Investigation of *hsdS* Phase Variants Distribution in *Streptococcus suis* Isolates from Healthy and Diseased Swine Populations

MSc. RESEARCH PRACTICE REPORT

Written by:

Muhammad Hamzah Syaifullah Azmi, B.Sc.

1280147

Daily Supervisor:

Maria Juanpere Borras, MSc.

Supervisors:

prof. dr. Jerry Wells

dr. ir. Peter van Baarlen

dr. Jos Boekhorst

LABORATORY OF HOST-MICROBES INTERACTOMICS (HMI)

WAGENINGEN UNIVERSITY AND RESEARCH

February 2025

PREFACE & ACKNOWLEDGEMENTS

It has been four months since I first joined the Host-Microbes Interactomics (HMI) chair group at Wageningen University and Research as an MSc research practice student. Throughout my research, I have gained invaluable skills, techniques, and knowledge critical for genomics studies. These include multiplexing and DNA library preparation, Nanopore sequencing procedures, and developing bioinformatics pipelines to process Nanopore sequencing results and perform statistical analyses. This progress would not have been possible without the support and guidance of several individuals, and I am especially grateful to **Maria Juanpere Borras, MSc.**, my daily supervisor. I would like to extend my deepest gratitude to you for your unwavering assistance and encouragement. You introduced me to the phase-variable type I methyltransferase systems in *Streptococcus suis* and guided me to explore this research area in depth. Your support, particularly during challenging moments, was invaluable, and your kindness and patience were truly exceptional. I am immensely grateful for the way you have supervised me—with understanding, patience, and a recognition that I am still learning. You created space for self-exploration while ensuring I stayed on the right track. I am especially thankful for your efforts in helping me understand coding and scripting in the Python environment for the first time, which has been a significant milestone for me. For any mistakes I may have made during my research practice, I sincerely apologize to you. I wish you all the very best for your PhD project and your future endeavors. It has been a privilege to work under your supervision.

Also, I would like to express my heartfelt gratitude to **dr. Jos Boekhorst**, the bioinformatician in the HMI group and one of my supervisors, for his invaluable support and guidance throughout my research. His expertise was instrumental in optimizing the Python script and ensuring its efficiency. He also assisted me in setting up my device for remote work, enabling me to carry out my tasks seamlessly. His deep knowledge in bioinformatics significantly enhanced my understanding of how sequencing results should be processed and analyzed. Moreover, his numerous suggestions and alternative approaches greatly contributed to the development and refinement of the bioinformatics pipeline. I sincerely appreciate his patience, guidance, and generosity in sharing his expertise, which has been crucial to the completion of my research.

I would like to extend my sincere gratitude to **prof. dr. Jerry Wells**, the Principal Investigator of the project and the head of the HMI chair group, for providing the opportunity for me to become a part of this esteemed research group. His leadership and vision have created an inspiring and supportive environment, which greatly contributed to the progress of my research. Although I did not have the chance to interact with him directly, I am deeply appreciative of the resources and guidance made available under his supervision, which were essential for the successful completion of this research practice. Furthermore, I am honored to have him as one of the examiners for my thesis and truly value the time and expertise he dedicated to reviewing my work. His commitment to excellence in host-microbes interactomics has been a source of inspiration throughout this journey. I hope that maybe in the future, there will be another opportunity or possibility for me to work again together with your group, whether as a PhD student, research assistant, or collaborator researcher.

I would also like to express my deepest gratitude to **dr. ir. Peter van Baarlen** for his leadership and for giving me the exceptional opportunity to work on this research as part of his project. His guidance, though indirect, has been instrumental in shaping my work through Maria's supervision, and I truly appreciate the trust he placed in me to contribute to this study. His expertise, vision, and dedication to fostering a collaborative and supportive research environment have significantly influenced my learning experience and professional growth. I am incredibly grateful for the opportunity to be involved in this project under his leadership, and I sincerely hope for the chance to collaborate again in the future, whether as a PhD student, research assistant, or fellow researcher.

I would like to extend my sincere gratitude to **all the members of the HMI lab** for their invaluable support throughout my research. Their willingness to offer guidance, technical assistance, and insightful discussions significantly contributed to the progress of my study, especially during times when Maria was unavailable. Whether it was troubleshooting experiments, providing constructive feedback, or simply offering words of motivation, their contributions were instrumental in shaping the outcome of this work. I am deeply appreciative of their generosity in sharing their time and knowledge, and I am grateful to have been part of such a supportive research community.

I would also like to say thank you very much for the **LPDP scholarship** that has been providing me with allowances for my study at Wageningen University and Research.

Also, **Audrey Angelina Putri Taharuddin**, my girlfriend, who has been my main support system during my study and helped me to keep going when I started to feel demotivated or burnt out. You have become my main motivation to keep going and finish my thesis and my studies at Wageningen University and Research. Even though we are in a Long-Distance Relationship (Indonesia-The Netherlands), I am really grateful that we still get in touch with each other every day via phone, because you are the place where I can talk about my day.

And for **Juan Dennis Bahrian**, my best friend who lives next to my room. You have helped me a lot to reduce my anxieties and worries during my study here. You are the only one who is close to me when I am far away from all the people I know (in distance). You have also explained to me how to use several programs because of your excellent understanding of bioinformatics and statistics.

And lastly to **my mother, my brother, and my sister** in Indonesia who have been giving me a lot of support and suggestions when I encountered some troubles during my study here.

Muhammad Hamzah Syaifullah Azmi

1280147

TABLE OF CONTENTS

ABSTRACT

BACKGROUND AND OBJECTIVES. *Streptococcus suis* is a Gram-positive swine pathogen responsible for meningitis, arthritis, pneumonia, and septicemia, with zoonotic potential. To establish infection, *S. suis* employs virulence factors, including phase-variable genes, such as *hsdS* in the Type I methyltransferase system, which encodes the DNA recognition protein for the system. Homologous recombination between *hsdS* and its silent duplicate *hsdS'* generates recombinant phase variants (A, B, C, and D), potentially influencing virulence. This research aims to investigate the *hsdS* phase variants distribution in *S. suis* Isolates from healthy and diseased swine populations. By integrating Nanopore sequencing and bioinformatics analysis, this study assesses whether a specific *hsdS* phase variant is associated with the host health status, enhancing understanding of phase variation's roles in *S. suis* adaptation and pathogenicity.

