models accept inputs longer than those used in the training set by expanding the input and adding special characters at the end of the sequence. Finally, all models work with nucleotide-level inputs; none of the models analyzed worked with protein-level inputs.

VirHunter stands out from the related approaches. Like the deep learning model proposed in this work, VirHunter detects viral sequences in RNA-seq data acquired from plant samples. Thus, the used training data and procedure is similar. However, there are significant differences: VirHunter is CNN based, uses viral sequences on nucleotide level as reference, and classifies contigs instead of reads, i.e., nucleotide fragments with a larger length than typical reads obtained from HTS. Thus, in the context of a phytosanitary pipeline, VirHunter is a tool employed downstream of the classification flow whereas the proposed model here addresses challenges in the upstream part of the pipeline.

Paper Title	Objective	Architecture	Ref. Hosts	Ref. Viruses	Input Size
Viral Genome Deep Classifier	Identify viruses from genomic sequences.	CNN-based	No Host	Dengue, Hepatitis B, Hepatitis C, HIV-1, Influenza	All inputs are extended to the same length as the largest sequence present on training set: 24307nt
Identifying Viruses from Metagenomic Data using Deep Learning	Identify viral sequences in metagenomic data using deep learning.	CNN-based	Prokaryotes (bacteria and archaea)	Viruses infecting prokaryotes (bacteria and archaea).	Different models were developed to work with different input sizes. Three sizes are supported: 300nt, 500nt, and 1000nt.
CNN-MGP: Convolutional Neural Networks for Metagenomics Gene Prediction	Metagenomics gene prediction.	CNN-based	Prokaryotes (bacteria and archaea)	No Pathogens	Fixed input size: 700nt.
PACIFIC: A Lightweight Deep Learning Classifier of SARS-CoV-2 and Co-infecting RNA viruses	Detect SARS-CoV-2 and other common RNA respiratory viruses from RNA-seq data.	CNN and LSTM	Human	SARS-CoV-2, Influenza, Metapneumovirus, Rhinovirus, Coronaviridae	Fixed input size: 150nt
PPR-Meta: A Tool for Identifying Phages and Plasmids from Metagenomic Fragments using Deep Learning	Identifying phages and plasmids from metagenomic fragments.	CNN	Prokaryotes	Phages	Variable input size: From 100nt to 10knt
XVir: A Transformer-Based Architecture for Identifying Viral Reads from Cancer Samples	Identify viral reads from human cancer sequences.	Attention-based	Human	Human Papillomavirus	Fixed input size: 150nt
VirHunter: A Deep Learning-Based Method for Detection of Novel RNA Viruses in Plant Sequencing Data	Identify known and novel virus in Plant sequence data	CNN-based and Random Forest	Peach, Grapevine, and Sugar beet plants	*Viridiplantae* and Reference Bacterias	500nt and 1,000nt

Frame 1 – Comparative analysis of the related works presented in this chapter.

4 PROPOSED APPROACH

The previous chapter presented approaches to run viral identification or sequence classification tasks. Most models perform virus identification that infects humans or prokaryotes, working at the nucleotide level and accepting sequences or reads with 300nt or larger. This work developed a deep learning model to detect viral plant infection based on RNA-seq data obtained from samples of leaves or roots. RNA-seq data are not encoded directly but first translated to the protein level. The main goal is to improve the identification of sequenced reads of viral origin including both known and novel viruses.

4.1 MODEL ARCHITECTURE

This work proposes to improve HTS-based plant disease diagnostics by developing a novel deep-learning model for detecting viral reads in RNA-seq data of cassava plants. The main innovations in the proposed approach compared to most state-of-the-art deep-learning models for RNA-seq based virus detection described in the literature are the following:

- Classification of sequenced reads that are not pre-assembled
- Data encoding on the protein level
- Use of multi-head attention for feature extraction of k-mer relations

Figure 21 shows an overview of the proposed solution. The first step for read classification is the translation of the short sequences from the nucleotide level to the protein level. Every input nucleotide (nt) read generates 6 translated amino acid (aa) reads, three aa reads originate from translating the RNA sequence from left to right (starting from the first, second and third nucleotide in the sequence) and three from translating in the reverse direction. If a read on the protein level contains a stop codon, this read is discarded from the input. We hypothesize that such reads do not contain sufficient information to be reliably classified.

All further steps are executed for each protein-level translated read that does not contain the stop codon. Reads are encoded as a series of k-mers. A k-mer length of

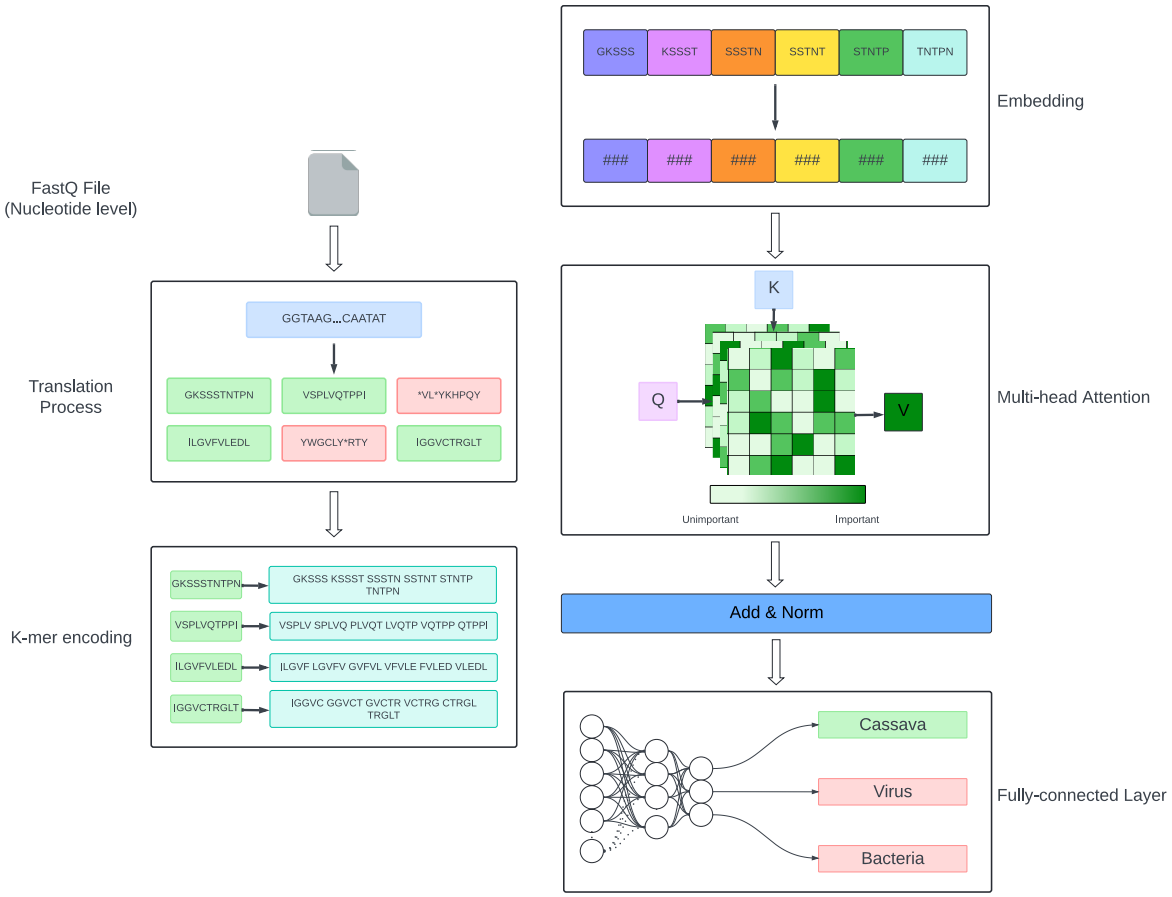


Figure 21 – Overview of the proposed model. On the left, the data preprocessing is illustrated that consists of translating reads from the nucleotide level to the protein level and of k-mer encoding for $k = 5$. On the right side, the model architecture is detailed. The model executes first the embedding layer, then the attention layer, and finally, a fully connected layer outputs the prediction values for each class.

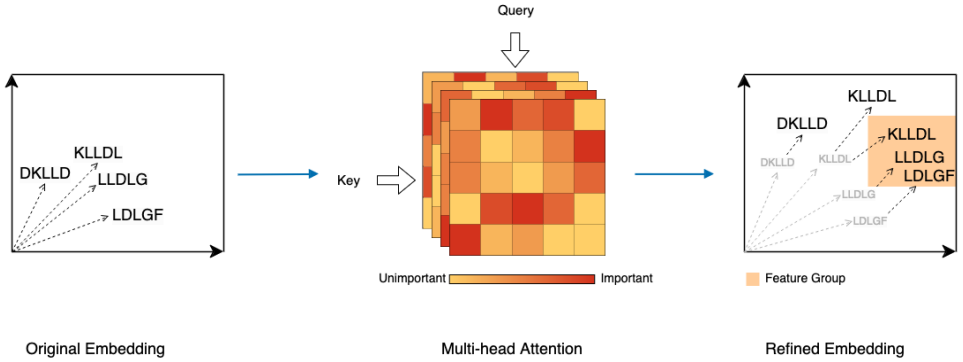


Figure 22 – Illustration of the process of embedding refinement.

$k = 5$ amino acids is chosen in the present work. The main step is the application of the deep learning model, shown on the right side of Figure 21. The first step of the model execution is the embedding layer that creates a numeric representation for every k-mer token in the sequence, also adding the positional information for every token. Then, the multi-headed attention step is executed, i.e., the model identifies all important k-mer relations for the final classification. As shown in Figure 22, this process can be seen as an embedding refinement where each k-mer is grouped in feature groups making it easier for the model to execute the classification step. Final classification runs on a fully connected network that will output the probability for every output class (plant, virus, or bacteria) indicating the likely origin of the analyzed sequenced read.

4.2 PERFORMANCE EVALUATION

The proposed machine learning model is trained with RNA sequences obtained from the NCBI nt database. All available data are used for training and validation. To test the model, RNA-seq data obtained from cassava leaves and roots were provided by the Phytovirology Laboratory at UFRPE. However, these samples needed to be analyzed first by state-of-the-art bioinformatic tools to serve as labeled benchmark data.

None of the DL models described in the literature were trained for detecting cassava infecting viruses. Moreover, most models classify contigs, i.e., assembled reads. However, assembling all sequenced reads present in a RNA-seq data comes at high computational cost. In addition, assembly can introduce errors, e.g., in the form of creating chimeric sequences, i.e., artifact sequences formed by two or more biological

sequences incorrectly joined together. Out of these reasons, a direct comparison with other DL models was out of scope of the present work.

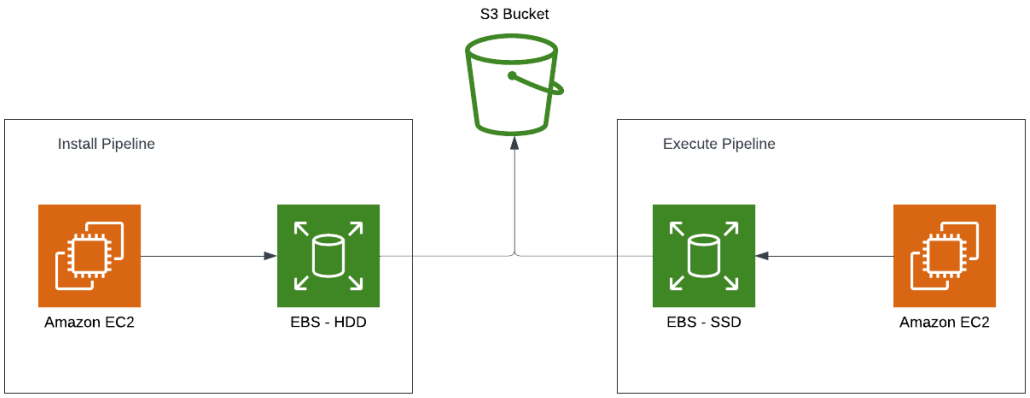


Figure 23 – Resources deployed on the Amazon Web Services (AWS) Cloud to run Phytopipe, a state-of-the-art phytosanitary bioinformatics pipeline used to label RNA-seq data obtained from cassava plants.

The outlined considerations led to the decision to implement and apply a state-of-the-art phytosanitary bioinformatics pipeline for plant pathogen detection. The pipeline of choice was PhytoPipe (HU et al., 2023) since the pipeline is open source, well documented, and used by the USDA Plant Germplasm Quarantine Program for HTS-based diagnostics. Due to the high computational demands, the pipeline was deployed on the Amazon Web Services (AWS) Cloud. Figure 23 shows the resources used to build and run Phytopipe. Once this pipeline was deployed, it was possible to label the RNA-seq data obtained from cassava plants and assess the performance of the proposed DL model. Moreover, a deeper understanding of the bioinformatic tools employed in HTS-based disease diagnostics helps to identify the challenges that deep learning models might able to overcome. In the context of the present work, the developed DL model could select the sequenced reads that should be passed to the assembly unit, thereby significantly decreasing the computational costs of any phytosanitary pipeline.

5 METHODS

This chapter details the applied methods for executing and evaluating experiments to validate a working model for classifying sequenced reads at the protein level obtained from RNA-seq data obtained from cassava plants. Section 5.1 explains how the dataset for training the model was built. Section 5.2 shows the approach used to conduct the experiments for building and evaluating the models proposed. To test the performance of the model, samples were classified collected from cassava plants. These samples are described in Section 5.3. Finally, Section 5.4 discusses the setup used to run all experiments.

5.1 DATASET CREATION

An important task in constructing the proposed model was the creation of the dataset. A challenge faced by this work was the unbalanced nature of the data that could be used to train the model. RNA sequences are retrieved from the NCBI database and artificially fragmented into short reads. The model shall sort the reads in three classes: host, virus and bacteria. Thus, a naive approach would include the genomes of cassava (the host), all viruses (taxid=10239) that are known to infect green plants (host *viridiplantae*, taxid=33090) and all bacteria . Note that for bacteria the database does not allow to specify a host taxid. Table 1 shows that this approach leads to a very unbalanced dataset.

Species	**Download Size (Nucleotide Level)**	**Download Size (Protein Level)**
Cassava (*Manihot esculenta*)	639.6 Mb	28 Mb
Viruses (host is *Viridiplantae*)	370 Mb	84 Mb
Bacteria (RefSeq)	23 Gb	15 Gb

Table 1 – Download size of sequence data for output classes chosen as reference for the model for both Nucleotide and Protein levels. The amount of data is very unbalanced. Source: NCBI.

Another challenge was that the dataset often occupied more space in RAM than the download size during the training of the deep learning model. Moreover, the final model should be lightweight to allow its routinely application in a phytopathological clinic facility with access to only limited hardware resources.

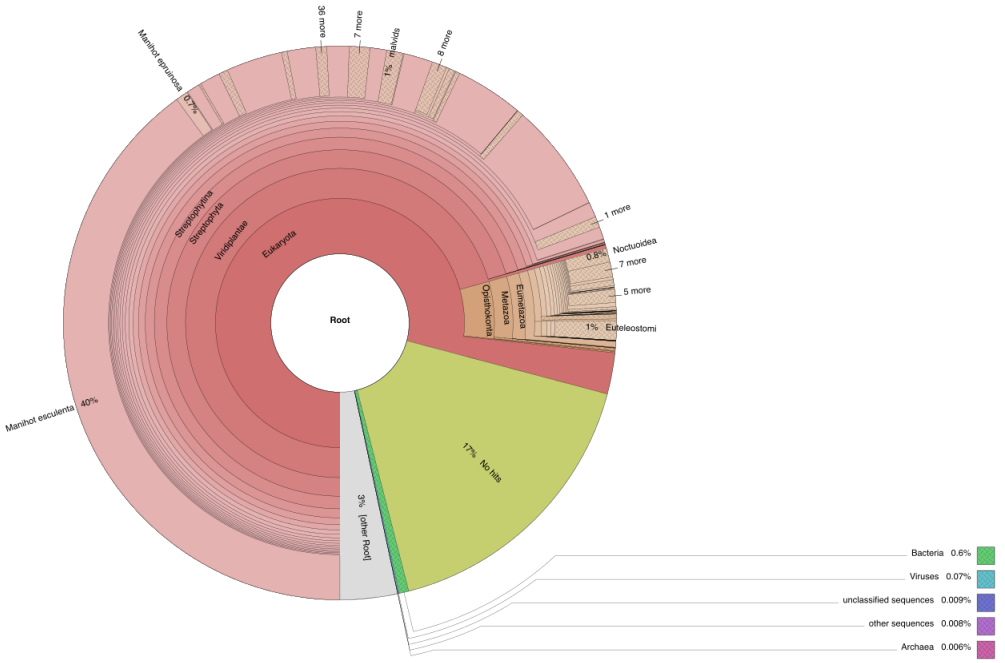


Figure 24 – Krona chart built with results of the execution of Kraken2 tool. It presents the taxonomy of every organism present in RNA-seq data.

These facts motivated the analysis of a typical cassava microbiome in order to create a dataset that is representative, balanced and feasible to be used in model training on common commercial computing clusters. However, plant microbiomes are not yet well studied and understood. Instead of a profound analysis, a randomly selected RNA-seq data obtained from cassava plants with clear disease symptoms was characterized using tools like Kraken (WOOD; SALZBERG, 2014) and Kaiju (MENZEL; NG; KROGH, 2016). These tools identify all organisms present in the RNA-seq data by running search algorithms on tailored gigabyte-sized reference databases (WOOD; SALZBERG, 2014; MENZEL; NG; KROGH, 2016). Figure 24 displays the results obtained from the Kraken2 analysis in the form of a pie chart created by the Krona tool. This interactive pie chart shows the taxonomy hierarchy for every organism identified in the sample. When examined for several different RNA-seq data samples, a reasonable picture of the most common bacteria species present in the cassava microbiome may be obtained. Frame 2 gives on overview about the predominant bacteria identified by this analysis.

The data discrepancy and size also motivated us to move from identifying reads on the nucleotide-level to a protein-level approach. The FASTA files for proteins are smaller than for genome sequences since only annotated coding regions are translated to the protein level. In other words, difficulties of identifying all proteins encoded by

Class	Filters
Cassava	—
Virus	Host is Viridiplantae
Bacteria	Genus is Kitasatospora | Acinetobacter | Streptomyces | Pseudomonas or Phylum is FCB group

Frame 2 – Filters executed while downloading datasets from NCBI.

complete reference nucleotide sequences is one of the reasons for dramatically reducing the dataset size. Moreover, not only the size of the dataset is naturally decreased but also the discrepancy between the classes diminished (although not removed).

All the datasets were downloaded from the NCBI database. The cassava data was obtained by accessing the NCBI web page, searching for cassava, and accessing the download genome page, choosing only the protein level. The virus data was also downloaded via the NCBI web page, accessing the NCBI virus portal, searching by all viruses that infect plants (setting the filter to a host equal to *viridiplantae*). Finally, the bacteria data were downloaded using the NCBI download utility executed via the command line, which allows for filtering some taxonomy. Appendix E, Appendix F, and Appendix G show the procedure to download the viral, bacteria, and bacteria sequences procedure respectively. Frame 2 shows all filters executed on the different NCBI endpoints to build the reference data for our dataset.

The last step in the dataset construction was the artificial read generator. A script was executed to get random pieces of a RNA sequence in a way that there are no repeated regions or repeated reads. Table 2 shows the number of fragments generated for each class in the dataset. All fragments were 100aa long. Appendix A contains the Python code for the script that generates the random reads.

Class Name	Number of Reads	Reads/Protein Sequence
Cassava	14,762,927	300
Virus	14,406,394	67
Bacteria	12,818,976	4

Table 2 – Absolute and relative number of generated artificial reads per class included in the training dataset.

5.2 MODEL TRAINING AND OPTIMIZATION

All attempts to find the optimal deep-learning model architecture to execute virus identification tend to start with a baseline model inspired by related works and change the hyperparameters or even some minor architectural aspects of the model. The following subsections describe the experiments targeted to validate four aspects of the optimization:

- Encoding strategy
- Type of reference data
- Output class number
- Model architecture

5.2.1 Encoding Strategy

Initially, one-hot was chosen as the encoding strategy as it is an easy but effective way to transform the genomic symbols into a numeric and vectorized representation. Table 3 shows how the representation is built for every nucleotide symbol. This strategy was well-suited when working with CNNs because it transforms the uni-dimensional sequence information into a bi-dimensional one, making it possible to use bi-dimensional CNNs. Despite its popularity, a model developed in this work using this approach didn't perform well when classifying sequenced reads from cassava RNA-seq data as will be discussed later. Another challenge when choosing this encoding strategy is that the input data become sparse as the number of characters increases because most of the vector will be mainly composed of zeros. Also, the one-hot encoding isn't a good choice when employing NLP models.

The outlined limitations of one-hot encoding motivated the usage of k-mers. Figure 25 shows how the encoding strategy works for a k-mer length of $k = 5$. The encoding starts by getting the first 5 symbols, grouping them together as a unique feature group, a k-mer. Then, the encoding window shifts by 1 symbol to create the next feature group. The process is repeated until reaching the end of the input sequence generating several groups of symbols.

Nucleotide Character	Vector Representation
A	$[1, 0, 0, 0]$
T	$[0, 0, 0, 1]$
C	$[0, 1, 0, 0]$
G	$[0, 0, 1, 0]$
N	$[\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}]$

Table 3 – Vector representation of one-hot encoded nucleotide characters.

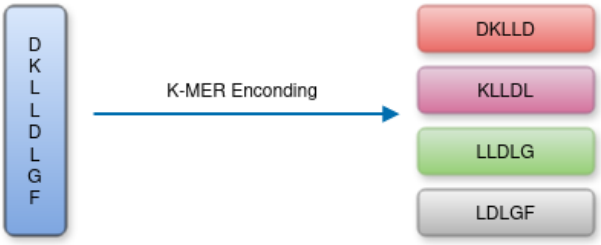


Figure 25 – Representation of k-mer encoding for $k = 5$. This is an example of an amino acid sequence, but the scheme works similar for nucleotide sequences.

This approach is very intuitive when working with RNA-seq data because a group of nucleotides encodes the proteins of an organism. Also, k-mers are the best choice when using a recurrent model because when transforming a sequence in a group of k-mers, it is possible to see the sequence as a group of words (the k-mers) and thus use embeddings.

Several experiments were conducted to find the optimal k-mer size. The tested k-mer length k varied from 3 to 11. Short k-mers might have the effect of not grouping enough symbols to represent relevant features. Long k-mers, on the other hand, might group too many symbols, creating embeddings with a high number of tokens. The size that presented the best performance was $k = 5$. Another important aspect that drives the choice of an optimal k-mer size is whether nucleotide or amino acid sequences are encoded. This is due to the fact that three consecutive nucleotides in an RNA molecule, a codon, translate into one amino acid. Thus, k-mer encoding at the protein level is equivalent to three times larger groups of nucleotides fragmenting the organism's genome.

5.2.2 Type of Reference Data

In principle, two types of reference data can be used to train the model: nucleotide sequences or amino acid sequences. In RNA-seq classification, the standard approach is often to classify samples at the nucleotide level. This preference is largely due to the abundance of nucleotide data in popular databases like NCBI and the straightforward implementation of models that operate directly on nucleotide sequences. Initially, our project also aimed to classify HTS data at the nucleotide level. However, this approach posed significant challenges, primarily due to the highly unbalanced nature and the size of the data. The genomic data from cassava and the viruses differ substantially in size and representation, complicating the classification task.

There are several biological and computational advantages to encoding amino acid sequences rather than nucleotide sequences. One key reason is the degeneracy of the genetic code, where multiple codons can encode the same amino acid. By translating nucleotide sequences into amino acid sequences, this degeneracy is naturally accounted for, which can simplify the comparison process. Furthermore, different viruses might code for similar proteins even if their nucleotide sequences are significantly different. Focusing on amino acid sequences can thus enhance the detection of viral infections and make it easier to discover novel viruses with divergent nucleotide sequences but similar protein products.

The HTS data analyzed by our model are derived from RNA molecules extracted from cassava samples, such as leaves or roots. These RNA molecules include messenger RNA (mRNA), ribosomal RNA (rRNA), transfer RNA (tRNA), and viral RNA if present. Typically, rRNA is depleted in the sequencing library. The use of the NCBI protein database is appropriate in this context, as it contains translated sequences from annotated coding regions, which are sufficient for identifying viral proteins.

To validate this approach, we conducted experiments comparing models with identical hyperparameters but operating at the nucleotide level versus the protein level. Implementing a model at the protein level necessitates a modification in the classification workflow during inference. Since high-throughput sequencing produces nucleotide reads, these reads must first be translated into amino acid sequences before classification can occur. Our results demonstrated that the protein-level model outperformed the nucleotide-level model, highlighting the effectiveness of this strategy in improving

the accuracy and robustness of virus detection in RNA-seq data.

5.2.3 Output Class Number

Since the model should detect the presence of viruses in cassava samples, a binary output might be sufficient: either a read is classified as cassava or virus. However, the experiments showed that this strategy does not provide a model with good performance. The reason is that organisms other than just viruses and plants (in this case, cassava) are also present in the sequence data. RNA-seq data obtained from cassava plants and analyzed with traditional tools like Kraken2 and Kaiju confirmed that the plant has a rich microbiome, see Figure 24. As it turned out, these reads, e.g., originating from bacteria, are frequently mistaken as viral reads leading to an increased number of false positives.

This motivated us to change the model to have three output classes: Cassava, Virus, or Bacteria. These three were chosen for two reasons: First, by analyzing the cassava microbiome, it was observed that these are the three most common kingdoms present in real-world samples. Second, adding other kingdoms like Archea significantly increased the size of the dataset, and it became intractable with the computational resources available and envisioned for a phytopathological clinic.

5.2.4 Model Architecture

CNNs were used in initial experiments similarly to other state-of-the-art models discussed in Chapter 3. However, this initial setup did not reach sufficient performance, although various model aspects were varied, such as the encoding strategy, type of reference data, and even the architecture and hyperparameters of the model. One of the reasons is that CNNs are good for capturing short-term dependencies, like the immediate neighborhood of a k-mer, but are not so great for revealing the relationships between k-mers at the beginning and the end of a sequence.

The poor performance of initial models motivated us to move from CNNs to recurrent models. The main idea was to enable the model to capture long-term dependencies. The initial attempts used LSTM models, but those models didn't perform well in training. Finally, attention-based NLP models were tested. These models performed

better compared to the previous models. Another advantage of using recurrent models and an NLP approach for read classification is that the same model can be used to classify different sequence lengths, which enables the model to work in the 100nt-300nt length range. Table 4 shows the number of paramters and the size of the model in Megabytes (MB).

Table 4 – The number of parameters and the size of the models.

Model	**Number of Parameters**	**Size (MB)**
3-Class Model	102,716,243	391.83
2-Class Model	83,849,758	319.86

5.2.5 Performance Evaluation

An important aspect of the proposed solution in this work is that the model employs protein data as reference. This choice led to a performance boost. However, the proposed model is unique in this aspect, which hampers a direct performance comparison with other state-of-the-art models like DeepVirFinder (REN et al., 2020). The main reason is that those models only work at the nucleotide level, and protein-level models must execute a pre-processing step before running the inference step because the employed reference data in the models aren't the same.

Nevertheless, it is known that DL models described in the literature do not perform well when classifying short reads. Figure 26 shows a chart taken from the DeepVirFinder paper (REN et al., 2020) that presents the value of the Area Under the Receiver Operating Characteristic Curve (AUROC) by contig length. The AUROC metric measures the ability of a classification model to distinguish between classes. We see that the larger the input length, the easier the DeepVirFinder model can distinguish the different classes. However, for input shorter than 300nt, the AUROC curve drops sharply and the classification performance of the model deteriorates.

Since it was not feasible to directly compare the performance of the developed deep learning model with that of other well-established models, we opted to verify its accuracy using traditional bioinformatics tools. An optimal model should accurately classify viral reads while avoiding the misclassification of reads originating from known non-viral sources. To evaluate this, we employed tools such as Kraken2 and Kaiju

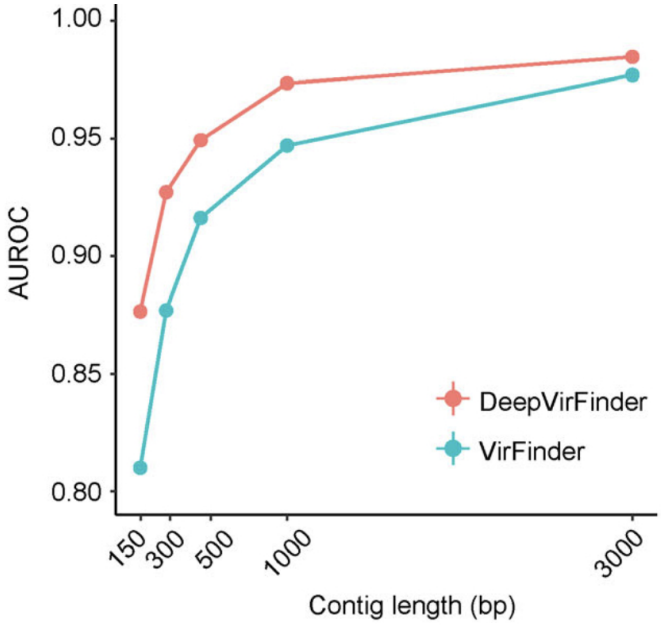


Figure 26 – AUROC metric for different nt-length of the DeepVirFinder Model (REN et al., 2020).

to analyze the reads that were labeled as viral by our deep learning model. These tools can help confirm whether the reads flagged as viral indeed belong to known viral sequences.

However, this approach has its limitations, as not all viruses are known, and the identification of novel viruses remains a complex task. To address this challenge, we used PhytoPipe (HU et al., 2023), a comprehensive phytosanitary pipeline that performs data cleaning, preprocessing, and the identification of novel viruses. By using PhytoPipe, we aimed to validate that the sequences classified as viral by the deep learning model correspond to sequences that PhytoPipe also identifies as viral. Additionally, this method allows us to determine if the deep learning model can detect novel viruses identified by PhytoPipe that were not included in the training data. Successfully classifying such sequences would demonstrate that the model can correctly identify viral reads from previously unknown viruses, showcasing its robustness and capability to generalize beyond the training set.

Figure 27 shows a flow chart with the steps executed for every modification of the proposed approach.

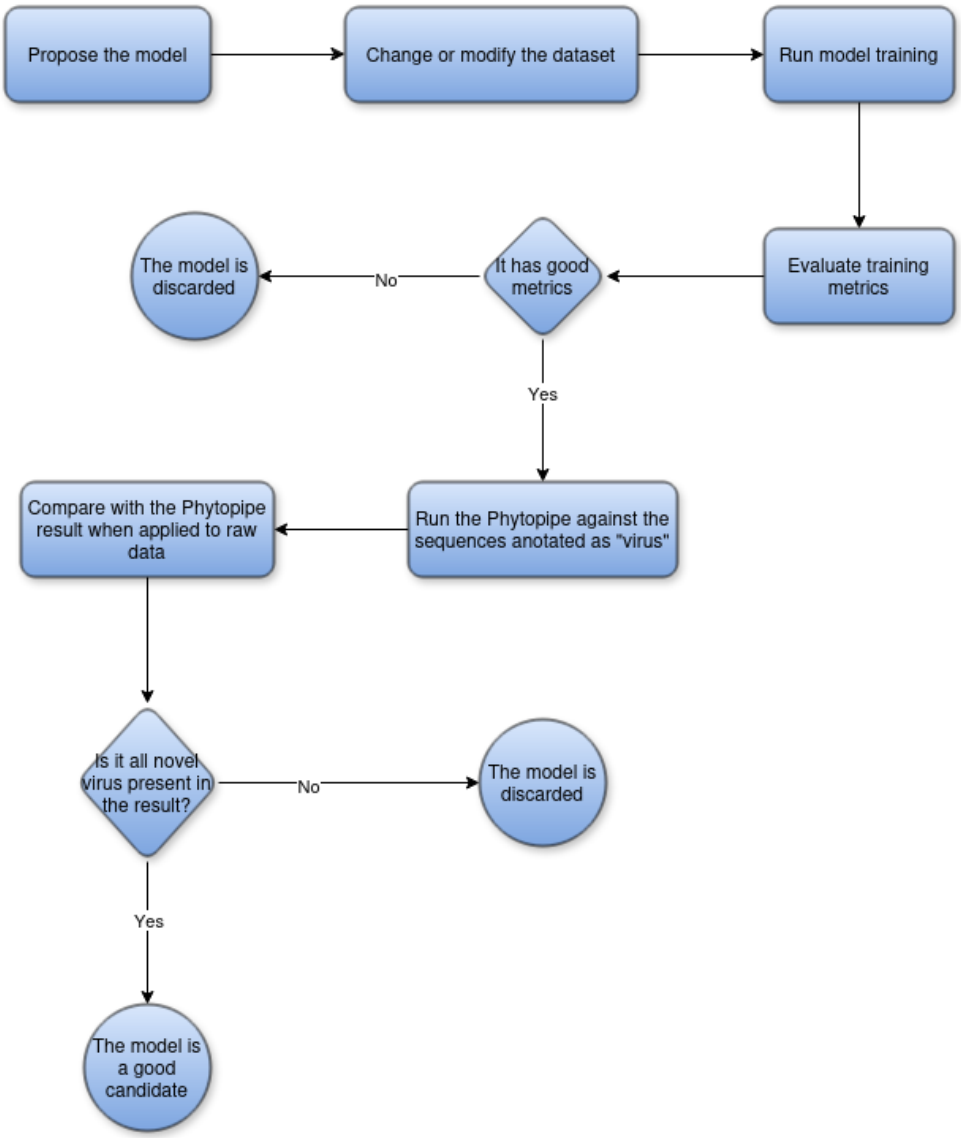


Figure 27 – Flowchart of all steps executed in experiments searching for a well performing model.

5.3 CASSAVA SAMPLES

Table 5 shows an overview of samples used in the present work to evaluate the performance of the model for the discovery of novel and known viruses. The data and sample descriptions were provided by the Phytovirology Laboratory of the Universidade Federal Rural de Pernambuco (UFRPE). Germplasm (seeds or plant parts used for breeding, conservation, and research) accessions from the collection of the Empresa Brasileira de Pesquisa Agropecuária (Embrapa) were maintained and propagated at the Plant Virus Department of the Leibniz Institute Deutsche Sammlung von Mikroorganismen und Zellkulturen (DSMZ). Plantlets were transferred to the soil and acclimated, and a growing-on test was conducted for a 6-8 months period in glasshouse

conditions. The libraries were sequenced on MiSeq (paired reads 2x301) or NextSeq (paired reads 2x151) platforms at the Leibniz Institute DSMZ.

Table 5 – Description of samples used in this work to evaluate the performance of the proposed model.

Code	Name	Platform	Sample Type	Reads
S16	DSCG1	MiSeq	Leaf	424,616
S23	DSCG12	MiSeq	Leaf	467,798
S27	P27-4C	NextSeq	Tuber	98,271,182

5.4 COMPUTATIONAL RESOURCES

For running the model's training step, the Apuana cluster from the Centro de Informática (CIn) at UFPE was used with the following specifications: 8 vCPUs, 64GB RAM, and 1 Nvidia A100 GPU. The PhytoPipe pipeline was executed on Amazon Web Services (AWS) Cloud, where we deployed a virtual server with 48 vCPUs, 384 GB of memory, and 3TB of storage.

5.4.1 Running Training Job on Apuana Cluster

To run training tasks on the Apuana cluster, it was first necessary to establish connection using the Secure Shell (SSH) protocol. Then, the dataset required to train the model was uploaded employing the Remote Synchronization (RSync) Protocol. Appendix C shows the script used to start a training job on the cluster.

6 RESULTS: PHYTOPIPE AS CLOUD SERVICE

The following chapter presents results obtained by executing a complete phytosanitary pipeline. The implementation of the pipeline as cloud service is an important contribution of the present work. The outlined techniques allow the access of state-of-the-art bioinformatics tool to facilities with limited hardware resources, either by direct execution in the cloud or by the generation of tailored reference databases. Section 6.1 presents an overview of the pipeline. Section 6.2 summarizes the main AWS services used to execute the pipeline. Section 6.3 explains how all necessary infrastructure resources were deployed and which modifications were needed to run the pipeline properly on AWS. Section 6.5 presents classification results of RNA-seq data obtained from Cassava plants.

6.1 PIPELINE OVERVIEW

PhytoPipe is an open-source pipeline for identifying plant pathogens using RNA sequence data. The pipeline can detect pathogens like bacteria, fungi, viruses, and viroids (HU et al., 2023). While developing the pipeline, the authors tested it on plant species like apple, pear, peach, potato, sweet potato, cassava, rice, sugarcane, and bamboo. Figure 28 shows all the steps executed by the pipeline. It can be divided into four main tasks: (i) preprocessing, (ii) classification, (iii) assembly-based annotation, and (iv) reference-based mapping. PhytoPipe supports both single- and paired-end FastQ files as input.

Preprocessing cleans the input data by removing host ribosomal RNAs, PCR duplicates, and low-quality reads.

The classification step uses Kraken2 to run a nucleotide-level classification by querying reads against the NCBI “nt” database. The pipeline also uses Kaiju to assign reads to taxa using the NCBI taxonomy and a microbial non-redundant database (nr+euk) of bacterial, viral, fungal, and other microbial eukaryotic proteins.