EXPERIMENTAL METHODS. The research was conducted in two major parts: laboratory work and bioinformatics analysis. The laboratory work involved the amplification and barcoding of *hsdS-hsdS'* by PCR using multiple primer sets (ps1, ps2, and ps3), library preparation (end-prep and adapter ligation), and Nanopore sequencing using MinION Flow Cell. The raw signal data was converted to nucleotide sequence data using Dorado basecaller. The bioinformatics analysis involved the development of an integrated pipeline to process sequencing results, including primer screening, *S. suis* classification, size classification, *hsdS* screening, barcodes demultiplexing, sequence ends trimming, isolate source classification, *hsdS* phase variant identification, and statistical analysis (non-parametrical: Chi-square test, Mann-Whitney U test, and Kruskal-Wallis test). Pipelines were developed using Python 3 (v3.9.2) and executed in a Linux-based environment within a Conda 3 (v24.11.2).

RESULTS. Amplification of the *hsdS-hsdS'* region revealed that ps1 failed in most samples, producing only smearing. However, ps2 and ps3 showed a better amplification efficiency, as each reaction produced multiple bands, indicating non-specific amplification. Despite this, in some samples, a band within the size range of 2.5-4.0 kb can be observed, possibly representing successful *hsdS-hsdS'* amplification. Following Nanopore sequencing, a significant proportion of reads for each primer screening were discarded after the primer screening, *S. suis* classification, size classification, *hsdS* phase variant identification, and barcode demultiplexing, thus refining the dataset for statistical analysis. The Chi-square test yielded a test statistic of 2.977 ($p = 0.395$) and 1.223 ($p = 0.7476$) for healthy and diseased isolate groups, respectively, after data normalization. This indicates that *hsdS* phase variant distribution is independent of host health status. Phase variant B was most prevalent in both groups (95.35% in healthy, 95.59% in symptomatic), suggesting selective retention, structural stability or a functional role of this phase variant in bacterial fitness and gene expression regulation. The Mann-Whitney U test further confirmed no statistically significant phase variant distribution between both isolate groups. A Kruskal-Wallis test assessed primer set influence, with all p-values >0.05, confirming no primer-induced bias.

KEYWORDS

***hsdS* phase variants, Nanopore sequencing, phase-variable, *Streptococcus suis*, virulence.**

1. INTRODUCTION

1.1. Research Background and Objectives

Streptococcus suis (S. suis) is a Gram-positive bacterium that primarily infects swine but is also capable of causing zoonotic infections in humans, representing a potential public health risk [1, 2]. In swine, *S. suis* is the primary contributor to several health conditions, including meningitis, arthritis, pneumonia, and septicemia. These diseases can significantly reduce the productivity of the swine industry which could eventually lead to serious economic problems, especially in regions where pork is widely consumed [3, 4]. Humans typically contract *S. suis* primarily through direct exposure to diseased pigs or contaminated pork, making individuals in the swine industry, such as farmers, veterinarians, and slaughterhouse workers, particularly vulnerable to infection [2, 5, 6]. In humans, *S. suis* can cause serious diseases, including meningitis, septicemia, endocarditis, and in some cases, irreversible hearing loss or even fatal outcomes [7–9]. *S. suis* infections in both humans and swine are generally treated with antibiotics, commonly from the β-lactams, aminoglycosides, amphenicols, and fluoroquinolones classes [10, 11]. However, the lack of a universal and effective vaccine to prevent *S. suis* infections, combined with the high rates of infection and associated mortality, has resulted in widespread and prolonged antibiotic use. This extensive use exerts selective pressure, promoting the emergence of antimicrobial-resistant (AMR) strains of *S. suis*, which exacerbates the challenges posed by the disease [10–13].

In swine, *S. suis* is naturally considered a commensal microorganism, as it commonly resides in the mucosal lining of the upper respiratory tract, particularly the tonsils. However, to establish invasive infection, *S. suis* should be able to enter the circulatory system and disseminate throughout the body fluids of the host, resulting in septicemia and/or meningitis [1, 10, 14, 15]. To achieve this, *S. suis* utilizes a range of virulence factors that enable it to thrive and proliferate within the host tissues and body fluids during the infection. These virulence factors play critical roles in penetrating the mucosal layer of the respiratory tract, adhering to the epithelial cells and extracellular matrix (ECM), evading the host's immune responses, and withstanding the effects of antibiotics which contribute to the development of antimicrobial resistance traits [10, 14–17]. Although several virulence factors of *S. suis* have been identified, the full spectrum of mechanisms it employs to cause disease in swine remains unclear. Understanding these factors is further complicated by the variability in infection outcomes among different strains, which can range from severe systemic disease to asymptomatic colonization. This variability suggests that some strains lack the essential virulence factors required to cause invasive infections and are therefore classified as non-virulent [1, 10, 14, 15].

One notable virulence factor of *S. suis* should have as a host-adapted pathogen is the phase-variable genes. These genes can undergo a process known as phase variation, a reversible and high-frequency change in expression levels. This allows pathogenic bacteria to switch genes on and off or to modulate their expression without altering the DNA sequence of the gene itself, thus considered an epigenetic mechanism [18–20]. An example of phase-variable genes found in the genome of *S. suis*, and many other host-adapted bacterial pathogens, is phase-variable methyltransferases (both Type I and Type III), which are associated with restriction-modification (RM) systems [5, 18, 19]. The phase variations of Type I and Type III methyltransferase are distinctive to each other and involve different molecular mechanisms. In Type I methyltransferase, as characterized in *Streptococcus pneumoniae* and *S. suis*, phase variation occurs through the rearrangement of DNA recognition domains encoded by the *hsdS* genes. This rearrangement is mediated by homologous recombination, facilitated by the presence of inverted repeat (IR) sequences within the *hsdS* phase variants. Such recombination events alter the DNA target specificity of the methyltransferase, enabling the bacterium to modify its DNA methylation profile [1, 5, 21]. In contrast, Type III methyltransferase varies in the number of simple sequence repeats (SSRs), which results in biphasic on-off switching of the methyltransferase expression [5, 18, 20]. Studies on other host-adapted pathogens, such as *S. pneumoniae* have shown that the phase variation of these methyltransferases gives rise to differential genome-wide methylation patterns, which in turn regulate the expression of multiple genes, especially those associated with virulence, including *dnaK*, *psaABC*, *dexB*, *luxS*, as well as enzymes for capsule biosynthesis, sucrose regulator, and colony opacity. This regulatory mechanism is referred to as phasevarion (phase-variable regulon), allowing bacterial pathogens to fine-tune gene expression for adaptation and survival [1, 5, 18, 20, 21].