During the assembly-based annotation phase, the pipeline removes the host reads identified by Kraken2 and assembles the remaining reads with Trinity. The assemblies are then evaluated with Quality Assessment Tool for Genome Assemblies (QUAST)

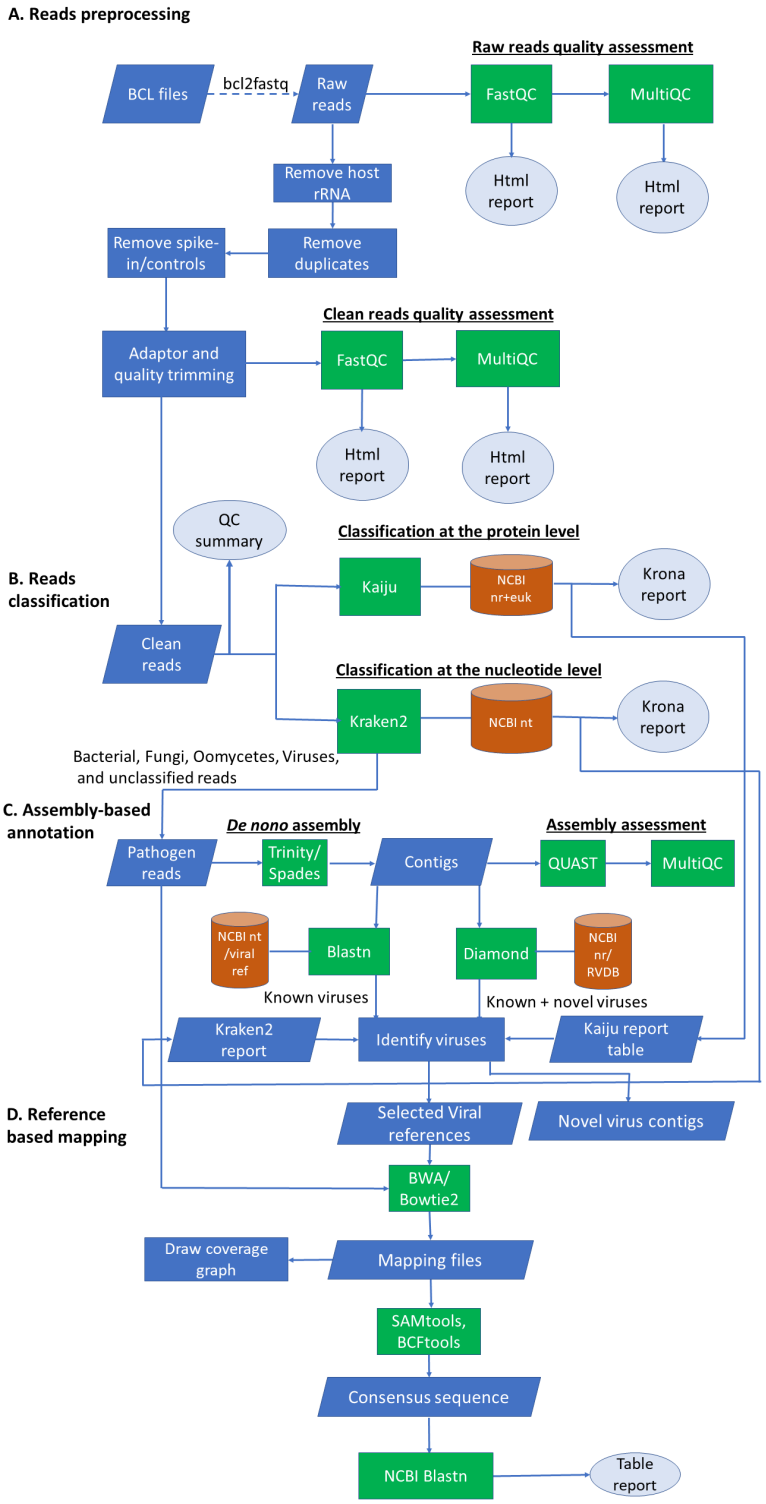


Figure 28 – The flowchart of all steps executed by PhytoPipe, a phytosanitary pipeline for plant pathogen discovery (HU et al., 2023). The following abbreviations for non-standard databases have been used: (i) nr+euk: NCBI non-redundant protein data base (nr) Bacteria, Archaea, Viruses, Fungi and microbial eukaryotes; (ii) RVDB: Reference Viral Database <https://rvdb.dbi.udel.edu>

and finally annotated at the nucleotide level against the NCBI “nt” database and at the protein level gainst the NCBI “nr” database using the alignment-based blastn and blastx tools, respectively.

At the end, the pipeline executes a reference-based mapping step, mapping the clean reads to the references identified by read classification and contig annotation, calculating read coverage, drawing a coverage graph, and generating consensus sequences. The consensus sequences receive their final annotation by an online query against the NCBI nucleotide BLAST database.

The PhytoPipe output for each sample provides several key reports and analyses. The generated results include FastQC/MultiQC reports for assessing the quality of the provided HTS reads. Krona taxonomy pie charts are generated from the Kraken2/Kaiju read classification and the blastn/blastx contigs identification. Additionally, the output features a QUAST report to evaluate assembly quality. Detailed blastn/Diamond blastx search result tables, mapping statistics, and coverage graphs for viruses and viroids are given. All of these findings are summarized in an Hypertext Markup Language (HTML) report (report.html). The authors set minimum system requirements to run the pipeline under Linux, Mac OS, or Windows (Docker only) operating systems that a very demanding: 300 GB of RAM, more than 32 cores CPU, and 1TB fast storage. However, as discussed later in this chapter, the actually system requirements are even higher, especially concerning the necessary storage space.

6.2 AMAZON WEB SERVICES OVERVIEW

Due to the high requirements for running PhytoPipe, we faced challenges in executing the pipeline locally. This motivated us to use Amazon Web Services (AWS) resources to run PhytoPipe at its maximum capacity. The AWS cloud has hundreds of services, and running any solution might be a challenge since a suboptimal configuration could increase costs dramatically. The PhytoPipe implementation fits the High-Performance Computing (HPC) use case, and we could use the basic computing services.

AWS’s most popular and basic computing service is the Elastic Compute Cloud (EC2), which offers a computing platform that allows us to choose processor, memory, and storage capacity. The storage capacity is offered as integral part of another AWS

service called Elastic Block Storage (EBS), which makes it possible to configure the size and performance of the store for a virtual service. The configuration of a virtual server can be saved by configuring an image. As a result, the EC2 can be easily created with the target configuration. In addition, it is possible to save generated data on storage units so it can persist between different virtual server runs. Those are very important aspects of virtual computing cost optimization enabling to stop running all resources while not being used.

Another important AWS service for our use case is the Amazon Simple Storage Service (S3), which provides object storage. As it isn't block storage, S3 can't be mounted on virtual servers, but the greatest advantage of this service is the cost. The objects represent the files stored in S3 and are saved into buckets - the name AWS gave to the containers for objects. As we need to run experiments on several High-Performance Computing (HPC) files, which generate a dozen output files, the best choice for storing the input and output files of PhytoPipe is the S3.

6.3 PIPELINE IMPLEMENTATION

Deploying the resources into the AWS cloud is the first step before running PhytoPipe. Analyzing the tutorial provided by the PhytoPipe developers, we observed that we need to run a database-building step before running the pipeline. Tools like Kraken2, Kaiju, and Blast need high-performance storage to build the databases that will execute the classification and annotation tasks. We chose to deploy all resources in two configurations, one for database building and the other for experiment execution. Figure 29 shows a diagram containing the services we used to build and run the PhytoPipe.

For the building phase of the pipeline, we deployed an r5a.12xlarge EC2 instance type with 48 CPUs, $384\,\mathrm{GB}$ of RAM, and $5\,\mathrm{TB}$ SSD storage. Once the virtual server was running, we could execute the updateDatabase.sh script provided on the PhytoPipe repository. We needed to apply two changes to run the script on AWS in a cost-effective way. The first change was related to the Kraken2 database. By default, this tool needs more than $700\,\mathrm{GB}$ of RAM to query the sequenced reads against the NCBI nt database. However, the size can be reduced by setting a parameter called maximum database size at build time. A reduction to $256\,\mathrm{GB}$ does not significantly decrease the accuracy of the Kraken2 classification. The second change was related to

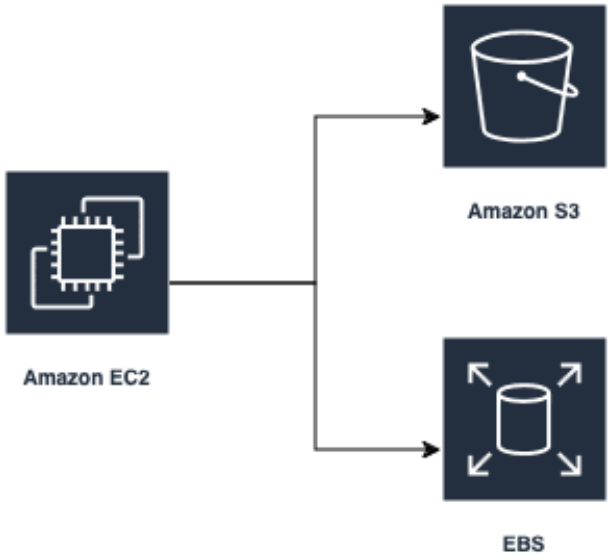
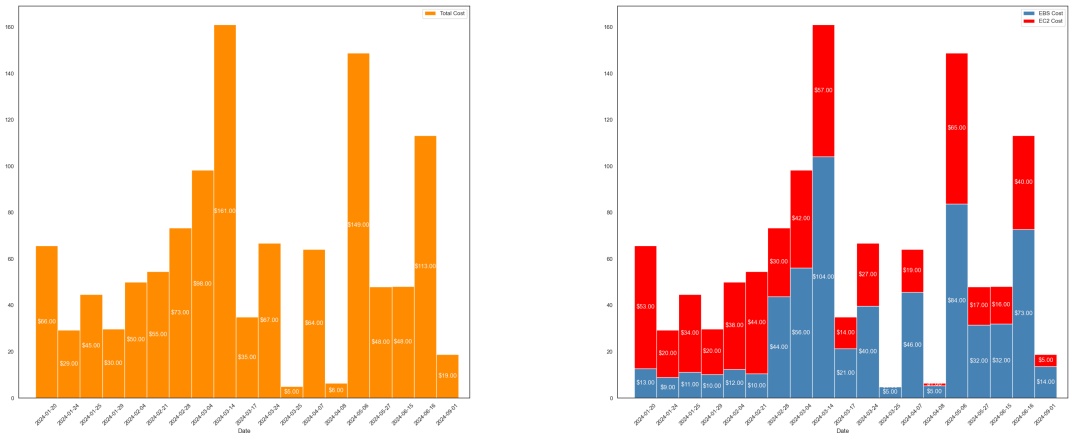


Figure 29 – Resources deployed on the AWS cloud to run PhytoPipe.

building the Kaiju database. Kaiju provides pre-built databases that can be retrieved from a file server. The download eliminates the need to build the Kaiju database, which reduces the costs of executing the PhytoPipe assembly script by reducing the execution time. After applying these changes, the build step was initiated and took five days to conclude due to the high data volume processed by tools like Kraken2 and Blast. Once the pipeline was successfully assembled, we created a snapshot of EBS storage to restore it when running the pipeline to analyze a new HTS sample.

To execute the pipeline for analyzing a FastQ input file, an r5a.12xlarge EC2 instance type with 48 CPUs, 384 GB of RAM, and 5 TB magnetic storage was deployed. The change from SSD to magnetic storage significantly decreased the cost for every execution without dramatically affecting the performance. To run the pipeline, we first upload the FastQ files into an S3 bucket, create the EC2 and EBS resources, and start PhytoPipe. When the pipeline execution was finished, we uploaded the results into the S3 bucket and deleted the EC2 and EBS resources to avoid unnecessary costs. To make the creation and deletion of resources easy, we used Terraform, an Infrastructure as a Code (IaaC) tool used to manage resources on the cloud programmatically. Appendix B shows the script we developed using Terraform to manage the resources we used on AWS to run Phytopipe. Table 6 shows the major experiment files that were processed using the AWS cloud infrastructure.

6.4 AWS COSTS



(a) Total Cost for AWS execution of the pipeline for experiments executed on specific days.

(b) Cost per Service for AWS execution of the pipeline for experiments executed on specific days.

Figure 30

We executed an analysis on the costs generated by the execution of the pipeline on the AWS Cloud. Analyzing the total cost it was possible to notice that the average cost of the execution for each experiment processed by the Phytopipe was USD6 1.00, but the total cost for some executions reached USD 100.00. Also, we analyzed the cost by the AWS services used during the processing. Basically, during the processing it was used the EBS (storage) and the EC2 (computing) services. For each experiment, the average cost for the EBS service was USD 32.70 and for the EC2 service was USD 28.90, this shows that the our configuration had a higher cost for the storage of the virtual machine when compared with the cost of the virtual machine itself. This was due to the high volume of data generated by the datasets for the Blast, Kraken2, and Kaiju, and the high amount of input and output operation on the disk generated by the same tools.

Experiment File	Preprocessed by (or raw file)	Date	AWS cost for this date
DSCG1_S1 R1 & R2	raw	24/01/2023	$ 24.00
DSCG1_S1 R1	kraken2_virus	24/03/2024	$ 67.00
DSCG1_S1 R2	kraken2_virus		
DSCG1_S1 R1	2-class model	04/03/2024	$ 98.00
DSCG1_S1 R2	2-class model	17/03/2024	$ 35.00
DSCG1_S1 R1	3-class model	28/02/2024	$ 73.00
DSCG1_S1 R2	3-class model	14/03/2024	$ 161.00
DSCG12_S12 R1 & R2	raw	04/02/2024	$ 50.00
DSCG12_S12 R1	kraken2_virus	24/03/2024	$ 67.00
DSCG12_S12 R2	kraken2_virus		
DSCG12_S12 R1	2-class model	04/03/2024	$ 98.00
DSCG12_S12 R2	2-class model	24/03/2024	$ 67.00
DSCG12_S12 R1	3-class model	28/02/2024	$ 73.00
DSCG12_S12 R2	3-class model	14/03/2024	$ 161.00
P27-4C R1 & R2	raw	06/05/2024	$ 149.00
P27-4C R1	kraken2_virus	01/09/2024	$ 19.00
P27-4C R2	kraken2_virus		
P27-4C R1	2-class model	15/06/2024	$ 48.00
P27-4C R2	2-class model	16/06/2024	$ 113.00
P27-4C R1	3-class model	27/05/2024	$ 48.00
P27-4C R2	3-class model		

Table 6 – Execution of experiments on the AWS cloud splited by the experiment file and date. We also added the total cost of AWS cloud for the date, but it does not represent the cost of the experiment running itself once that multiple runs was executed on a single day.

6.5 VIRUS DETECTION

We analyzed three cassava sample files executing PhytoPipe, each containing different scenarios of viral infections. Those sample files were provided by the Phytovirology Laboratory from UFRPE and previously analyzed by bioinformatics specialists. The findings of PhytoPipe confirmed the initial assessments. Frame 3 summarizes the results of the Phytopipe execution for each file. The S16 sample doesn't contain nucleotide reads originating from any known or unknown virus. The S23 sample evidenced viral infection of the cassava plant by the known viruses Cassava common mosaic virus and Cassava Torrado-like virus. Finally, in the S27 sample, the following

viruses were detected: Cassava Torrado-like virus, Cassava ampelovirus 1, Cassava torrado-like virus 2 and Cassava satellite virus. The table indicates also that some consensus sequences for the latter two have lower blast identity and, thus, may represent novel variants.

Sample Name (Code)	**Viruses Detected**	**Novel Viruses / Isolates**
DSCG1 (S16)	—	—
DSCG12 (S23)	Cassava common mosaic virus Cassava Torrado-like virus	—
P27-4C (S27)	Cassava torrado-like virus 2 Cassava ampelovirus 1	Cassava torrado-like virus 2 Cassava satellite virus

Frame 3 – Results of virus detection by PhytoPipe for three cassava RNA samples.

Figure 31 gives an example of an HTML report generated by Phytopipe summarizing the analysis of the sample file S27. The shown part of the report focus on the mapping of reads to novel contigs. Phytopipe employs two criteria to identify a novel virus, a low identity at the nucleotide level but a high match at the protein level. In the standard configuration a novel contig has a blastn identity smaller than 80% to the closest viral reference but an expectation value (e-value) lower than 1×10^{-100} when aligned against the closest viral protein, indicating a highly significant match. Figure 31 shows that some contigs close to the reference virus Cassava torrado-like virus 2 or Cassava satellite virus might be novel.

The output report shows several other pieces of information, such as the identified known viruses, the amount of raw reads, the amount of reads after cleaning, etc. It's possible to access the Krona charts generated by the executions of Kraken2 and Kaiju directly from the file report.html.

The Krona pie charts for the sequenced reads classified by Kraken2 for the three samples S16, S23 and S27 are shown in Figure 32 (a), (b) and (c), respectively. The virus and bacteria classes identified for the sample S27 are separately displayed in Figure 32(d) and (e), respectively, for better visibility. The provision of partial results obtained during the execution of the phytosanitary pipeline helps to deepen the understanding of the classification flow. The interactive pie charts generated by Phytopipe allow for better comprehension and visualization of the sample's microbiome.

Figure 33 shows the Krona pie charts for sequenced reads that Kaiju classified for the same three samples (S16, S23, and S27). A comparison of the Kraken2 and

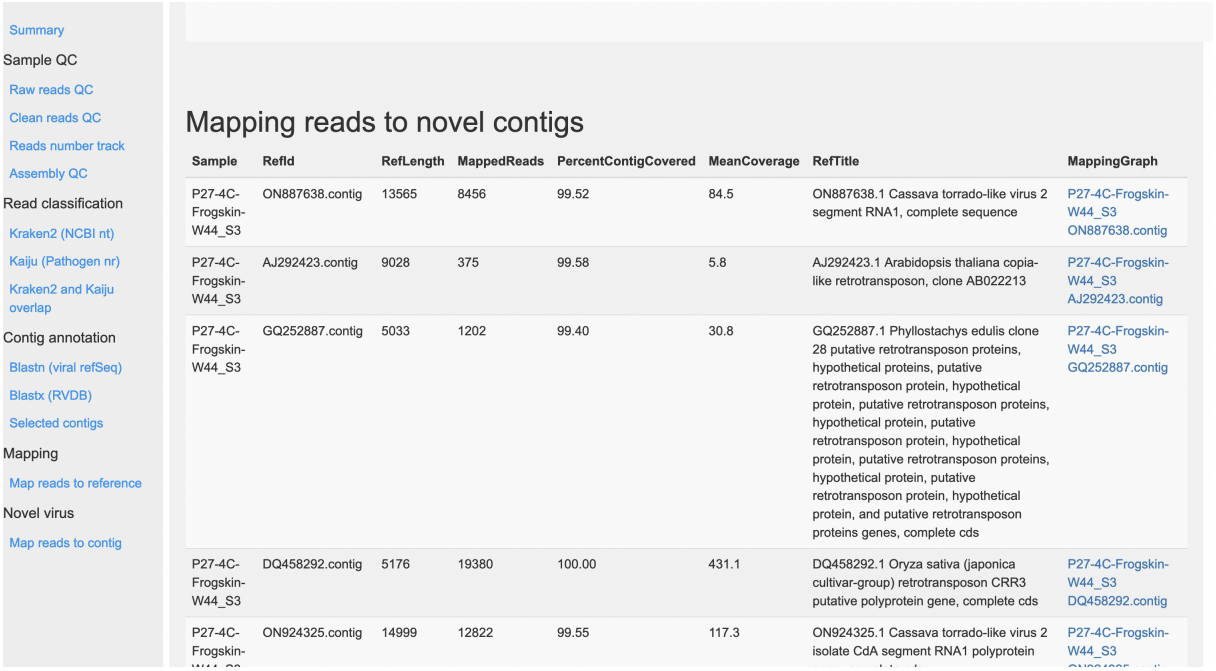


Figure 31 – An example of an HTML report generated by PhytoPipe for the S27 sample. This image shows identified contigs with a low blastn identity to viral references but an excellent match to viral proteins, singling them out as potentially novel as indicated by the header.

Kaiju results is interesting since we gain insights about the differences of nucleotide- versus protein-level classification, although the Kaiju reference does not include plant proteins. Thus, an obvious difference is that Kaiju does not identify reads from the plant host. Additionally, Kaiju does not show unclassified or unknown reads as “No Hits” and “Other Root” in the charts.

The results obtained for the disease free sample S16 are shown in Figure 33a. Kaiju identified mostly bacteria reads, while Kraken2 identified mostly Cassava reads. For the S23 sample shown in Figure 33b, Kaiju also identified mostly bacteria reads and the Cassava common mosaic virus. Kraken2 does not identify bacteria on this sample but identified viral pathogens similar to Kaiju. Finally, for the S27 sample, Figure 33c demonstrate that Kaiju identified more bacterial and viral taxons than Kraken2. However, both tools identified the Cassava-torrado like virus 2 and the Cassava ampelovirus 1 on this sample. Other viral reads identified by Kaiju do nor represent plant infecting viruses. As a general trend, we can state that Kaiju gives a more detailed view of the plant’s microbiome. However, since the identification of plant proteins is not included, the tool cannot serve as a filter for passing only pathogen reads (defined as not host) to the assembly unit in a phytosanitary pipeline.

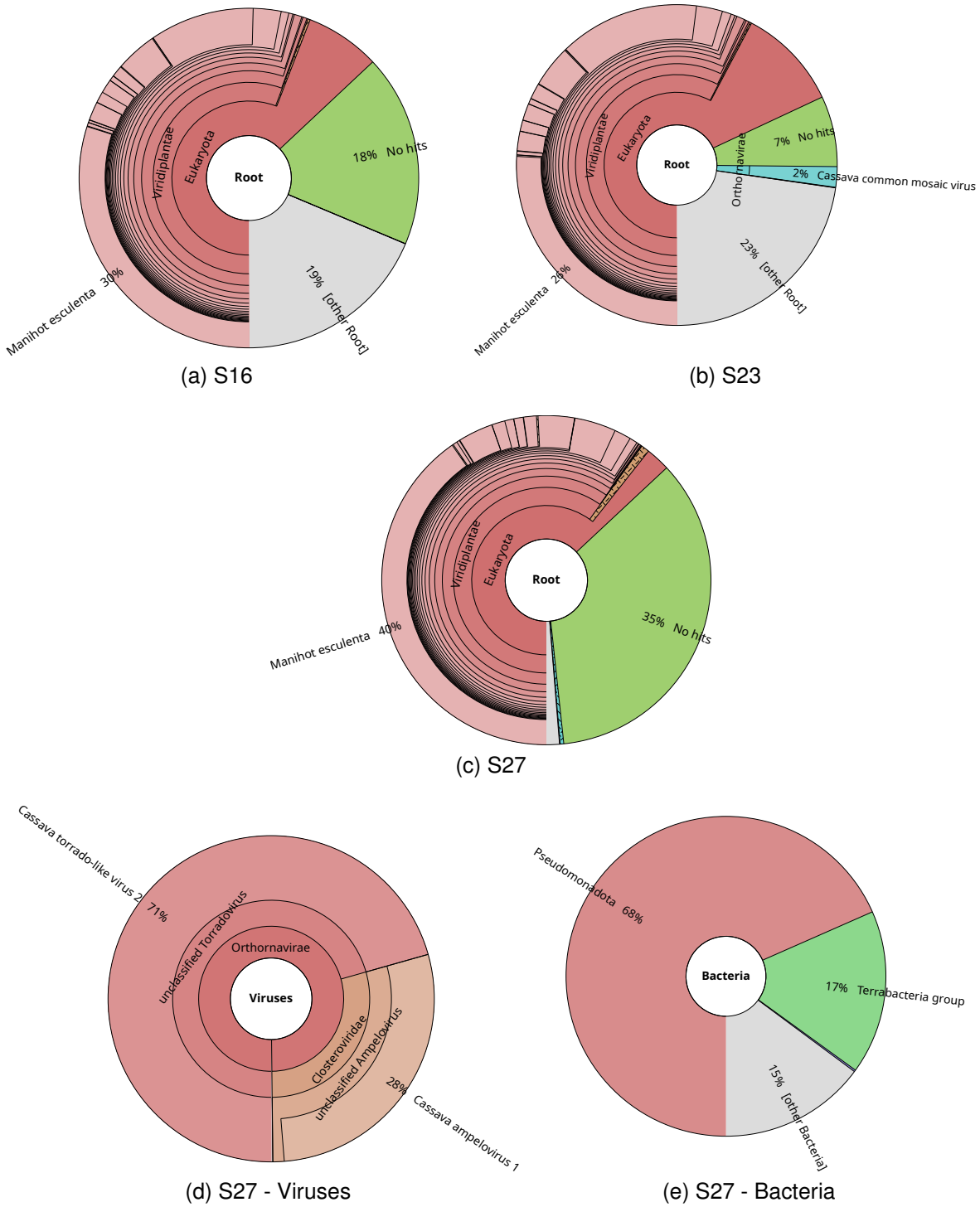


Figure 32 – Krona pie charts displaying the Kraken2 classification of the sequenced reads obtained during the Phytopipe analysis of three cassava samples, S16 (a), S23 (b), and S27 (c). The subfigures (d) and (e) represent the drill-down view of the viral and bacteria classes. Because of the high number of non-pathogen reads, these classes are not visible in (c).

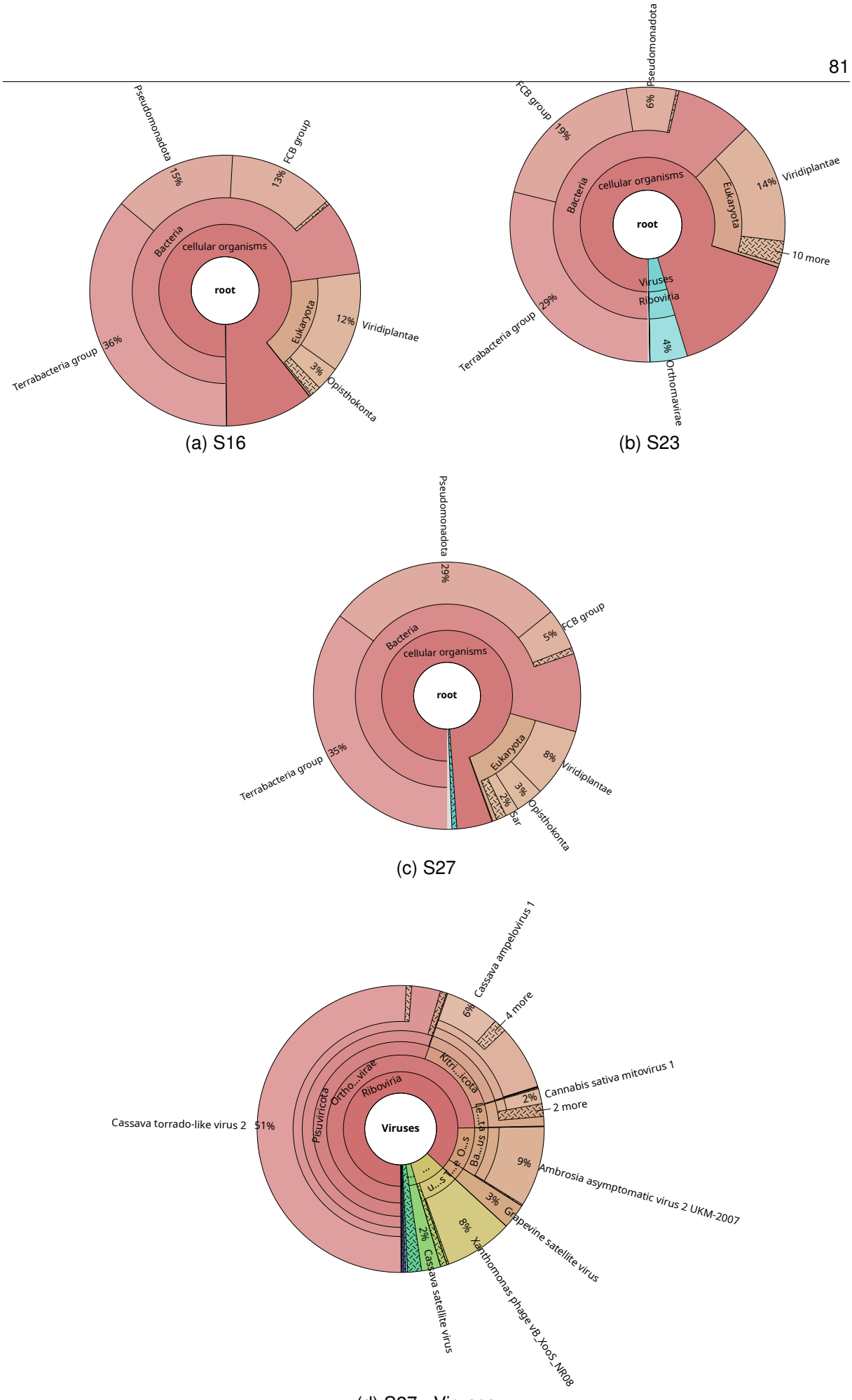


Figure 33 – Krona pie charts displaying the Kaiju classification of the sequenced reads obtained during the Phytopipe analysis of three cassava samples, S16 (a), S23 (b), and S27 (c). The subfigure (d) represents the drill-down view of the viral classes.

6.6 PRE-BUILT DATABASES

Another result achieved in this work was to facilitate the PhytoPipe implementation for other potential users by publishing pre-built database files for Kraken2 and Blast. This was motivated by the challenges faced when trying to build those databases. Much more computational capacity is needed to build the reference databases than to run the pipeline. Additionally, a stable internet connection over a longer period of time is required to download large or multiple files from the NCBI server.

Figure 38 in the appendix D shows the pre-built files for the NCBI non-redundant nucleotide Kraken2 reference database. By default, this database requires more than 700GB of memory, but Kraken2 offers a parameter to limit the maximum size of the database through downsampling of minimizers (for both the database and query sequences) using a hash function (WOOD; LU; LANGMEAD, 2019). In this work, the maximum size of the database was set to 256GB, the size recommended by Phytopipe's authors. However, we created also a 120GB database to enable users with less computational resources to run Phytopipe.

Figure 39 in the appendix D shows the pre-built files for the NCBI nucleotide BLAST database. The challenge faced when users try to download these files from the original source - the NCBI servers - is that a stable connection is required allowing the download of thousands of files. Sometimes, the connection with NCBI servers faces instability, so the download fails, forcing the users to retry this process several times or even preventing the user from downloading them. As we used AWS servers, setting a stable and fast connection with NCBI servers was possible. Finally, by using the Amazon Simple Storage Service (S3), other users can download this database from a more stable source, as Amazon offers a global connection infrastructure to access its services.

Researchers from UFRPE and the Leibniz Institute DSMZ successfully used the pre-built files described in this section to run Phytopipe.

6.7 KRAKEN2 WITH VIRUS-ONLY DATABASE

Another investigation we conducted evaluated the performance of Kraken2 using the NCBI virus database as a reference, which is a virus-only database. As mentioned

in Section 2.6, Kraken2 allows the construction of different databases with various configurations. For the results presented in this section, we used:

```
kraken2-build –download-library viral –db kraken2_viral
```

For optimal sequence classification, using the complete NCBI nt reference database is recommended due to its comprehensive nature, but this comes with high computational demands. In contrast, using a virus-only database reduces computational requirements, making it feasible to run Kraken2 on most personal computers. This section examines whether the accuracy of viral sequence classification is significantly affected by using a reduced reference database.

We classified the raw reads of three cassava samples (S16, S23, and S27) using Kraken2 with the NCBI virus reference, an approach we refer to as viral Kraken2. We then reclassified the reads initially annotated as viral using the complete NCBI nt database. Figure 34 shows the Krona charts generated after this second classification.

In the S16 sample (Figure 34a), most reads initially categorized as viral by the viral Kraken2 approach were actually identified as cassava reads when using the complete nt database.

Figure 34b shows that in the S23 sample, most viral reads were correctly identified as originating from cassava-infecting viruses. Specifically, Cassava common mosaic virus accounted for 66% of the viral reads, and Cassava Torrado-like virus for 1%. However, 33% of reads initially classified as viral were misclassified, with the complete database revealing them as belonging to the categories "Eukaryota," "Other Root" (It can’t determine the taxonomy), or resulting in "No hits" (It can’t find similar reference).

For the S27 sample, Figure 34c indicates that 84% of the initial virus class reads originated from the plant host. A closer look at the reads classified as viral when queried against the complete NCBI nt database (Figure 34d) showed that only a few reads from Cassava Torrado-like virus and Cassava Torrado-like virus 2 were included in the original virus classification.

Table 7 provides the total number of viral reads classified by the viral Kraken2 approach and reclassified using the complete NCBI nt database. In samples S16 and S27, viral Kraken2 showed poor performance, with most reads initially annotated as viral actually originating from the cassava genome. This indicates a significant overestimation of viral diversity in these samples by the viral Kraken2 approach. Additionally, the method was insufficient in identifying relevant virus species, as seen with the low

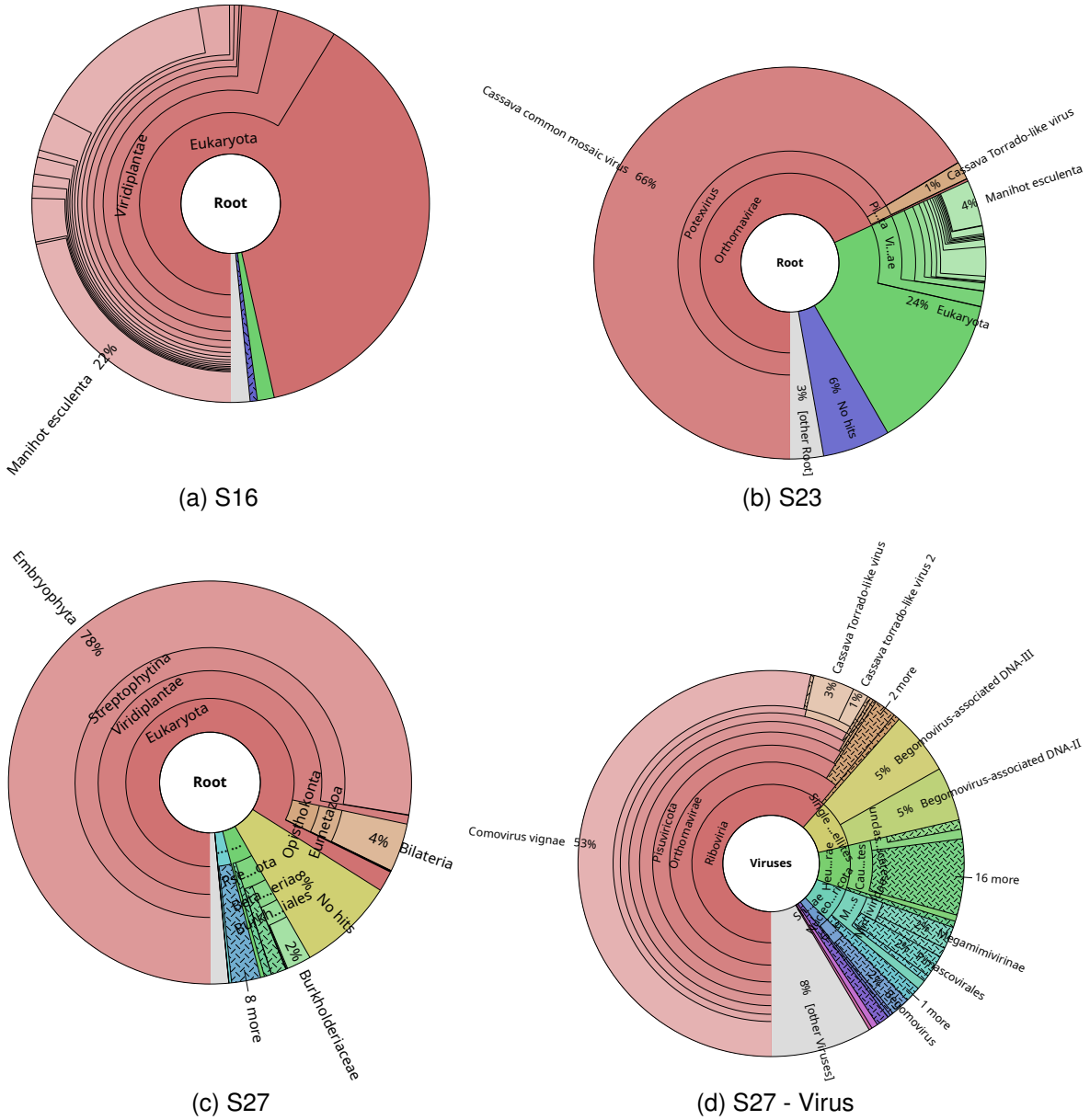


Figure 34 – Krona pie charts displaying the reclassification of reads annotated as viruses when queried against the NCBI virus database using Kraken2. These charts were generated by reclassifying the initial virus class by querying against the complete NCBI nt database using Kraken2. Results are shown for the three samples: S16 (a), S23 (b), and S27 (c). For sample S27, (d) provides a detailed view of the reads reclassified as viral.

Sample	#Total Reads	#Cassava Reads	#Viral Reads / Species
S16	511	493 (96%)	2 / Cassava Satellite virus
			1 / Cassava brown streak virus
S23	4494	1058 (24%)	2985 / Cassava common mosaic virus
S27	14844	12465 (84%)	13 / Cassava Torrado-like virus
			2 / Cassava Torrado-like virus 2

Table 7 – Number of reads for three samples originating from the cassava genome and plant viruses as classified and reclassified by Kraken2 using a viral-only and the complete NCBI nt database in the two annotation steps, respectively.

detection rates for Cassava Torrado-like viruses in sample S27 (13 reads for Cassava Torrado-like virus and 5 for Cassava Torrado-like virus 2 out of 98 million total reads).

In conclusion, reliable Kraken2 classification of HTS reads requires including the host genome in the reference data. Part of the cassava genome appears to resemble viral sequences, at least for fragments of the size of typical HTS reads. We confirmed this by generating over six million artificial reads (100 nt each) from the *Manihot esculenta* reference genome, finding that 0.2% were classified as viral by viral Kraken2. Furthermore, to enable Kraken2 to accurately score classification paths, diverse reference data covering different taxa are necessary. Therefore, in the context of this study, a reliable benchmark of deep learning models demands significant computational resources. While using a virus-only database offers low computational costs, it cannot replace the comprehensive analysis provided by the NCBI nt database.

7 RESULTS: ATTENTION-BASED SEQUENCE CLASSIFIER

The following chapter presents the results obtained for the deep-learning model proposed to classify plant RNA-seq data, identifying viral reads at the protein level. Section 7.1 shows the model's architecture and how the model was trained. Section 7.2 discusses the results obtained when running the model on real-world data and evaluates the performance by comparison with a PhytoPipe analysis.

7.1 MODEL ARCHITECTURE AND TRAINING

Figure 35 shows the architecture of the proposed model. The first step is the nucleotide-to-protein translation. This translation step generates six aa sequences for every nucleotide read. Some combinations of three nucleotides represent a stop codon not coding for an amino acid. If any aa fragment contains the corresponding symbol, it is removed from the input group. The presence of a stop codon signifies the end of a valid protein-coding region. Any amino acids that appear after a stop codon during in silico translation are generally non-biological artifacts and should be ignored. Thus, we decided to exclude the corresponding fragments completely since they may contain only limited information.

The following classification steps are executed for every aa fragment. If the model classifies any protein read as originating from a pathogen, then the original nucleotide-level read is labeled correspondingly. If an aa fragemt has a finite probability to belong to both pathogen classes, virus and bacteria, the class with the higher probability is used for the labeling. After translation to the protein level, we employ k-mer encoding with $k = 5$. The consecutive embedding represents the k-mers in a dense vector space and adds positional information to the transformation into numerical data. The embedding is refined by the attention mechanism employing four parallel heads. Finally, a fully-connected layer computes the output probabilities for every class.