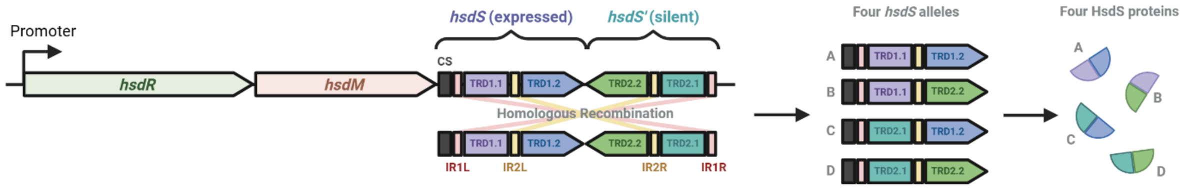

Figure 1 Phase variation of Type I methyltransferase system in *S. suis* through homologous recombination between *hsdS* and *hsdS'*. **(left)** The expression system of Type I methyltransferase in *S. suis*, consisting of promoter, *hsdR* (*SsuRS06440*), *hsdM* (*SsuRS06435*), *hsdS* (*SsuRS06425*), and *hsdS'* (*SsuRS06430*). CS: conserve sequence, IR(1/2)(L/R): inverted repeat (1/2) (left/right), TRD: target recognition domain. **(middle)** Four different *hsdS* phase variants as the result of HR between *hsdS* and *hsdS'*, denoted as phase-variant A, B, C, and D. **(right)** Four different *HsdS* proteins with different combinations of TRDs, denoted as protein A, B, C, and D. **Created by Biorender®.**

This research is carried out in accordance with the general research objective of the Host-Microbe Interactomics chair group, Wageningen University and Research, which is to investigate conditionally essential genes required for *S. suis* survival, growth, and establishment of infection in the host body fluids. Specifically, this research is studying one of the phase-variable Type I methyltransferases in *S. suis* that has never been studied previously, consisting of a gene cluster that includes *SsuRS06440* (*hsdR*), *SsuRS06435* (*hsdM*), *SsuRS06425* (*hsdS*), and *SsuRS06430* (*hsdS'*). The actively expressed *hsdR*, *hsdM*, and *hsdS* are regulated by a common promoter, located upstream of *hsdR*. The main focus of this study, *hsdS* gene, contains two variable regions encoding the target recognition domains (TRDs) of the SsuRS06425 protein and shares inverted repeat sequences with its truncated and inverted duplicate, *hsdS'*, allowing homologous recombination, resulting in four different recombinant phase variants (A, B, C, and D) – **Figure 1**. In general, the length of the *hsdS* gene is approximately 1.5 kb to 2.5 kb, while the *hsdS-hsdS'* region spans approximately 3.0 to 4 kb. However, even though *hsdS'* encodes alternative TRD phase variants, it lacks key elements for expression, including the canonical 5'-end sequence of the *hsdS* gene, ribosomal binding sites, and promoters. Therefore, this research specifically aims to investigate the frequency distribution of *SsuRS06425* (*hsdS*) recombinant phase variants and their association with the virulence and pathogenicity level of *S. suis* isolates from healthy and infected swine tonsils through an integrated combination of Nanopore sequencing and bioinformatics analysis. The findings will shed light on *S. suis* genetic variability and its possible association with virulence, potentially leading to new diagnostic and therapeutic approaches for infections in swine and humans. Additionally, understanding this phase-variable Type I methyltransferase could pave the way for future studies on *S. suis* adaptation and evolution, the emergence of new AMR strains, also their interactions with the host.

1.2. Research Questions

According to the research background and objectives that have been mentioned previously, the research questions can be formulated as follows.

1. What is the frequency distribution profile of *SsuRS06425* (*hsdS*) phase variants in the phase-variable Type I methyltransferase system of *S. suis* isolates from diseased and healthy swine tonsils?
2. Which phase variant of *SsuRS06425* (*hsdS*) is predominantly encountered in *S. suis* isolates from diseased swine tonsils compared to healthy swine tonsils?

2. RESULTS

2.1. Amplification of *hsdS-hsdS'* Region of The Type I Methyltransferase System from *S. suis* Isolates

To investigate the frequency distribution of *hsdS* phase variants within the phase-variable Type I methyltransferase system of *S. suis*, the *hsdS-hsdS'* region was targeted for amplification and subsequent Nanopore sequencing. This research builds on prior research in which DNA samples, potentially containing the *S. suis* genome, were successfully isolated from tonsil swabs of healthy and diseased swine across various farms in Europe. The amplification of the *hsdS-hsdS'* region was performed by conventional polymerase chain reaction (PCR) to assess its structural variability and prepare the amplicons for Nanopore sequencing. Agarose gel electrophoresis was utilized to evaluate the success of the amplification by providing a visual confirmation of PCR products. Given the structural complexity and possible genetic rearrangements between *hsdS* and *hsdS'*, multiple primer sets were designed to ensure comprehensive amplification coverage.

Moreover, to verify the reliability of the PCR results, negative (-) and positive (+) controls were included in the reactions. The negative control, which contained no DNA template, showed no visible bands, confirming the absence of contamination. This ensured that the observed amplification results were solely due to the presence of the target DNA and not an artifact of reagent contamination [22]. The positive control (+) is the PCR reaction to amplify the 16S rRNA gene from a sample using the specific primer set. The positive control showed a single, high-intensity band at approximately 1.5 kb, confirming the successful amplification of the 16S rRNA gene. This further supports that the PCR conditions were optimal and that the variability observed in the *hsdS-hsdS'* amplification was due to sample-specific factors rather than experimental inconsistencies [23–25].