We trained two different models and employed them in inference experiments. The first model has two output classes: cassava and virus. The second model has three output classes: cassava, bacteria, and virus. The models were trained with over 39 million artificial reads for five epochs. For the two-class-model, the loss value varied

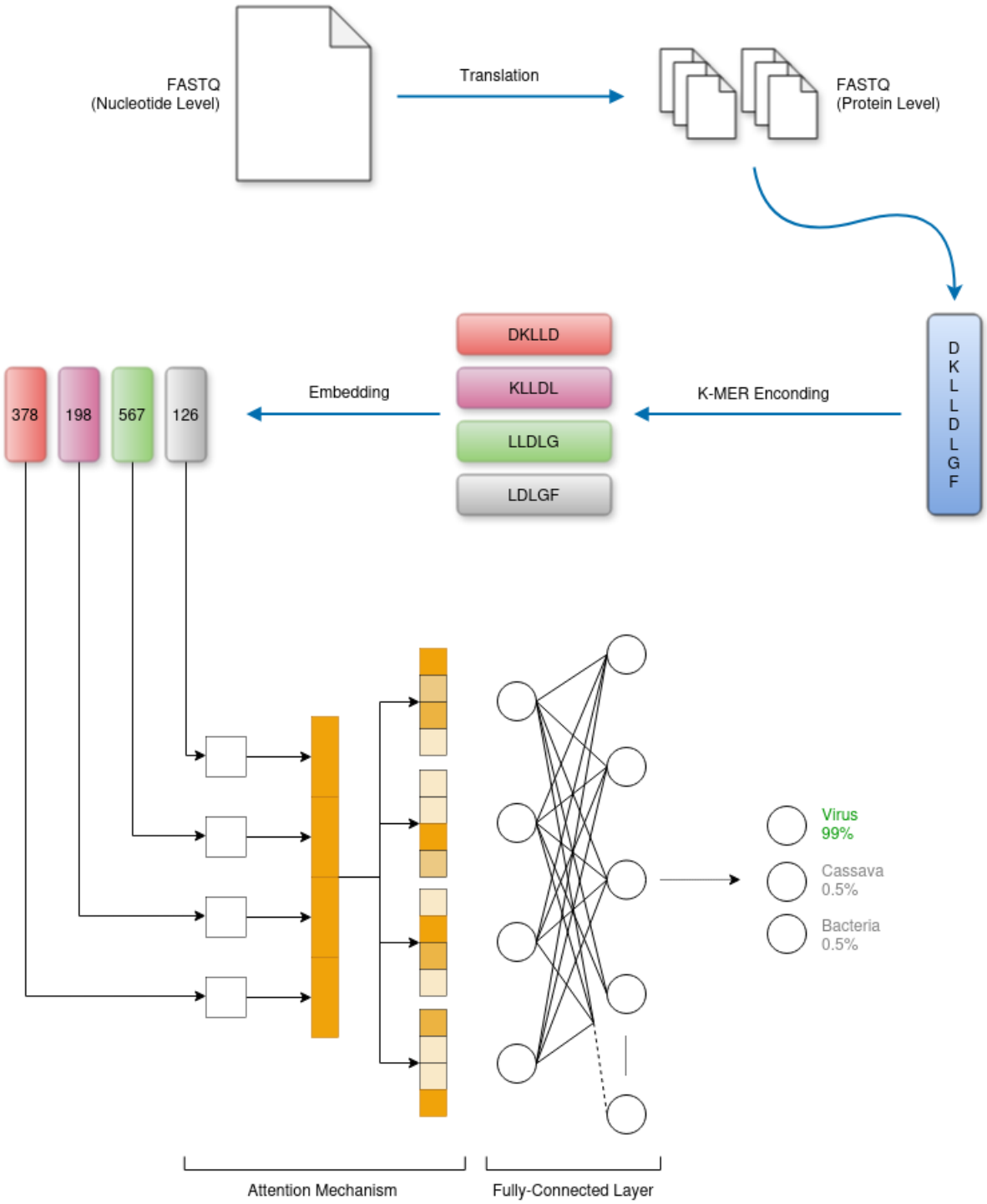


Figure 35 – The architecture of the proposed DL classifier on protein level.

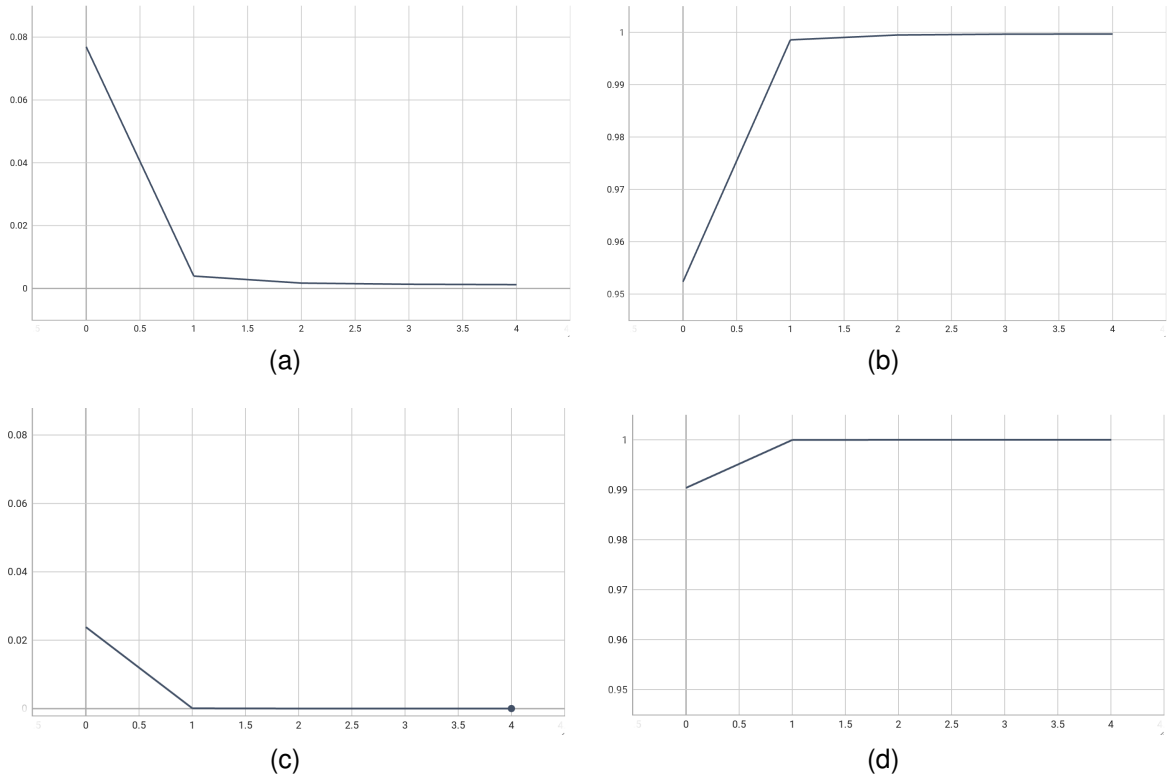


Figure 36 – Values for loss and accuracy during training. The upper row show the loss (a) and the accuracy curves (b) for the three-output-class model. The lower row displays the same quantities, loss (c) and accuracy (d), for the two-output-class model.

from 2.24×10^{-2} to 1.17×10^{-5} and the accuracy from 0.9904 to approximately 1 through the training epochs as shown in Figure 36c and 36d, respectively. Figure 36a displays the loss and 36b the accuracy curve during training the three-output-class model. Loss improved from 7.69×10^{-2} to 1.21×10^{-3} and accuracy from 0.9523 to 0.9997.

The actual performance of the models, when classifying real-world data, will be evaluated in the next section.

7.2 PERFORMANCE EVALUATION

We applied the trained models to classify sequenced reads obtained from three cassava samples to evaluate the performance in each case. This experiment aimed to verify whether the proposed models could be used as a trustful classifier. To this extent, the reads classified as viral by each model were further analyzed by well established bioinformatics tools. As mentioned in Subsection 5.2.5 and shown in Figure 27 we executed the developed models on the raw sample reads. Then, we analyzed the model predictions by the traditional tools executed as part of PhytoPipe.

A stage of high computational cost in a phytosanitary pipeline is the filtering of sequenced reads that will be passed to the assembler unit. Only pathogen or unclassified reads shall be passed. Since most reads originate from the host, see Figure 24, their elimination from the downstream data analysis of the pipeline decreases hardware demands, reduces execution time and improves contig and consensus sequence quality. Thus, a high number of identified viral reads without contamination by sequences known to originate from other kingdoms (false positives) is an interesting performance metric for the intended use of the DL model.

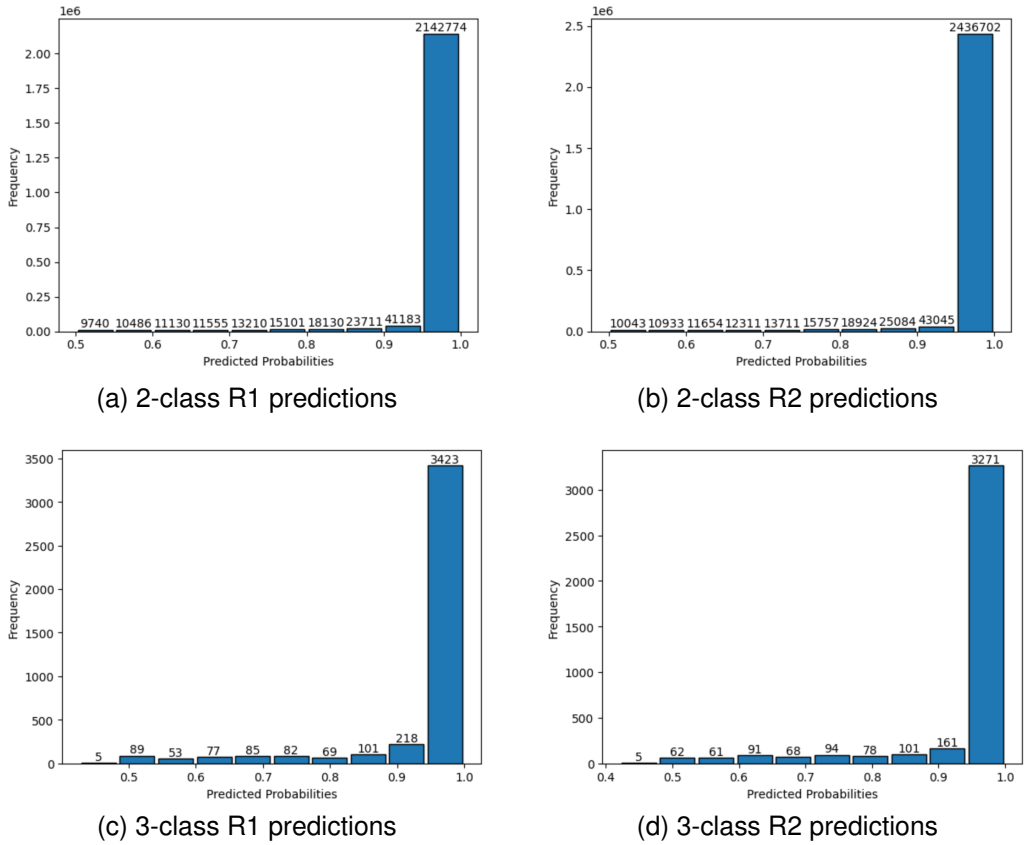
We discuss the model performance first by comparing with the findings in Section 6.5. Second, by channeling the reads of the virus output class into PhytoPipe, the pipeline reveals if the DL model kept enough viral information that traditional tools could confirm. In addition, we aimed to verify if the model has improved its performance by adding an output class. Thus, we report on the findings for both models.

7.2.1 General Trends

For each model, we ran inference for three cassava samples and grouped all reads with a class probability higher than 98%. We chose such a stringent criteria because the main goal is to identify reads matching viral proteins with high confidence. This performance metric is well suited for the intended use as a filter: keep a small number of reads that are most likely of viral origin. Table 8 summarizes the obtained number of viral, cassava, and bacteria reads as predicted by the two models.

Sample	Total Reads	2 Class Model >98% probability		3 Class Model >98% probability		
		Virus	Cassava	Virus	Cassava	Bacteria
S16 (R1)	72,603	5,090	31,841	249	16,574	18,324
S16 (R2)	76,722	6,191	33,556	401	14,737	22,278
S23 (R1)	136,919	13,999	64,707	2,578	24,225	47,948
S23 (R2)	147,139	15,804	68,232	2,591	20,690	55,734
S27 (R1)	47,958,919	1,789,807	12,303,556	2,542	3,294	14,642,441
S27 (R2)	47,958,919	2,099,988	12,149,422	2,421	3,169	14,793,205

Table 8 – Deep learning classification of sequenced RNA reads obtained from three cassava samples. Only reads with a predicted probability higher than 98% are grouped into the corresponding output class. Other reads are considered as unclassified.



(a) 2-class R1 predictions

(b) 2-class R2 predictions

(c) 3-class R1 predictions

(d) 3-class R2 predictions

Figure 37 – Histogram of S27 derived reads grouped into probability bins for being of viral origin. The upper row shows the classification results of the two-class model for R1 (a) and R2 (b) reads, the lower row results of the three-class-model for R1 (c) and R2 (d) reads.

The S16 sample has the lowest number of reads. Analyzing Figure 32a, we expect that this sample has no virus pathogens. Thus, ideally, no nucleotide read should be labeled as viral. In this respect, the 3-class model achieves better results than the 2-class model. Interesting are the findings concerning the bacteria class. Based on the analysis shown in Figure 32a, we also do not expect to find bacterial reads contrary to the classification provided by the 3-class model. However, the nucleotide level analysis provided by Kraken2 does not reveal the whole microbiome. In fact, the Kaiju analysis on protein level also finds a considerable amount of bacterial reads very similar to the 3-class model. We hypothesize that Kraken2 may sort most bacterial reads into the category “Unclassified” due to the huge variability of bacterial genomes. However, the translated proteins are less varying and classification on protein level might be a better strategy for bacteria discovery.

The S23 sample has a small but relevant number of viral reads as shown in Figure 32b. Both, the Kraken2 and the Kaiju analysis labeled ca 3000 viral reads. The number of viral reads identified by the 2-class model is much larger. The 3-class model on the

other hand grouped a similar amount of reads into the virus class.

The viral and bacterial load is very high in sample S27, as shown in Figure 32d and 32e, respectively. For this sample, the 3-class model assigned a relatively low number of reads as viral. The sample S27 is the only one from the investigated exemplary cassava samples that may include novel viruses or variants. Thus, the finding raises the concern that the 3-class model is too conservative when labeling viral reads. The impact of the threshold probability on the model performance needs to be investigated in more detail.

As a first step, we analyzed the number of viral reads grouped into different prediction probability bins. Figure 37 shows the resulting histograms for the S27 sample for both R1 and R2 reads and for both models. We see that most reads predicted as viral have class probabilities in the 0.95 to 1.0 range in both models. Thus, a careful fine-tuning of the threshold probability might be required.

To summarize, the expected general trends in the classification of the sequenced reads obtained from three exemplary cassava samples are well reproduced by the 3-class model. The 2-class model performed significantly poorer and the inclusion of the bacterial class is required. When employed as a filter in the context of a phytosanitary pipeline, the 3-class model is capable of removing most reads keeping only some with a high probability of being a pathogen read.

7.2.2 PhytoPipe Analysis

A more detailed performance assessment is obtained when the reads assigned by the DL model to each class is channeled through PhytoPipe. This analysis examined whether the DL classification results could be confirmed by traditional and well-established bioinformatics tools. Table 9 summarizes the Kraken2 results as part of PhytoPipe for all reads previously processed by the deep learning model. As expected, the 3-class model outperformed the 2-class model for predicting all three classes, virus, bacteria, and cassava.

Sample	Virus Class		Cassava Class		Bacteria Class
	2 Class Model	**3 Class Model**	**2 Class Model**	**3 Classes Model**	**3 Class Model**
S16 (R1)	48% Eukaryota (Host) 36% No Hits 16% Others	46% Others 39% Eukaryota (Host) 15% No Hits	88% Eukaryota (Host) 7% No Hits 5% Others	99% Eukaryota (Host) 1% No Hits	68% Eukaryota (Host) 19% No Hits 12% Others 1% Bacteria
S16 (R2)	48% Eukaryota (Host) 36% No Hits 16% Others	48% Eukaryota (Host) 35% No Hits 17% Others	86% Eukaryota (Host) 7% No Hits 7% Others	99% Eukaryota (Host) 1% No Hits	67% Eukaryota (Host) 19% No Hits 13% Others 1% Bacteria
S23 (R1)	44% Eukaryota (Host) 24% No Hits 16% Virus 16% No Hits	70% Virus 16% No Hits 8% Eukaryota (Host) 5% Others	91% Eukaryota (Host) 7% Others 2% No Hits	100% Eukaryota (Host)	77% Eukaryota (Host) 13% Others 9% No Hits 1% Virus
S23 (R2)	43% Eukaryota (Host) 25% No Hits 17% Others 15% Virus	66% Virus 16% No Hits 12% Eukaryota (Host) 6% Others	90% Eukaryota (Host) 8% Others 2% No Hits	100% Eukaryota (Host)	76% Eukaryota (Host) 13% Others 9% No Hits 1% Virus
S27 (R1)	62% No Hits 36% Eukaryota (Host) 1% Bacteria 0.5% Others 0.5% Virus	56% Virus 36% No Hits 8% Eukaryota (Host)	74% Eukaryota (Host) 24% No Hits 1% Others 1% Bacteria	96% Eukaryota (Host) 4% No Hits	69% Eukaryota (Host) 29% No Hits 1% Bacteria 1% Others
S27 (R2)	63% No Hits 34% Eykaryota (Host) 1% Others 1.5% Bacteria 0.5% Virus	56% Virus 35% No Hits 9% Eukaryota (Host)	74% Eukaryota (Host) 24% No Hits 1% Others 1% Bacteria	96% Eukaryota (Host) 4% No Hits	68% Eukaryota (Host) 30% No Hits 1% Bacteria 1% Others

Table 9 – Probable origin of sequenced reads of three cassava samples as grouped by the DL models in the classes virus, cassava ans bacteria. To predict the likely origin, PhytoPipe was executed for all reads, and the Kraken2 classification was recorded.

To fully appreciate the findings, we need to comment on the two Kraken2 classifications called “No Hits” and "Other Root". Both are categories used to describe reads that do not fit neatly into well-defined taxonomic classifications. The category “No Hits” refers to reads that could not be matched to any sequence in the Kraken2 reference database. In other words, none of the k-mers derived from these reads had significant matches in the reference database. The category “Other Root” includes reads that are somewhat matched to sequences in the reference database but cannot be confidently assigned to a specific, lower-level taxonomic group. These reads have ambiguous classification results, leaving them categorized at the highest possible taxonomic level (the “root”). Thus, reads classified as “No hits” by Kraken2 but sorted into one of the three classes by the DL model call for further analysis since these reads may represent novel sequences very different from the established taxonomy tree.

The 3-class model predicted the virus class for the samples S23 and S27 with only ca. 10% of false positives originating from eukaryota. Host plants have frequently virus sequences incorporated in their genome, which might contribute to this classification result. The viral class of the pathogen-free sample S16 appears to be a random group of reads belonging to the categories eukaryota, “No hits” and “Others” predicted similarly by both models.

Both DL models correctly grouped mostly reads into the cassava class that indeed originated from the host plant. Moreover, the 3-class model included no reads from the category “Others”. The few additional reads (less than 1% for S16 and S23) that could not be traced to the plant genome are of unknown origin.

The bacteria class was not well predicted by the 3-class model. Most of the included reads originated from eukaryota. The result may indicate that the model did not have enough data to correctly classify bacterial reads. On the other hand, there is a huge discrepancy between the prediction of bacterial read when working on the nucleotide or the protein level as discussed in the Subsection 7.2.1.