Initial amplification attempts were carried out using primer set 1 (ps1), designed under the assumption that the *hsdS* gene maintained the same orientation as *hsdM* as shown in **Figure 8**. Five randomly selected DNA samples from the stock collection were initially tested and the results of the amplification are shown in **Figure 2**.

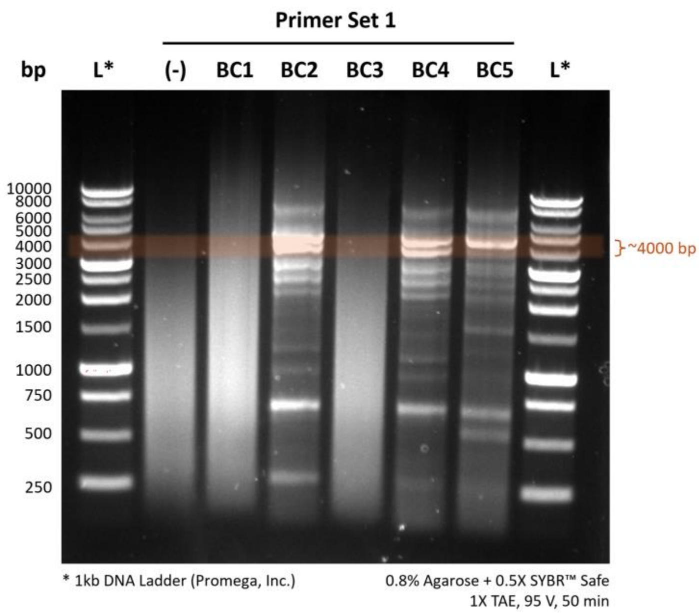

Figure 2 Amplification results of *hsdS-hsdS'* region from 5 random DNA samples, labelled with BC1-5. **L**= 1kb DNA Ladder (Promega, Inc.) as the molecular size marker; **(-)**= negative control without the DNA sample; **BC1**= sample **PD107 (159B-A7)**, which was isolated from a diseased swine at the DE1 farm; **BC2**= sample **PR13TM1 (161A-B9)**, obtained from a diseased swine at ES2 farm; **BC3**= sample **PB21TM1 (161A-H9)**, originating from a diseased swine at ES3 farm; **BC4**= sample **PQ20TM1 (161B-G5)**, which was collected from a healthy swine at ES1 farm; **BC5**= sample **JT09 (161D-E2)**, derived from a diseased swine at ES4 farm. No amplifications were observed in (-), BC1, and BC3, while multiple bands of amplicons were seen in BC2, BC4, and BC5 samples. For these three samples, relatively strong bands can be seen with the size of approximately 4000 bp, labelled with orange color. Gel composition and electrophoresis condition are indicated.

Among the tested samples, samples labelled with BC1 (PD107, from DE1 farm) and BC3 (PB21TM1, from ES3 farm) showed no amplification, with only a faint smear visible on the gel. The failure of amplification in these samples indicates that they are not suitable for further analysis. Consequently, they were excluded from the subsequent Nanopore sequencing and replaced with alternative samples to ensure a complete dataset as shown in **Table S1**. In contrast, successful amplification was observed in samples labelled with BC2 (PR12TM1, from ES2 farm), BC4 (PQ20TM1, from ES1 farm), and BC5 (JT09, from ES4 farm), where multiple bands appeared on the gel. Despite these multiple bands, a distinct and strongly visible band was detected at approximately 4,0 kb in all three samples, corresponding to the expected amplicon size of the *hsdS-hsdS'* region. This confirms that the samples are suitable for Nanopore sequencing and downstream analysis. However, despite the successful amplification of samples with BC2 and BC5, these samples were not included in Nanopore sequencing. The barcodes assigned to these samples were later reassigned for use in primer set 2 (ps2), which was applied to different samples obtained from NL1 farm, where severe disease cases were encountered.

Following this inconsistent amplification results observed with ps1, further attempts were made to amplify other DNA samples to replace BC1 and BC3 and to assign the remaining barcodes for sequencing. Amplification attempts using ps1 were performed up to BC15. However, despite multiple attempts using different DNA isolates, none of these samples showed successful amplification, and only a smear was observed on the gel. Repeated trials, including sample replacements and optimizing PCR conditions (annealing temperature, DNA concentration, PCR cycle parameter, etc), did not yield any amplicons of the expected size (results not shown). This consistent amplification failure suggested that ps1 was ineffective for the majority of the tested isolates. Therefore, to address this issue, and improve amplification efficiency, ps2 was designed to accommodate potential alternative orientations or rearrangements of the *hsdS-hsdS'* region as shown in **Figure 9**. This new primer set was used for the amplification of additional samples and to assign the remaining barcodes (up to BC30).

Before applying ps2 to a larger set of samples, it was first used to amplify a small number of DNA samples that had previously failed to be amplified with ps1. These included sample with BC3 (PR32TM1, from ES2 farm), BC12 (PR08TM1, from ES4 farm), and BC15 (PR40TM1, from ES2 farm). Additionally, a positive control (+) was included using a sample that had been amplified successfully using ps1 before the commencement of this research (PR21T1400, from ES2 farm), to confirm PCR conditions were optimal for amplification [22]. As shown in **Figure 3**, electrophoresis analysis revealed successful amplification in all tested samples, including positive control, as indicated by the presence of multiple bands. Despite some nonspecific bands, distinct bands were observed within the 3000-4000 bp range in all samples, aligning with the expected amplicon size of the *hsdS-hsdS'* region. This result confirmed that ps2 was capable of amplifying samples that had previously failed with ps1. Thus, ps2 was subsequently used to amplify other samples and assign barcodes that had not been utilized.