Finally, Frame 4 shows the cassava viruses identified by PhytoPipe on reads labeled as viruses by the DL models. The comparison with the PhytoPipe output for the raw sample reads, see Frame 3, shows that a sufficient number of reads are kept by the DL models in the virus class to still identify most of the viruses present in the samples. However, the DL models fail to maintain sufficient reads for Cassava Torrado-like virus and Cassava satellite virus in sample S27. The absence of the latter can be

Sample	2 Class Model	3 Class Model
S16 (R1) **S16 (R2)**	No virus detected	
S23 (R1)	Cassava common mosaic virus	Cassava common mosaic virus Cassava Torrado-like virus
S23 (R2)	Cassava common mosaic virus Cassava Torrado-like virus	Cassava common mosaic virus Cassava Torrado-like virus
S27 (R1)	Cassava torrado-like virus 2 Cassava ampelovirus 1	Cassava ampelovirus 1 Cassava torrado-like virus 2
S27 (R2)	Cassava torrado-like virus 2 Cassava ampelovirus 1	Cassava ampelovirus 1 Cassava torrado-like virus 2

Frame 4 – Cassava viruses identified by PhytoPipe on the subset of reads previously classified as viral by the DL models.

explained by the fact that the present variant may be novel and the employed cut-off probability was too high. The absence of the former is curious, since the consensus sequence of the identified Cassava Torrado-like virus has nearly 99% blastn identity with the reference genome. We hypothesize that not enough reads are kept for correct assembly.

The performance analysis of the proposed 3-class model provided proof of concept that deep learning algorithms can successfully extract features for virus detection even from short not-assembled reads directly obtained from high-throughput sequencing. This fact has been previously doubted in the literature. We provided evidence that the key innovations are the attention mechanism and the use of protein references. Furthermore, the analysis revealed that DL models could be used as lightweight alternative for successfully filtering pathogen reads to be used in phytosanitary pipelines. The fishing for pathogen reads is an essential step, reducing the computational costs of the downstream tasks, but should not compromise the identification of (known or unknown) viruses in the sample. We state that the proposed model can be further optimized to serve as classification filter that keeps the most relevant viral reads among millions of others aiding in the timely detection of virus infections.

8 CONCLUSION

This work proposed a deep learning model for identifying unassembled viral reads within High-throughput sequencing (HTS) data obtained from RNA samples extracted from cassava. The primary goal of this model was to reduce the complexity and computational resource requirements of virus discovery. A significant motivation for developing this model was the lack of effective solutions for classifying short sequences, as state-of-the-art methods described in the literature typically underperform when applied to the 100-300nt HTS reads commonly found in most proposed approaches to similar problems.

We demonstrated that the proposed model could identify viral reads, which are relatively few in RNA-seq data containing millions of short sequences, and classify them into a single group. In addition to using protein-level references, a significant innovation was incorporating the attention mechanism to capture relationships between different k-mers within a sequenced read. The model effectively separates viral reads from non-viral reads, such as those from cassava or bacteria, in sufficient numbers to allow subsequent processing. Downstream tools in a phytosanitary pipeline, such as BLASTN and BLASTX, can then be used for final virus detection. The computational costs associated with these tools are significantly reduced when they operate on a subset of data provided by the proposed model's virus classification. This capability allows bioinformatics professionals and researchers to run a reliable virus identification pipeline on personal computers rather than relying on high-end servers.

However, when analyzing the performance of the model when identifying bacterial reads we found that the model requires further improvement. One of the main challenges in building the model for virus identification is the imbalance in available sequence data. Viral genomes are generally smaller and less represented in databases compared to plant and bacterial genomes. This imbalance often leads to overfitting and underfitting in many versions of the model trained. The imbalance cannot simply be eliminated by restricting the included plant genomes to the host plant and by choosing a representative reference set for bacteria. The number of fragments generated in silico is limited by hardware constraints and too small to cover the genomes of all plant infecting viruses. When analyzed the accuracy and loss values from train (see

Figure 36), the number of parameters (see Table 4), and the model idenfication performance (see Table 9), this leads us to conclude that the final model presents overfitting behavior.

Another significant contribution of this work was the implementation of a state-of-the-art phytosanitary pipeline, PhytoPipe, as a cloud service on AWS. This documented approach facilitates access to advanced bioinformatics tools for detecting pathogens in RNA-seq plant data, even in environments with limited computational resources. Additionally, generating reference data structures for tools like Kraken2 and Kaiju broadens the pipeline's applicability, as creating these structures can be more computationally intensive than the pipeline's execution. Finally, the direct comparison between traditional bioinformatics tools and deep learning-based read classification provides valuable insights into the relative performance and potential of these approaches. The source code for the model definition, the training script, and additional code can be accessed on the following GitHub page: <https://github.com/elissonlima/CassBERT>.

8.1 FUTURE WORKS

While this dissertation has made significant strides in developing a deep learning model for identifying viral reads in cassava RNA-seq data, there remain several avenues for enhancing the model's performance and usability. These opportunities for future work include:

- Improving the training dataset: By adjusting the representation of key bacterial and other pathogen genera, the model can be better equipped to distinguish between different types of sequences. This enhancement would likely improve the model's overall accuracy and robustness in identifying non-viral reads.
- Expanding dataset size: Increasing the size of the dataset to achieve more comprehensive genome coverage could help reduce underfitting. This expansion may require access to more substantial computational resources for model training but could lead to a more accurate and reliable model.
- Explore modifications of the model's architecture: Optimize the hyperparameters

of the model, for example the number of attention heads, and the embedding size. Also, it may be good to analyze specialized embeddings designed to represent sequence data, for example ProtVec (ASGARI; MOFRAD, 2015).

- Examine other modifications of the model design and training: Explore dropout and batch normalization to prevent overfitting, and try to find a trained foundation model that can be fine-tuned and applied to the presented problem.

- Analyzing attention maps: Investigating the learned attention maps could provide insights into k-mer features that are characteristic of viral sequences, beyond simple frequency analysis. Understanding these features may enable the development of more sophisticated detection mechanisms and improve the interpretability of the model's decision-making process.

- Integrating the deep learning model with PhytoPipe: By directly incorporating the deep learning model into PhytoPipe, the need to query reads against extensive and complex reference data structures could be eliminated. This integration would significantly reduce the computational resources required for pathogen detection pipelines, making them more accessible and efficient for bioinformatics experts.

BIBLIOGRAPHY

5 Types of Word Embeddings and Example NLP Applications. 2024. Disponível em: <https://swimm.io/learn/large-language-models/5-types-of-word-embeddings-and-example-nlp-applications>.

AL-AJLAN, A.; ALLALI, A. E. CNN-MGP: Convolutional neural networks for metagenomics gene prediction. *Interdiscip Sci*, Germany, v. 11, n. 4, p. 628–635, dez. 2018.

ALLALI, A. E.; ROSE, J. R. MGC: a metagenomic gene caller. *BMC Bioinformatics*, England, v. 14 Suppl 9, n. Suppl 9, p. S6, jun. 2013.

ALMEIDA, F.; XEXéO, G. *Word Embeddings: A Survey*. 2023. Disponível em: <https://arxiv.org/abs/1901.09069>.

ALZUBAIDI, L.; ZHANG, J.; HUMAIDI, A. J.; AL-DUJAILI, A.; DUAN, Y.; AL-SHAMMA, O.; SANTAMARÍA, J.; FADHEL, M. A.; AL-AMIDIE, M.; FARHAN, L. Review of deep learning: concepts, CNN architectures, challenges, applications, future directions. *Journal of Big Data*, v. 8, n. 1, p. 53, mar. 2021.

ASGARI, E.; MOFRAD, M. R. K. Continuous distributed representation of biological sequences for deep proteomics and genomics. *PLOS ONE*, Public Library of Science (PLoS), v. 10, n. 11, p. e0141287, nov. 2015. ISSN 1932-6203. Disponível em: <http://dx.doi.org/10.1371/journal.pone.0141287>.

BAHDANAU, D.; CHO, K.; BENGIO, Y. *Neural Machine Translation by Jointly Learning to Align and Translate*. 2016. Disponível em: <https://arxiv.org/abs/1409.0473>.

BIRUNDA, S. S.; DEVI, R. K. A review on word embedding techniques for text classification. Springer Singapore, Singapore, p. 267–281, 2021.

BRAUWERS, G.; FRASINCAR, F. A general survey on attention mechanisms in deep learning. *IEEE Transactions on Knowledge and Data Engineering*, Institute of Electrical and Electronics Engineers (IEEE), v. 35, n. 4, p. 3279–3298, abr. 2023. ISSN 2326-3865. Disponível em: <http://dx.doi.org/10.1109/TKDE.2021.3126456>.

CONSUL, S.; ROBERTSON, J.; VIKALO, H. Xvir: A transformer-based architecture for identifying viral reads from cancer samples. *bioRxiv*, Cold Spring Harbor Laboratory, 2023. Disponível em: <https://www.biorxiv.org/content/early/2023/08/29/2023.08.28.555020>.

CONTIG. 2024. Disponível em: <https://www.genome.gov/genetics-glossary/Contig>.

CONVOLUTIONAL Neural Network — Lesson 9: Activation Functions in CNNs. 2024. Disponível em: <https://medium.com/@nerdjock/convolutional-neural-network-lesson-9-activation-functions-in-cnns-57def9c6e759>.

CRISTINA, S. *The Attention Mechanism from Scratch*. 2023. Disponível em: <https://machinelearningmastery.com/the-attention-mechanism-from-scratch/>.

DAI, H.; UMAROV, R.; KUWAHARA, H.; LI, Y.; SONG, L.; GAO, X. Sequence2vec: a novel embedding approach for modeling transcription factor binding affinity landscape. *Bioinformatics (Oxford, England)*, 2017.

DESHPANDE, D.; CHHUGANI, K.; CHANG, Y.; KARLSBERG, A.; LOEFFLER, C.; ZHANG, J.; MUSZYŃSKA, A.; MUNTEANU, V.; YANG, H.; ROTMAN, J.; TAO, L.; BALLIU, B.; TSENG, E.; ESKIN, E.; ZHAO, F.; MOHAMMADI, P.; ŁABAJ, P. P.; MANGUL, S. RNA-seq data science: From raw data to effective interpretation. *Front Genet*, Switzerland, v. 14, p. 997383, mar. 2023.

FABIJAńSKA, A.; GRABOWSKI, S. Viral genome deep classifier. *IEEE Access*, v. 7, p. 81297–81307, 2019.

FABIJAńSKA, A.; GRABOWSKI, S. Viral genome deep classifier. *IEEE Access*, v. 7, p. 81297–81307, 2019.

FANG, Z.; TAN, J.; WU, S.; LI, M.; XU, C.; XIE, Z.; ZHU, H. PPR-Meta: a tool for identifying phages and plasmids from metagenomic fragments using deep learning. *Gigascience*, United States, v. 8, n. 6, jun. 2019.

FUKUSHIMA, K. K. neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 1980.

GE, L.; MOH, T.-S. Improving text classification with word embedding. In: *2017 IEEE International Conference on Big Data (Big Data)*. [S.l.: s.n.], 2017. p. 1796–1805.

GOMES, J. de C. *Cultivo da Mandioca para a Região dos Tabuleiros Costeiros*. 2024. Disponível em: <https://sistemasdeproducao.cnptia.embrapa.br/FontesHTML/Mandioca/mandioca_tabcosteiros/importancia.htm>. Acesso em: 20 mar. 2024.

GREFF, K.; SRIVASTAVA, R. K.; KOUTNíK, J.; STEUNEBRINK, B. R.; SCHMIDHUBER, J. Lstm: A search space odyssey. *IEEE Transactions on Neural Networks and Learning Systems*, v. 28, n. 10, p. 2222–2232, 2017.

HBVDB. 2024. Disponível em: <https://hbvdb.lyon.inserm.fr/HBVdb/HBVdbIndex>.

HU, X.; HURTADO-GONZALES, O. P.; ADHIKARI, B. N.; FRENCH-MONAR, R. D.; MALAPI, M.; FOSTER, J. A.; MCFARLAND, C. D. PhytoPipe: a phytosanitary pipeline for plant pathogen detection and diagnosis using RNA-seq data. *BMC Bioinformatics*, v. 24, n. 1, p. 470, dez. 2023.

IBGE. *Produção de Mandioca*. 2024. Disponível em: <https://www.ibge.gov.br/explica/producao-agropecuaria/mandioca/br>. Acesso em: 20 mar. 2024.

KIM, H.; BO, N.; LONG, H.; HIEN, N.; CEBALLOS, H.; HOWELER, R. Current situation of cassava in vietnam. 12 2017.

LANL. 2024. Disponível em: <https://lanl.libguides.com/az.php>.

LECUN, Y.; BOTTOU, L.; BENGIO, Y.; HAFFNER, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, v. 86, n. 11, p. 2278–2324, 1998.

LI, Z.; SUN, Y.; ZHU, J.; TANG, S.; ZHANG, C.; MA, H. Improve relation extraction with dual attention-guided graph convolutional networks. *Neural Computing and Applications*, v. 33, 03 2021.

LINDEMANN, B.; MüLLER, T.; VIETZ, H.; JAZDI, N.; WEYRICH, M. A survey on long short-term memory networks for time series prediction. *Procedia CIRP*, v. 99, p. 650–655, 2021. ISSN 2212-8271. 14th CIRP Conference on Intelligent Computation in Manufacturing Engineering, 15-17 July 2020. Disponível em: <https://www.sciencedirect.com/science/article/pii/S2212827121003796>.

LU, J.; SALZBERG, S. L. Ultrafast and accurate 16S rRNA microbial community analysis using kraken 2. *Microbiome*, v. 8, n. 1, p. 124, ago. 2020.

MACAVANEY, S.; YATES, A.; COHAN, A.; GOHARIAN, N. Cedr: Contextualized embeddings for document ranking. In: *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*. New York, NY, USA: Association for Computing Machinery, 2019. (SIGIR'19), p. 1101–1104. ISBN 9781450361729. Disponível em: <https://doi.org/10.1145/3331184.3331317>.

MAHESH, B. Machine learning algorithms -a review. 01 2019.

MATEOS, P. A.; BALBOA, R. F.; EASTEAL, S.; EYRAS, E.; PATEL, H. R. Pacific: a lightweight deep-learning classifier of sars-cov-2 and co-infecting rna viruses. *Scientific Reports*, v. 11, n. 1, 2021.

MENZEL, P.; NG, K. L.; KROGH, A. Fast and sensitive taxonomic classification for metagenomics with kaiju. *Nature Communications*, 2016.

METASIM. 2024. Disponível em: <https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0003373>.

MIKOLOV, T.; CHEN, K.; CORRADO, G.; DEAN, J. *Efficient Estimation of Word Representations in Vector Space*. 2013. Disponível em: <https://arxiv.org/abs/1301.3781>.

MIKOLOV, T.; CHEN, K.; CORRADO, G.; DEAN, J. *Efficient Estimation of Word Representations in Vector Space*. 2013. Disponível em: <https://arxiv.org/abs/1301.3781>.

MIN, S.; LEE, B.; YOON, S. Deep learning in bioinformatics. *Briefings in Bioinformatics*, v. 18, n. 5, p. 851–869, 07 2016. Disponível em: <https://doi.org/10.1093/bib/bbw068>.

MINAEE, S.; KALCHBRENNER, N.; CAMBRIA, E.; NIKZAD, N.; CHENAGHLU, M.; GAO, J. Deep learning–based text classification: A comprehensive review. *ACM Comput. Surv.*, Association for Computing Machinery, New York, NY, USA, v. 54, n. 3, 04 2021. ISSN 0360-0300. Disponível em: <https://doi.org/10.1145/3439726>.

NATURAL Language Processing (NLP): What Is It & How Does it Work? 2024. Disponível em: <https://monkeylearn.com/natural-language-processing/>. Acesso em: 31 mar. 2024.

NCBI. 2024. Disponível em: <https://www.ncbi.nlm.nih.gov/>.

NIU, Z.; ZHONG, G.; YU, H. A review on the attention mechanism of deep learning. *Neurocomputing*, v. 452, p. 48–62, 2021. ISSN 0925-2312. Disponível em: <https://www.sciencedirect.com/science/article/pii/S092523122100477X>.

ORPHELIA. 2024. Disponível em: <https://academic.oup.com/nar/article/37/suppl_2/W101/1134091>.

OTTER, D. W.; MEDINA, J. R.; KALITA, J. K. A survey of the usages of deep learning for natural language processing. *IEEE transactions on neural networks and learning systems*, 2020.

PATIL, B. L. Cassava mosaic geminiviruses: actual knowledge and perspectives. *Molecular plant pathology*, v. 10, n. 5, p. 685–701, 2009.

PETERS, M. E.; NEUMANN, M.; IYYER, M.; GARDNER, M.; CLARK, C.; LEE, K.; ZETTLEMOYER, L. Deep contextualized word representations. *CoRR*, abs/1802.05365, 2018. Disponível em: <http://arxiv.org/abs/1802.05365>.

PONNE, B. *Understand TF-IDF in Python*. 2023. Disponível em: <https://www.codingthepast.com/2023/04/26/Text-Mining-in-Python.html>.

PROPHET. 2024. Disponível em: <https://github.com/jaumlrc/ProphET>.

RAMOS, J. Using tf-idf to determine word relevance in document queries. 01 2003.

READ. 2019. Disponível em: <https://www.genomicseducation.hee.nhs.uk/glossary/read/>.

REN, J.; SONG, K.; DENG, C.; AHLGREN, N. A.; FUHRMAN, J. A.; LI, Y.; XIE, X.; POPLIN, R.; SUN, F. Identifying viruses from metagenomic data using deep learning. *Quant Biol*, China, v. 8, n. 1, p. 64–77, mar. 2020.

RYBICKI, E. P. A top ten list for economically important plant viruses. *Archives of Virology*, v. 160, n. 1, p. 17–20, 2015.

SARKER, I. H. Machine learning: Algorithms, real-world applications and research directions. *SN Computer Science*, 2021.

SCHMIDT, R. M. *Recurrent Neural Networks (RNNs): A gentle Introduction and Overview*. 2019. Disponível em: <https://arxiv.org/abs/1912.05911>.

SHERSTINSKY, A. Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network. *Physica D: Nonlinear Phenomena*, v. 404, p. 132306, 2020. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0167278919305974>.

SIMS, G. E.; JUN, S.-R.; WU, G. A.; KIM, S.-H. Alignment-free genome comparison with feature frequency profiles (ffp) and optimal resolutions. 2009.

SUKHORUKOV, G.; KHALILI, M.; GASCUEL, O.; CANDRESSE, T.; MARAIS-COLOMBEL, A.; NIKOLSKI, M. Virhunter: A deep learning-based method for detection of novel rna viruses in plant sequencing data. *Frontiers in Bioinformatics*, v. 2, 2022. ISSN 2673-7647. Disponível em: <https://www.frontiersin.org/journals/bioinformatics/articles/10.3389/fbinf.2022.867111>.

SZE, V.; CHEN, Y.-H.; YANG, T.-J.; EMER, J. Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, v. 105, 03 2017.

TBLASTX. 2024. Disponível em: <https://blast.ncbi.nlm.nih.gov/Blast.cgi?PROGRAM=tblastx&PAGE_TYPE=BlastSearch&LINK_LOC=blasthome>.

TRIPATHY, A.; AGRAWAL, A.; RATH, S. K. Classification of sentimental reviews using machine learning techniques. *Procedia Computer Science*, v. 57, p. 821–829, 2015. ISSN 1877-0509. 3rd International Conference on Recent Trends in Computing 2015 (ICRTC-2015). Disponível em: <https://www.sciencedirect.com/science/article/pii/S1877050915020529>.

WOOD, D. E.; LU, J.; LANGMEAD, B. Improved metagenomic analysis with kraken 2. *Genome Biology*, v. 20, n. 1, p. 257, nov. 2019.

WOOD, D. E.; SALZBERG, S. L. Kraken: ultrafast metagenomic sequence classification using exact alignments. *Genome Biology*, 2014.

YAMASHITA, R.; NISHIO, M.; DO, R. K. G.; TOGASHI, K. Convolutional neural networks: an overview and application in radiology. *Insights into Imaging*, v. 9, n. 4, p. 611–629, ago. 2018.

YIN, W.; KANN, K.; YU, M.; SCHüTZE, H. Comparative study of cnn and rnn for natural language processing. 2017. Disponível em: <https://arxiv.org/abs/1702.01923>.

APÊNDICE A – PYTHON SCRIPT TO GENERATE RANDOM READS

```
def get_random_reads_from_seq(input_file,
                              output_file,
                              max_read_len,
                              depth,
                              strand='both'):
    '''
    Generate random reads from sequence string

        Parameters:
            sequence (str): Sequence as string
            seq_name (str): Sequence id
            max_read_len (str): Max size of output reads
            depth:
                int value -> How many time Increase the
                    number of out reads
                "max" -> Get maximum distinct reads from
                    sequence
            strand (str):
                both -> generate forward or reverse read (randomly)
                antisense -> only generate reverse read
        Returns:
    '''

    fasta_sequences = SeqIO.index(input_file, "fasta")
    arq_out =  open(output_file, "w")

    for idx in fasta_sequences.keys():

        sequence = str(fasta_sequences[idx].seq)

        maxstart = len(sequence) - max_read_len + 1
        maxbp = len(sequence) # max number of bp to be generated
        if depth == "max":
            maxbp = float('inf')
        else:
            maxbp *= depth
        #$thisseq =~ tr/ACGT/TGCA/;
        bpcount = 0
        repetead_count = 0

        result_set = set()

        if len(sequence) < max_read_len:
            continue
```

```
45          elif len(sequence) == max_read_len:
                arq_out.write(">%s\n%s\n" % (fasta_sequences[idx].id,
47                                          sequence))

49          while bpcount < maxbp and repetead_count < len(sequence):

51              startidx = random.randint(0, maxstart)
                new_seq = sequence[startidx:startidx+max_read_len]
53
                num_strand = 1
55              if strand == "both":
                    if startidx % 2 == 0:
57                      num_strand = -1
                        new_seq = rev_comp(new_seq)
59              elif strand == "antisense":
                    new_seq = rev_comp(new_seq)
61
                    # new_seq = new_seq[::-1]
63                  # translate_dict = new_seq.maketrans("ACGT","TGCA")
                    # new_seq = new_seq.translate(translate_dict)
65
                if new_seq in result_set:
67                  repetead_count += 1
                else:
69                  repetead_count = 0
                    result_id = "%s:%d:%d" %\
71                  fasta_sequences[idx].id,startidx,num_strand)
                    arq_out.write(">%s\n%s\n" % (result_id, new_seq))
73
                bpcount += max_read_len
```

APÊNDICE B – TERRAFORM SCRIPT TO MANAGE AWS RESOURCES FOR RUNNING PHYTOPIPE

```
# Create an VPC (Virtual Private Cloud) on the Account
resource "aws_default_vpc" "default" {
  tags = {
    Name = "Default VPC"
  }
}

# Create an security group. This resource enables the virtual machine
# to connect to the internet and enable us to access the virtual machine
# using an SSH client.
resource "aws_security_group" "ingress-all-test" {
name = "allow-all-sg"
vpc_id = "${aws_default_vpc.default.id}"

ingress {
    cidr_blocks = [
      "0.0.0.0/0"
    ]
from_port = 22
    to_port = 22
    protocol = "tcp"
  }

  egress {
   from_port = 0
   to_port = 0
   protocol = "-1"
   cidr_blocks = ["0.0.0.0/0"]
 }
}
# Create an Key Pair to authenticate on the SSH.
resource "aws_key_pair" "deployer" {
  key_name   = "deployer-key"
  public_key = ""
}

# This create a Policy for the Virtual Machine. A Policy gives permission to the
# Virtual Machine to access other AWS Service. Here, for example, we give
    permission
# to the VM access the files on the S3 bucket that contains the input files to
# execute the Phytopipe experiments, and to write the output of the Phytopipe so
# these informations will not be lost when the VM is deleted after execution.
```

```
resource "aws_iam_policy" "s3-full-access" {
  name = "s3-full-access"
  path = "/"
  description = ""
  policy = jsonencode(
    {
      Version = "2012-10-17"
      Statement = [
        # Permission to access S3 Bucket
        {
          Sid = "VisualEditor1"
          Effect = "Allow"
          Action = "s3:*"
          Resource = [ "arn:aws:s3:::clovis-phytopipe-inout/*",
                       "arn:aws:s3:::clovis-phytopipe-inout"]
        }
      ]
    }
  )
}

# A role is a container for 1 or many Policies
resource "aws_iam_role" "ec2-s3-full-access" {
  name = "ec2-s3-full-access"
  assume_role_policy = jsonencode(
    {
      Version = "2012-10-17"
      Statement = [
        {
          Action = "sts:AssumeRole"
          Effect = "Allow"
          Sid = ""
          Principal = {
            Service = "ec2.amazonaws.com"
          }
        }
      ]
    }
  )
}

# Attach the policy on the IAM Role.
resource "aws_iam_role_policy_attachment" "custom" {
  role = aws_iam_role.ec2-s3-full-access.name
  policy_arn = aws_iam_policy.s3-full-access.arn
}
```

```
90
# It creates the virtual machine that will run the Phytopipe
92 resource "aws_instance" "server_processing" {
  # The AMI is the Ubuntu VM Image that contains all necessary software
94   ami = "ami-03c0b10ce2bbd7bf6"
  # The Availability Zone is the geographic region where the VM will be deployed.
96   availability_zone = "us-east-1c"
  # The instance type defines the computational resource of the VM. For this case
98   # we created an instance with 387GB of RAM and 48 vCPUs.
  instance_type = "r5a.12xlarge"
100   # Attach the security group created previously
  vpc_security_group_ids = ["${aws_security_group.ingress-all-test.id}"]
102   # Attach the SSH Key Pair created previously
  key_name = aws_key_pair.deployer.key_name
104   # Attach the permissions created previously
  iam_instance_profile = aws_iam_instance_profile.ec2-profile.name
106
  # Create some Tags to identify the VM created.
108   tags = {
    Name = "PhytoPipe"
110     Phase = "Process"
  }
112 }
```

APÊNDICE C – BASH SCRIPT TO RUN A TRAINING JOB ON APUANA

```
#!/bin/bash
#SBATCH --job-name=seqjob
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=8
#SBATCH --gpus=1
#SBATCH --mem=64G
#SBATCH -w cluster-node10
#SBATCH --output=log/output.txt
#SBATCH --error=log/error.txt

hostname

module load Python/3.8
module load CUDA
module load cuDNN
source $HOME/venv/bin/activate

CUDNN_PATH=$(dirname $($HOME/venv/bin/python -c \
"import nvidia.cudnn;print(nvidia.cudnn.__file__)"))
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$CUDNN_PATH/lib
export XLA_FLAGS=--xla_gpu_cuda_data_dir=/opt/easybuild/software/CUDA/11.8.0

echo $CUDNN_PATH
echo $LD_LIBRARY_PATH
echo $XLA_FLAGS

$HOME/venv/bin/python $HOME/CassBERT/run_model.py
#$HOME/venv/bin/python $HOME/CassBERT/classificator.py
```

APÊNDICE D – PRE-BUILT FILES FOR NCBI DATABASES

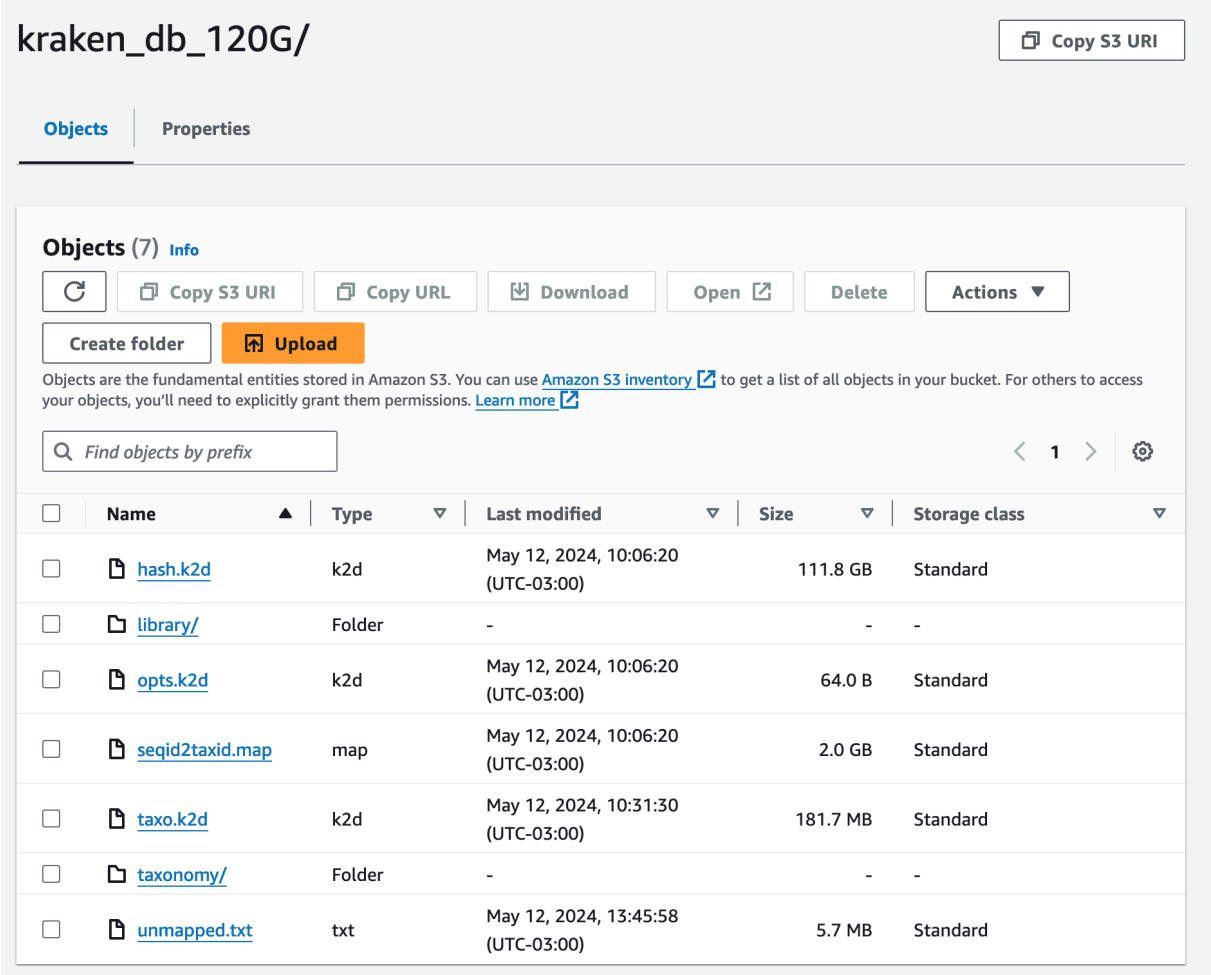


Figure 38 – Pre-built files for the NCBI non-redundant nucleotide database that can be used to run Kraken2

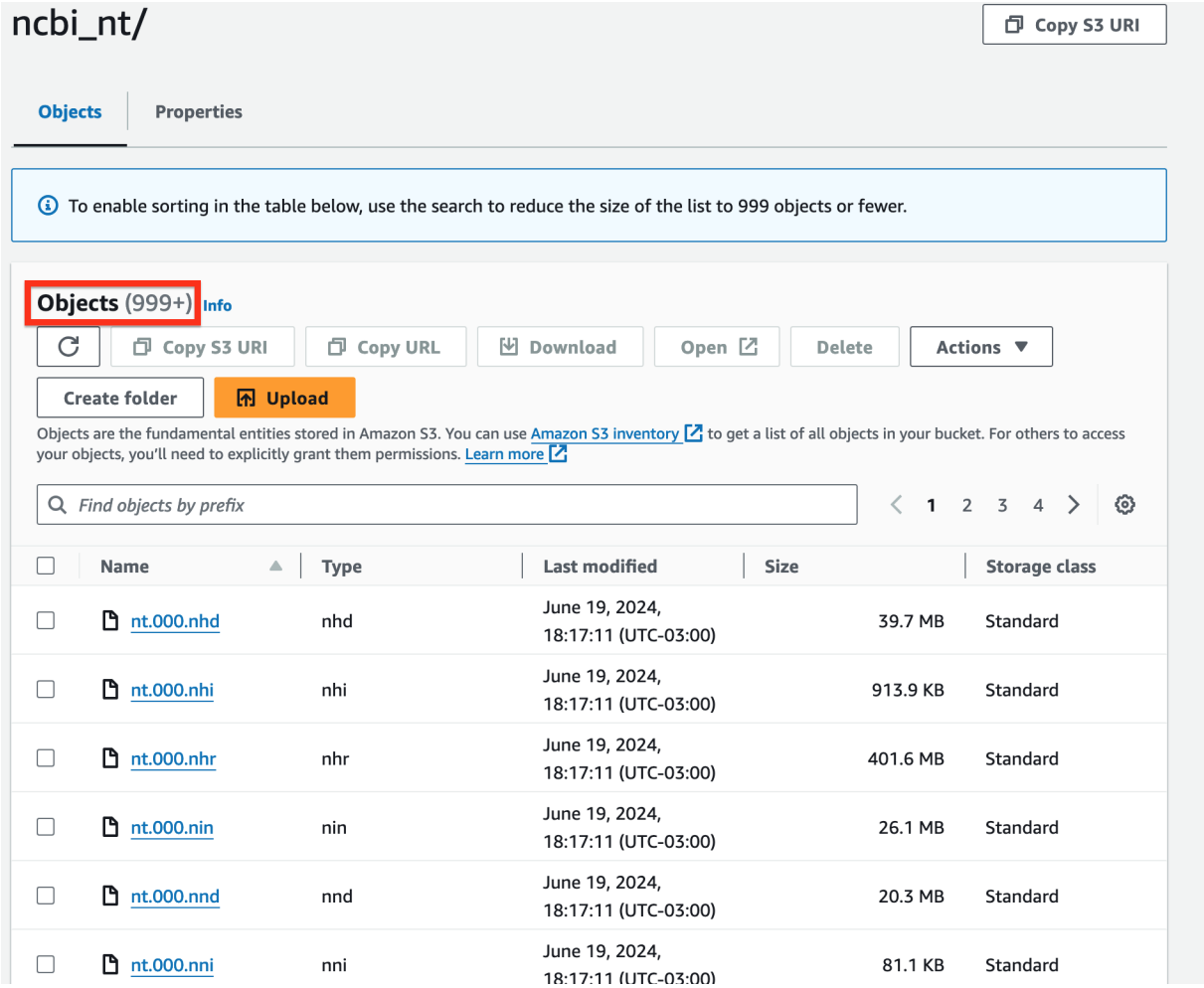


Figure 39 – Pre-built files for the NCBI nucleotide BLAST database. The large number of objects is highlighted.

APÊNDICE E – PROCEDURE FOR DOWNLOADING VIRAL DATASET

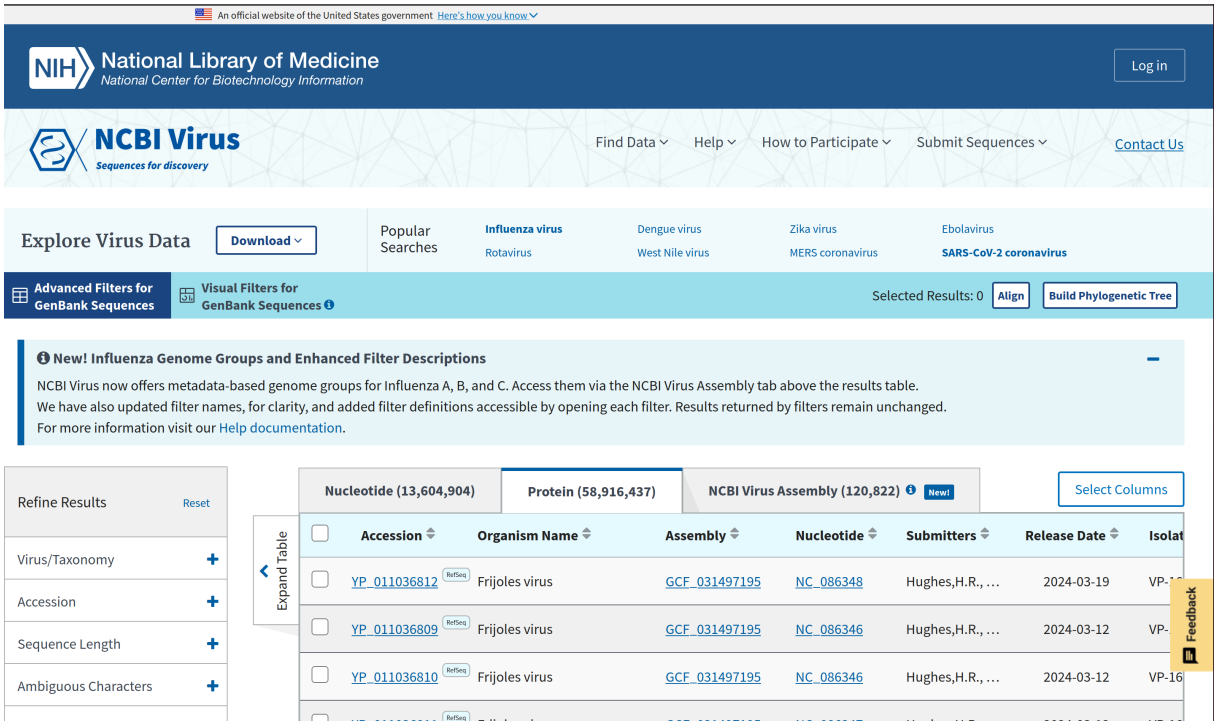


Figure 40 – First step to download the viral dataset: Access the NCBI virus portal and select the protein tab.

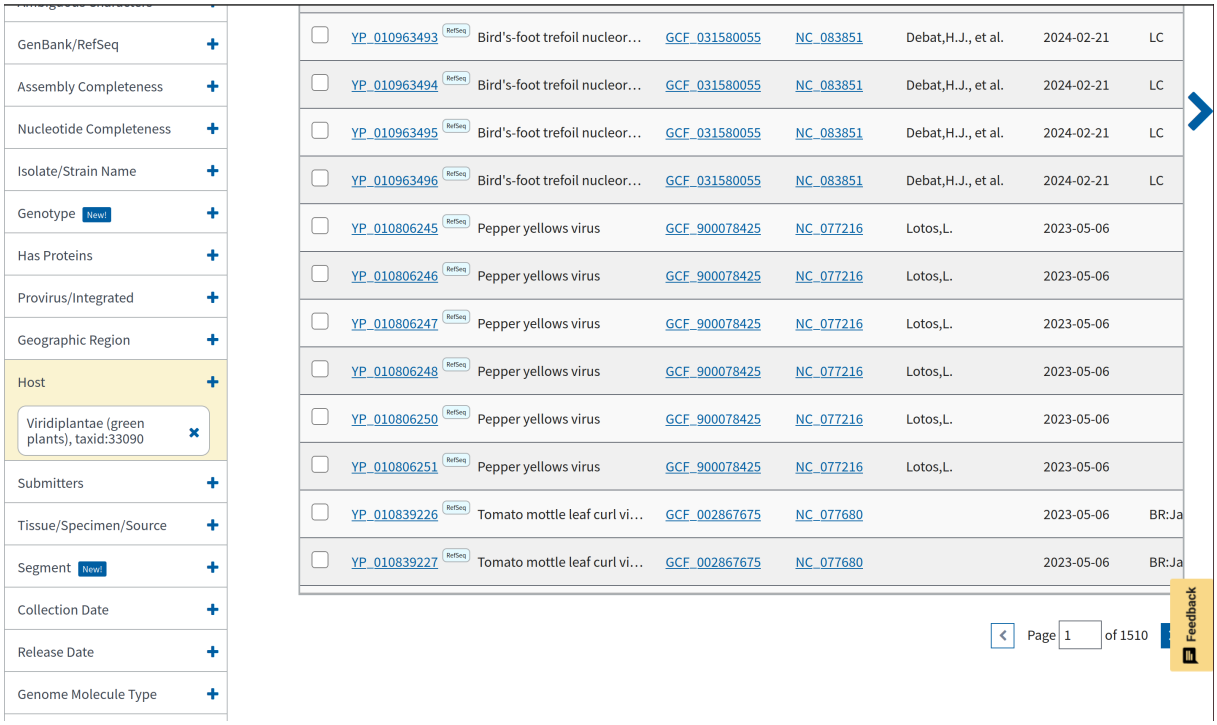


Figure 41 – Second step to download the viral dataset: Filter for *Viridiplantae* hosts.