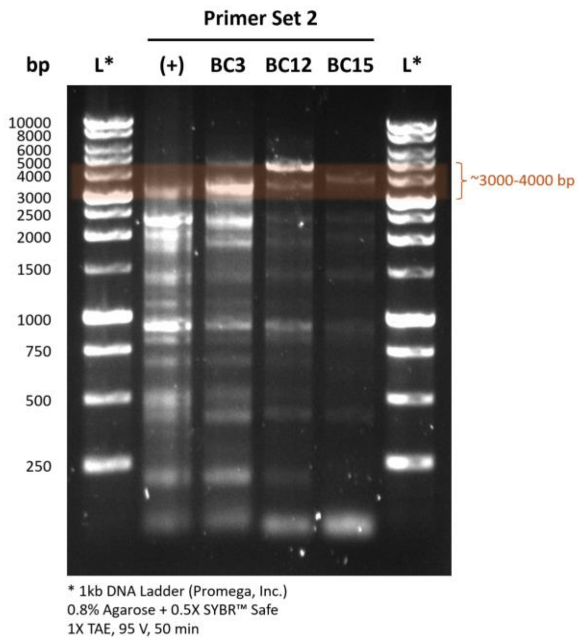

Figure 3 Amplification results of four different DNA samples for assessment of ps2 effectiveness before proceeding to a larger set of samples. **(+)**= positive control, sample **PR21T1400 (161A-B6)**, which was isolated from a diseased swine at the ES2 farm; **BC3**= sample **PR32TM1 (161A-H5)**, obtained from a healthy swine at ES2 farm; **BC12**= sample **PR08TM1 (161B-B7)**, originating from a healthy swine at ES4 farm; **BC15**= sample **PR40TM1 (161B-D3)**, which was collected from a healthy swine at ES2 farm. Multiple amplification bands can be seen in all four samples, indicating nonspecific amplification. However, there are bands sized approximately 3.0-4.0 kb, which may indicate successful amplification of *hsdS-hsdS'* region (orange color)

As previously mentioned, BC2 and BC5 were reassigned to different DNA samples obtained from the NL1 farm, where severe disease cases were encountered in swine. The complete list of DNA samples amplified with ps2 is detailed in **Table S1**. Gel electrophoresis analysis of the PCR products (**Figure 4**) revealed that the PCR produced multiple non-specific bands for all samples. Despite this, a prominent band at approximately 2.0-2.5 kb was observed in all samples, which likely corresponds to the *hsdS* gene alone, without its inverted duplicate, *hsdS'* or with a part of *hsdS'*. Additionally, all samples exhibited bands within the 3.0-4.0 kb range, which is consistent with the expected size of the full *hsdS-hsdS'* region. However, the intensity of the bands varied between each sample. In most samples, the bands within this size range appeared faint, but they were still visible upon careful examination. Despite their lower intensity, the presence of these faint bands is significant, as they may indicate successful amplification of the *hsdS-hsdS'* region, which remains crucial for further sequencing and analysis. In contrast, some samples with BC1, BC2, BC5, BC6, BC7, BC9, BC10, BC17, BC27, BC28, and BC30 exhibited stronger, more clearly visible bands in the 3.0-4.0 kb range, suggesting more efficient amplification in these isolates. Interestingly, the sample with BC5 and BC11 showed additional bands between 5.0-6.0 kb, which are larger than the expected size of the *hsdS-hsdS'* region.

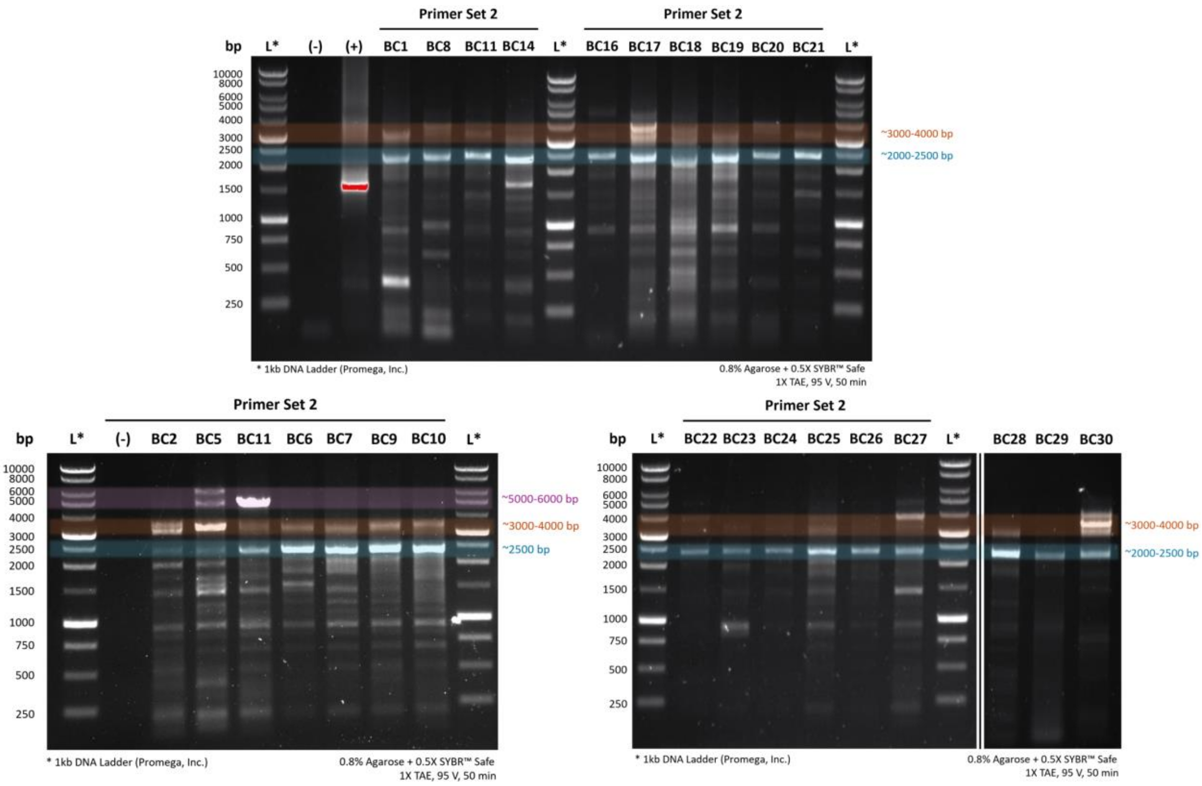

Figure 4 Electrophoresis results of PCR products of multiple DNA samples amplified using ps2. **L=** 1kb DNA Ladder (Promega, Inc.). **(-)=** negative control, included on the bottom left and top gels, no amplification was observed. **(+)=** positive control, the amplification reaction of 16S rRNA gene from one sample, included on top gel, showed bands sized approximately 1.5 kb, matching the size of the 16S rRNA gene. **BC1-30=** barcode labels, indicating specific DNA samples as listed in **Table S1**. Multiple bands were observed in all samples. A 2000-2500 bp band was consistently present across all samples **(blue)**. A band at the size of 3000-4000 bp was also detected in all samples, though it appeared faint in most samples, except for BC1, BC2, BC5, BC6, BC7, BC9, BC10, BC17, BC27, BC28, and BC30, where it was more prominent **(orange)**. In addition, BC5 and BC11 showed an additional larger-than-expected size at 5000-6000 bp **(pink).**

In summary, these observations indicate that ps2 successfully amplified the *hsdS-hsdS'* region in all selected samples, albeit with varying efficiency. The variability in band profile, intensity, and the presence of unexpected amplicon sizes highlight the heterogeneity of the *hsdS-hsdS'* region across isolates or could be attributed to inefficient PCR reactions. Therefore, sequencing is necessary to accurately characterize the amplified products and determine the true nature of the structural variations of the amplicons observed in the gel.

As mentioned previously, amplification of samples using ps2 predominantly produced a relatively prominent band around 2.0-2.5 kb, which may correspond to a single *hsdS* gene. Additionally, previous attempts using ps1 failed to amplify almost all tested samples. Therefore, to overcome these limitations and ensure comprehensive amplification of the *hsdS-hsdS'* region, primer set 3 (ps3) was designed to anneal outside the *hsdS-hsdS'*, flanking both genes, as can be seen in **Figure 10**. This approach was intended to amplify the *hsdS-hsdS'* region regardless of potential genetic orientation or structural rearrangements between *hsdS* and *hsdS'*. Similar to amplification using ps2, amplicons produced using ps3 were labelled with barcodes but using different numbering (BC31 to BC60). These barcodes were assigned to the same DNA samples amplified with ps2 or ps1 (e.g., BC1 and BC31 were assigned to the same sample, and so on), as listed in **Table S1**. Three samples initially selected for the assignment of BC6, BC7, and BC9 failed amplification with ps2 and were replaced. These unamplified samples were later re-amplified using ps3, assigned BC36, BC37, and BC39. Additionally, the successfully amplified ps2 samples for BC6, BC7, and BC9 were also reamplified with ps3, labelled as BC36', BC37', and BC39', to compare amplification efficiency. **Figure 5** presents the results of all amplifications using ps3.

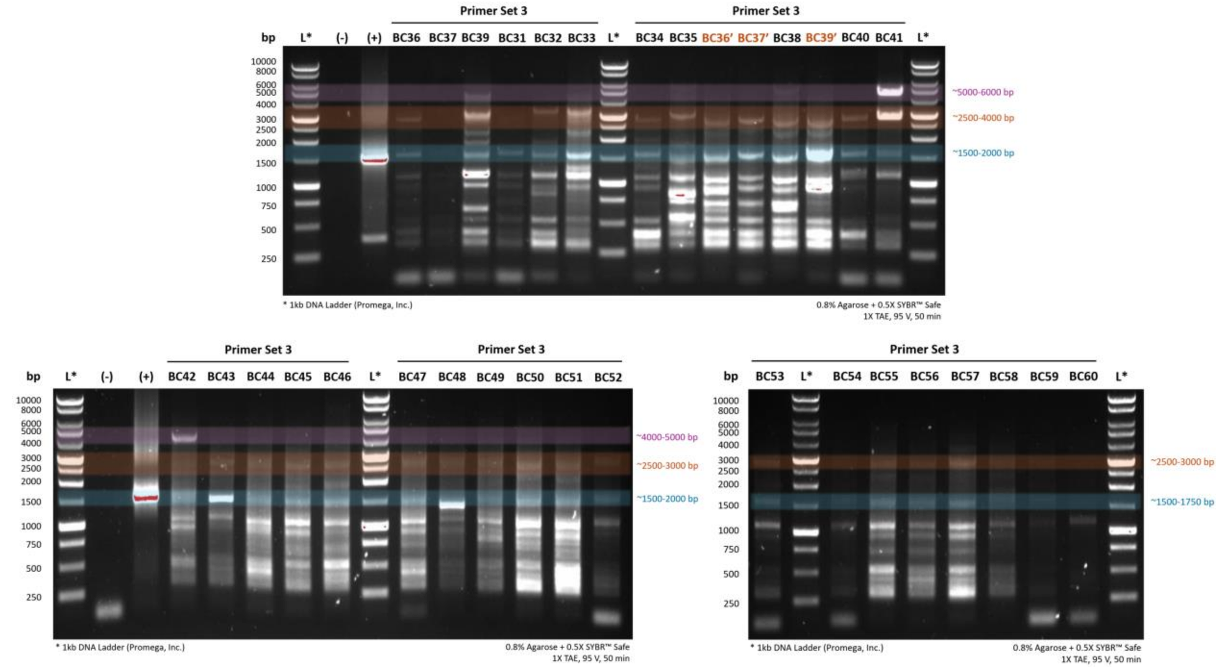

Figure 5 Electrophoresis results of PCR products of multiple DNA samples amplified using ps3. **L=** 1kb DNA Ladder (Promega, Inc.). **(-)=** negative control, included on the top and bottom left gels, no amplification was observed. **(+)=** positive control, the amplification reaction of 16S rRNA gene from one sample, included on top and bottom left gels, showed bands sized approximately 1.5 kb, matching the size of the 16S rRNA gene. **BC31-60=** barcode labels, indicating specific DNA samples as listed in **Table S1**. **BC36', 37', and 39' (orange)**= barcode labels, indicating the samples that can also be amplified using ps2. Multiple bands were observed in all samples. A 1500-2000 bp band was consistently present across all samples, where the strongest intensity can be seen in BC33, BC36', BC37', BC38, BC39, BC43, and BC48 **(blue)**. A band at the size of 2500-4000 bp was also detected in all samples, though it appeared faint in most samples, except for BC36, BC39, BC32, BC33, BC35, BC36', BC37', BC38, BC39', BC40, BC41, where it was more prominent **(orange)**. In addition, BC33, BC35, BC39, BC41, and BC42 showed an additional larger-than-expected size at 4000-6000 bp **(pink).**