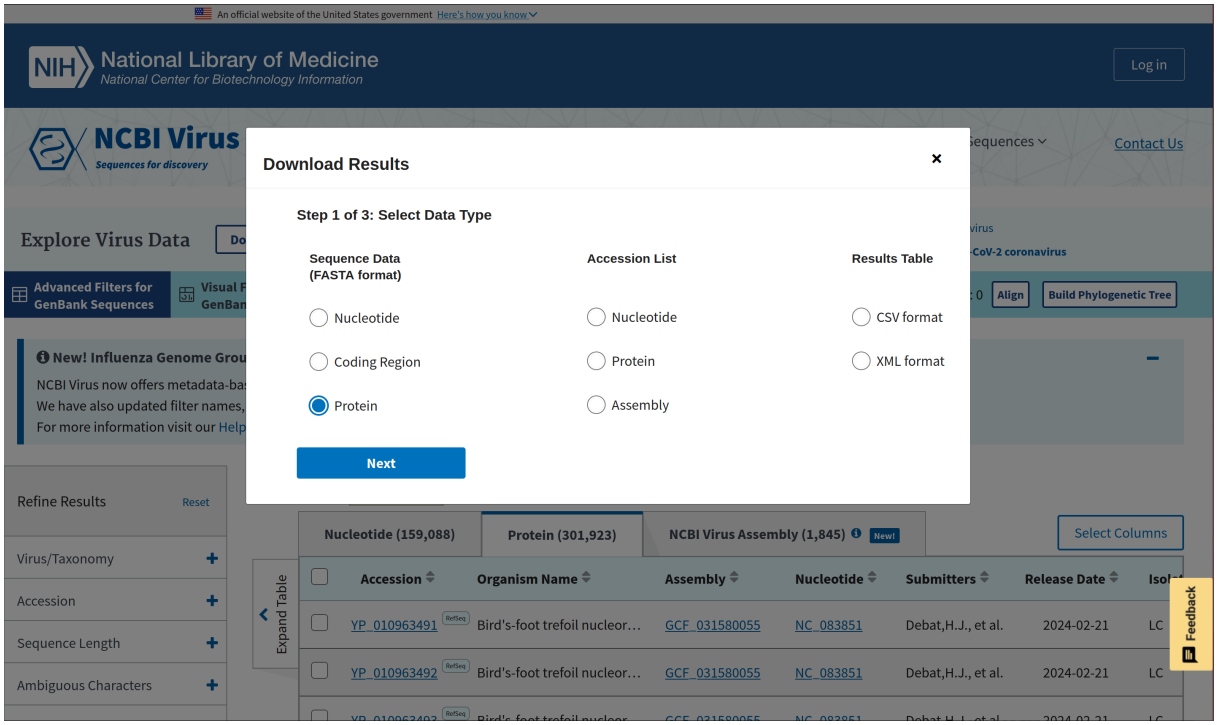


Figure 42 – Third step to download the viral dataset: Click on the “Download” button and select only the option “Protein” in the “Sequence Data” column.

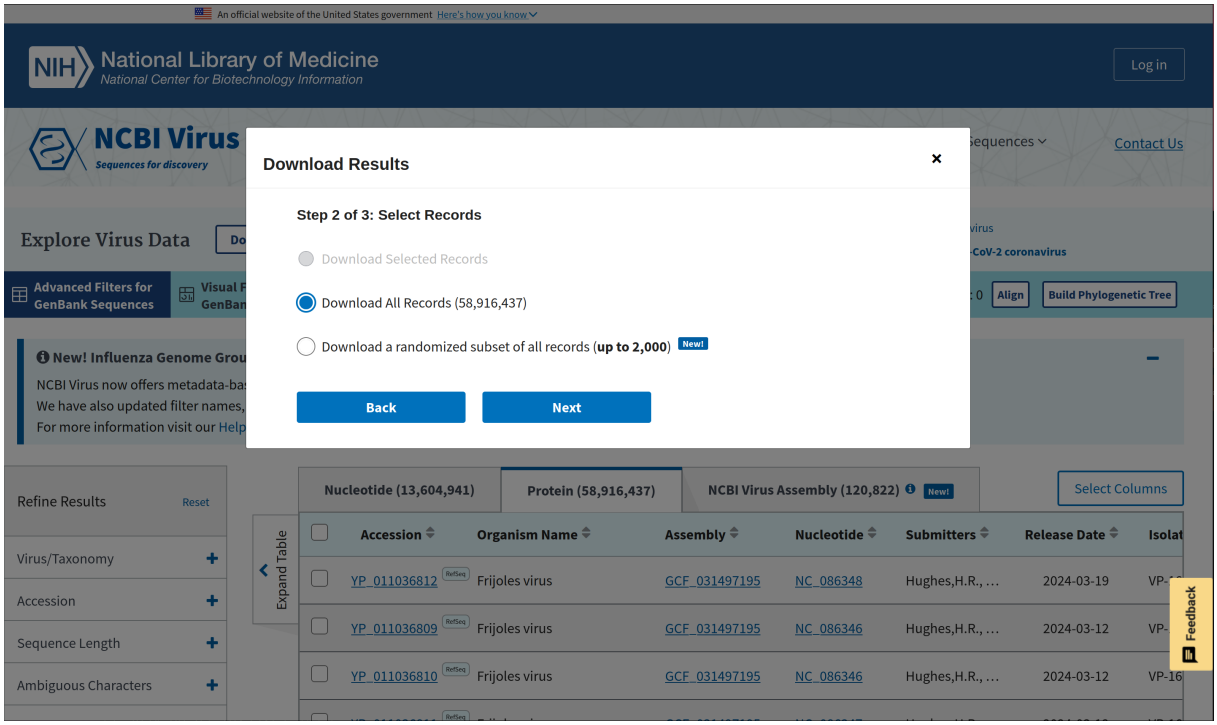


Figure 43 – Fourth step to download the viral dataset: Select “All Records” to opt for downloading all sequences.

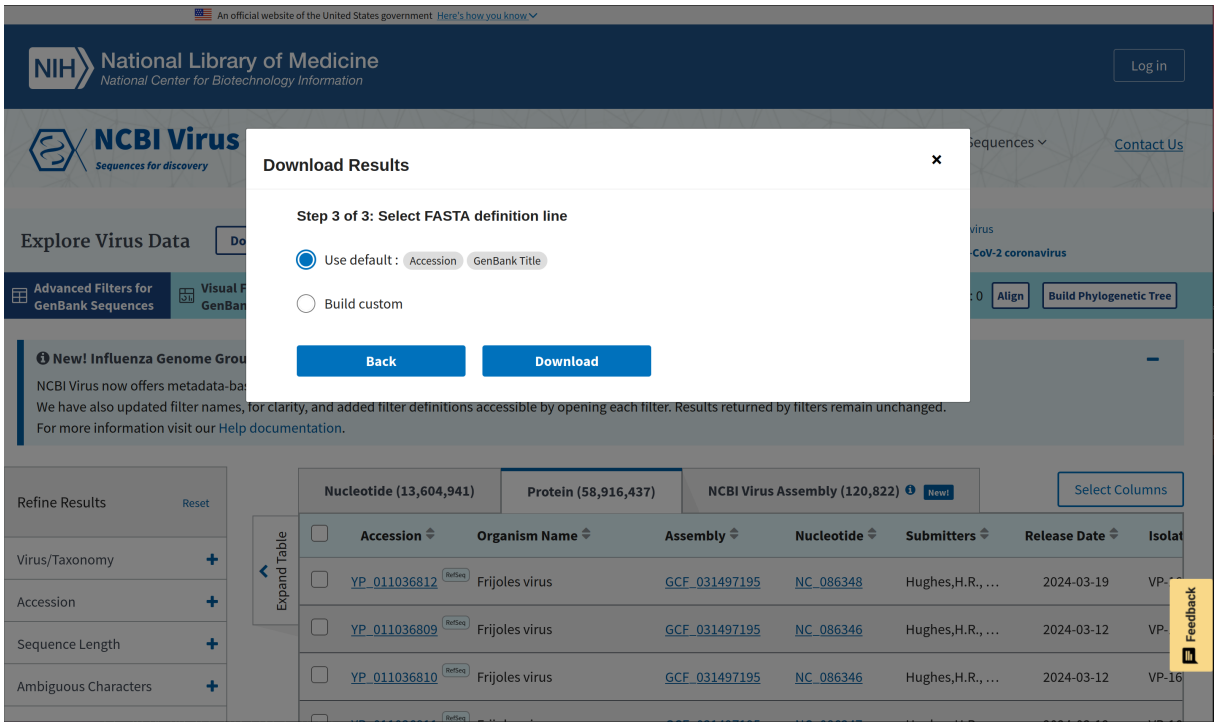


Figure 44 – Fifth step to download the viral dataset. Select “Use default” for the FASTA definition line and click the “Download” button to start the download.

APÊNDICE F – PROCEDURE FOR DOWNLOADING CASSAVA DATASET

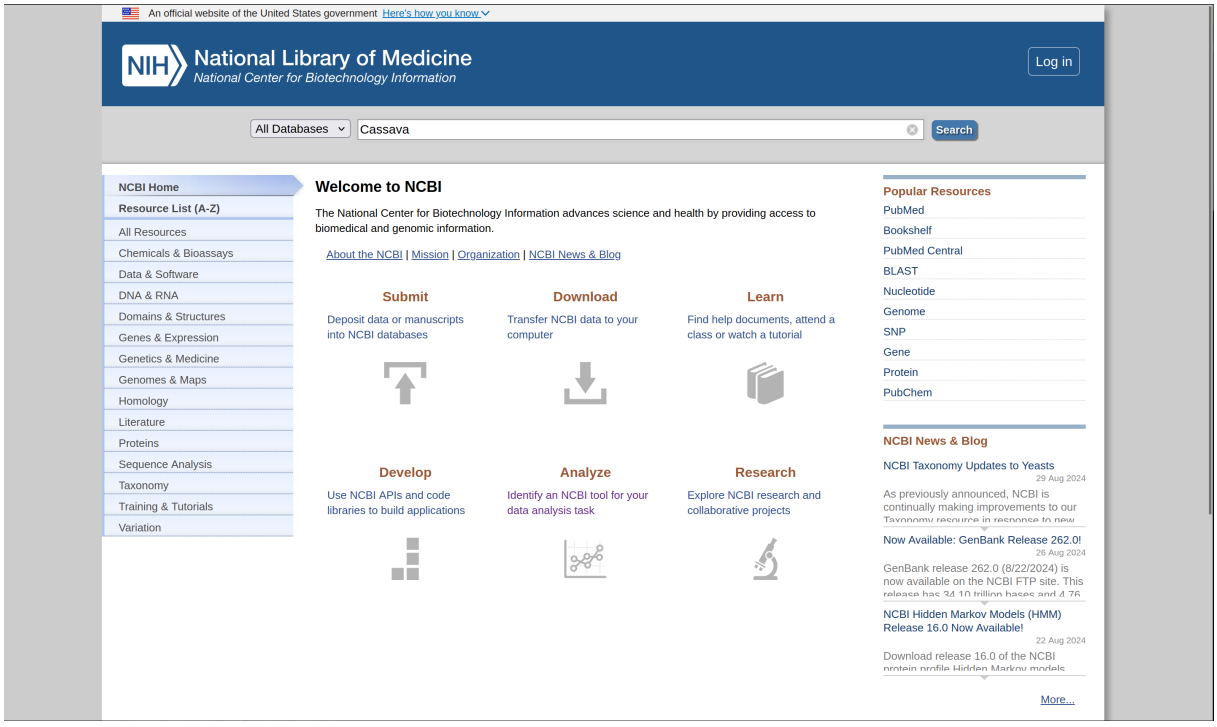


Figure 45 – First step to download the Cassava dataset: Access the NCBI portal and search for cassava.

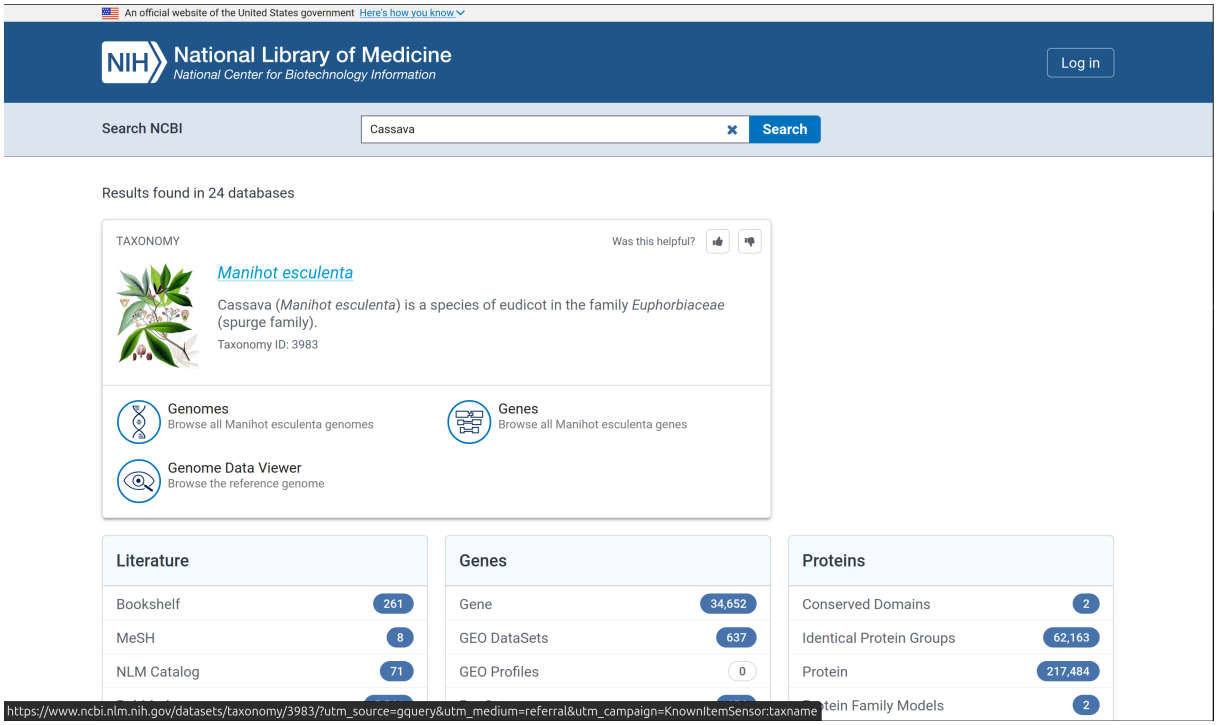


Figure 46 – Second step to download the Cassava dataset: Click on the “Manihot esculenta” link.

Figure 47 – Third step to download the Cassava dataset: Click on the “Download” button in the “Reference genome” section.

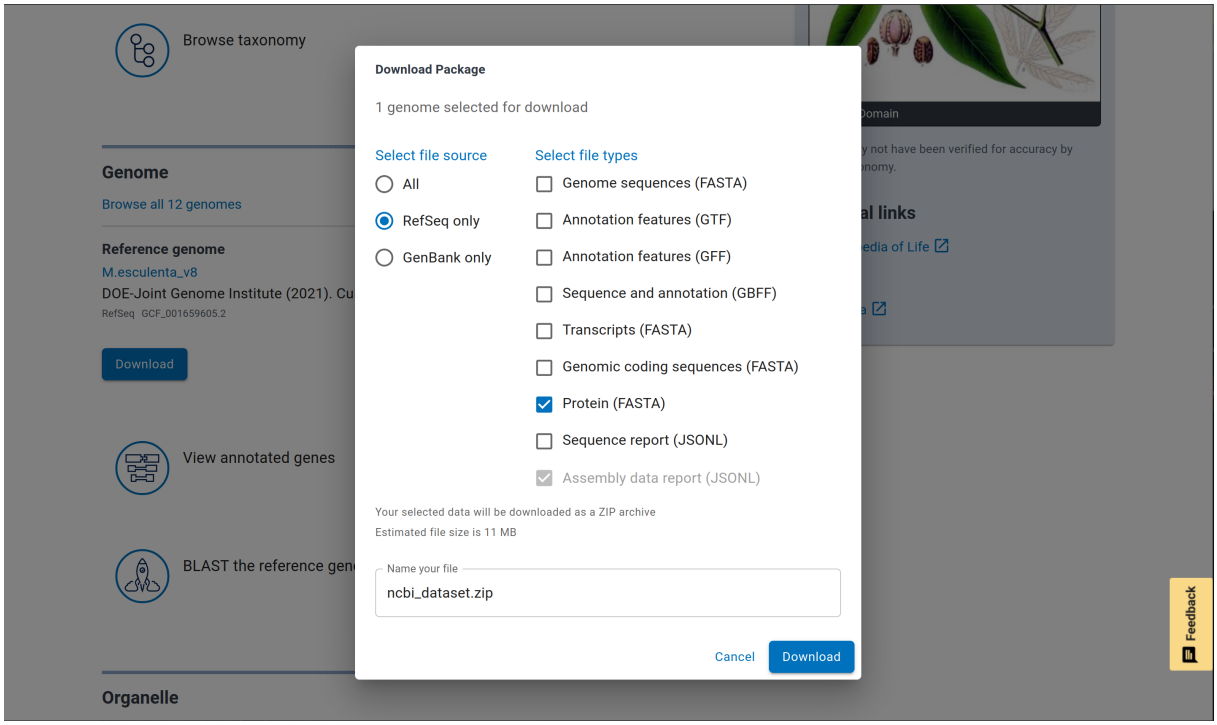


Figure 48 – Fourth step to download the Cassava dataset: Select “RefSeq only” on the “Select file source” column and select “Protein (FASTA)” on the “Select file format” column.

APÊNDICE G – PROCEDURE FOR DOWNLOADING BACTERIA DATASET

```
import pandas as pd
import os
input_file = "kaiju.classified.names.out"
cols = ['classified',
        'sequence_id',
        'unknow_number_1',
        'unknow_number_2',
        'unknow_number_3',
        'ncbi_id', 'sequence', 'taxonomy_path']
df = pd.read_csv(input_file, names=cols, sep='\t')
def filter_bacteria(x):
    line_list = x.split(';')
    if line_list[0].strip() == 'Bacteria':
        return line_list[2]
    else:
        return 'NA'

paths = df['taxonomy_path']
more_commons = paths.apply(filter_bacteria).value_counts()
for val in list(more_commons[:5]):
    command = f'datasets.exe download genome taxon \"{val}\" --reference\
    --include protein --assembly-level complete --filename {val}.zip'
    os.system(command)
    os.system(f'move {val}.zip bacterias\\')
    print("OK", val)
```