PCR amplification using ps3 resulted in multiple bands across all tested samples, suggesting the production of amplicons of varying sizes. A band between 1.5 and 2.0 kb was observed in all samples with varying intensities. This band likely represents the amplification of the *hsdS* gene alone without *hsdS'*. However, the size of this amplicon was smaller than the corresponding band produced from the amplification with ps2, which sized around 2.0-2.5 kb, suggesting a different size covered by the ps3 or possible off-target amplification. Additionally, upon careful examination, a faint band between 2.5 and 4.0 kb was detected in all samples, potentially representing the full *hsdS-hsdS'* region. Although the intensity of this band varied, its consistent presence across all samples may indicate successful amplification of the target region, warranting further investigation through Nanopore sequencing analysis. Furthermore, a larger-than-expected band, which sized around 4.0-6.0 kb was observed in several samples, including BC33, BC35, BC39, BC41, and BC42.

Another key observation from this experiment was the successful amplification of previously unamplified samples from the ps2 reactions, as seen in BC36, BC37, and BC39. In particular, BC37 exhibited extremely faint bands, barely visible under standard visualization. However, upon very careful examination, the bands were able to be detected, confirming the presence of amplified products. Despite the faint band intensity, this band remains significant and will be included in further sequencing and analysis. Conversely, the three samples that successfully amplified with ps2 (BC36’, BC37’, and BC39’) showed relatively prominent multiple bands when amplified using ps3, further supporting the robustness of amplification with this primer set.

Following the completion of all PCR amplifications, the resulting PCR products were subjected to cleanup procedures for excess primers, nucleotides, buffers, and potential impurities. The DNA concentration of each purified amplification product was then measured using Qubit™ dsDNA Broad Range Assay Kit. The detailed DNA concentration results for each sample are provided in **Table S1**. In general, the Qubit measurements indicate differences in DNA concentrations among the samples, with values ranging from the lowest 10.6 $^{ng}/_{\mu L}$ (BC37, ps3) to the highest 222 $^{ng}/_{\mu L}$ (BC5, ps2). The Qubit measurement results were consistent with the bands intensity for each sample observed from the electrophoresis result, where the lower DNA concentration showed fainter bands and vice versa. After quantification, the PCR products were compiled and pooled for library preparation, ensuring an equimolar representation of each sample. The prepared library was then processed for Nanopore sequencing. The raw sequencing data generated from the Nanopore sequencing platform were subsequently analyzed using bioinformatics pipelines developed in this study. These pipelines were designed to efficiently process, filter, and classify sequencing reads, allowing for accurate identification of *hsdS* phase variant variants and their distribution among the tested samples.

2.2. Bioinformatics Pipeline Development: Nanopore Sequencing Results Processing and Analysis

The Nanopore sequencing generated raw output in the form of a POD5 file, which was subsequently processed using a basecalling Dorado software from Oxford Nanopore tehcnologiesto convert the raw signal data into a FASTQ file containing nucleotide sequences (sequencing reads) and their associated quality scores. The first bioinformatics pipeline developed for this study was designed to extract sequence IDs and their corresponding sequencing read from the FASTQ file and compile them into a FASTA file. Additionally, the pipeline generated a reverse complement of each read, stored separately in another FASTA file, to facilitate the following analysis, the primer screening. Primer screening is the main function of this pipeline to screen those reads that bring both forward and reverse primers of each primer set (ps1, ps2, or ps3) within the 200-nucleotide threshold from both ends of each read. **Table 1** summarizes the read count and distribution following primer screening for each primer set.

Table 1 Read count of the sequences with single and multiple forward-reverse pairs for each primer set and the discarded reads that do not bring a complete pair of forward-reverse primers. The percentage for each read count is also indicated.

Read Count	Primer Set		
	ps1	**ps2**	**ps3**
Single forward-reverse pairs	204,379 (0.93%)	1,388,953 (6.29%)	12,313,394 (55.81%)
Multiple forward-reverse pairs	2,6842 (0.12%)	140,125 (0.64%)	1,328,429 (6.02%)

A total of 22,062,880 reads were processed, of which 30.19% (6,660,758 reads) were discarded due to the absence of a complete forward-reverse primer pair within the predefined threshold. The remaining 69.81% (15,402,122 reads) contained at least one forward-reverse primer pair and were retained. Among the retained reads, ps3 yielded the highest number of reads, with 12,313,394 reads (55.81%) containing a single forward-reverse pair and an additional 1,328,429 reads (6.02%) containing multiple forward-reverse pairs within a single read. ps2 contributed 6.29% (1,388,953 reads) with a single primer pair and 0.64% (140,125 reads) with multiple primer pairs. ps1 exhibited the lowest read retention, with only 204,379 reads (0.93%) containing a single forward-reverse pair and 26,842 reads (0.12%) containing multiple primer pairs.

Following the primer screening, the analysis proceeded using only the reads that contained a single forward-reverse primer pair for each primer set. These reads were classified according to their similarity to the sequences within a custom *S. suis* genome database using BLASTn. The BLASTn search was executed using bash commands, ensuring efficient processing of large sequence datasets. Reads that achieved ≥80% identity to the *S. suis* reference genomes in the BLASTn output file were isolated and extracted into a new FASTA file by **Script S3**, allowing further downstream analysis focused on *S. suis*-associated reads. **Table 2** provides a summary of the BLASTn classification results, detailing the number of reads from each primer set that exhibited a ≥80% identity match to the reference *S. suis* genomes. Additionally, the table also lists the number of reads that did not produce a significant match. This classification offers insight into the proportion of reads confidently associated with *S. suis* and those requiring further investigation.

Table 2 Read count and their proportion that exhibited ≥80% identity to the *S. suis* reference genome, alongside reads that were discarded due to lower %identity. Proportions were calculated based on the total number of reads present in the input FASTA file for each primer set.

Read Count	**Primer Set**		
	ps1	**ps2**	**ps3**
Input FASTA file	**204,379**	**1,388,953**	**12,313,394**
Reads match *S. suis* (≥80%)	2,062 (1.01%)	4,692 (0.34%)	73,742 (0.60%)
Discarded reads	202,317 (98.99%)	1,384,261 (99.66%)	12,239,652 (99.40%)

Table 2 highlights the substantial reduction in read count for further analysis. Out of the total reads processed from each primer set, only a small fraction met the ≥80% identity threshold required to be classified as *S. suis*-associated reads, while the vast majority were discarded due to low read similarity or the absence of a significant match. For ps1, only 1.01% of reads were retained for downstream analysis while 98.99% of reads were discarded. Similarly, ps2 resulted in 0.34% retained reads, with 99.66% were removed. As for ps3, 0.60% of reads were classified as *S. suis*, but an overwhelming 99.40% of reads were discarded. This drastic reduction in read count emphasizes the high proportion of reads that failed to align significantly with the *S. suis* reference genomes, suggesting that a large fraction of the amplified reads may represent non-*S. suis* DNA, chimeric reads, or low-quality sequencing artifacts [26–29].

Following the initial BLASTn classification against the *S. suis* genome database, a subsequent analysis was performed to further refine the dataset by identifying reads that specifically matched the phase variants of *hsdS*. This additional filtering step aimed to retain only reads corresponding to *hsdS* phase variants while discarding non-relevant *S. suis* reads. To improve alignment accuracy and classification efficiency, the reads matching *S. suis* for each primer set were first categorized by size using **Script S4**. The script divided the reads into five groups based on length: reads <500 nucleotides – these reads were considered too short for reliable classification and were considered as junk, reads with the size 500-1000 nucleotides were expected to represent a partial *hsdS* gene, reads with the size 1000-2000 nucleotides were ikely to encompass a whole *hsdS* gene, reads with the size 2000-4000 nucleotides may contain a full *hsdS* gene along with a portion or the whole *hsdS'*, and reads >4000 nucleotides potentially include *hsdS-hsdS'* region, and additional regions, including possible chimeric reads. In general, the result of size classifications can be seen in **Table 3**.

Table 3 Read count for each size-classified FASTA file for each primer set. Total read present in the input FASTA file that matches with *S. suis* reference genome is also indicated for each primer set

Read Count	**Primer Set**		
	ps1	**ps2**	**ps3**
Input FASTA file	**2,062**	**4,692**	**73,742**
<500 nucleotides	262	2,855	70,265
500-1000 nucleotides	925	1,148	2,116
1000-2000 nucleotides	463	263	1,048
2000-4000 nucleotides	366	389	255
>4000 nucleotides	44	37	58

A substantial reduction in read count was observed notably in ps3, where a majority of reads (95.36%) were smaller than 500 nucleotides and were consequently considered unreliable for further analysis. This resulted in a drastic decrease in the number of reads available for subsequent *hsdS* phase variant classification. For ps2, while the read loss was not as significant as in ps3, 60.85% of reads fell below the 500-nucleotide size threshold and were excluded for analysis. In contrast, ps1 exhibited the least read loss, with only 12.71% of reads categorized as smaller than 500 nucleotides. These findings highlight the importance of size filtering in refining sequencing data, ensuring that only reads of sufficient length for reliable phase variant classification are retained for further investigation.

Table 4 Read count summary of size-classified *S. suis*-associated reads matching each *hsdS* phase variant or *hsdS-hsdS'* phase variant pairs across different primer sets. The read count from the input FASTA file and the total matching read are also indicated. Across all size-classified read groups, phase variant B (for reads sized 500-1000 and 1000-2000 nt) or phase variant pair B_C (for reads sized 2000-4000 and larger than 4000) exhibited the highest read counts, indicating their predominant presence in the dataset. nt = nucleotide.

Read Count	Primer Set		
	ps1	ps2	ps3
Read 500-1000 nt			
Input FASTA file	925	1,148	2,116
Total matching read	2	28	33
Match with phase variant A	0	0	0
Match with phase variant B	2	22	31
Match with phase variant C	0	0	0
Match with phase variant D	0	6	2
Read 1000-2000 nt			
Input FASTA file	463	263	1,048
Total matching read	75	111	34
Match with phase variant A	0	0	0
Match with phase variant B	73	107	34
Match with phase variant C	0	4	0
Match with phase variant D	2	0	0
Read 2000-4000 nt			
Input FASTA file	366	389	255
Total matching read	288	297	26
Match with phase variant A_D	2	2	2
Match with phase variant B_C	274	290	14
Match with phase variant C_B	12	5	8
Match with phase variant D_A	0	0	2
Read >4000 nt			
Input FASTA file	44	37	58
Total matching read	36	29	34
Match with phase variant A_D	0	0	0
Match with phase variant B_C	28	25	28
Match with phase variant C_B	8	4	6
Match with phase variant D_A	0	0	0

After size classification, reads in FASTA files for reads sized 500-1000 nucleotides and 1000-2000 nucleotides were aligned using BLATSn to the *hsdS* phase variant database, comprising four *hsdS* phase variants: A, B, C, and D. On the other hand, reads larger than 2000 nucleotides, such as those in 2000-4000 and larger than 4000 nucleotides FASTA files were aligned using BLASTn to the *hsdS-hsdS'* phase variant pair database, comprising of four possible *hsdS-hsdS'* phase variant pair: A with inverted D, B with inverted C, C with inverted B, and D with inverted A. The summary of the BLASTn results can be seen in **Table 4**. Then, **Script S5** was run to pick all the reads listed in the BLASTn result into new FASTA files.

In general, the BLASTn classification revealed a dominance of phase variant B and the B_C phase variant pair, while phase variants A and D were rarely detected. Within the group of reads with the size of 500-1000 nucleotides, only a small fraction of reads matched any *hsdS* phase variant, with phase variant B being the most frequently identified, particularly in ps2 (22 reads) and ps3 (31 reads), while no reads matched phase variants A or C. In the group of 1000-2000 nucleotide size, a higher number of reads aligned with *hsdS* phase variants, with phase variant B remaining predominant across all primer sets. For reads between 2000-4000 nucleotides, a substantial proportion matched *hsdS-hsdS'* phase variant pairs, with the B_C combination being the most frequently identified in ps1 (274 reads) and ps2 (290 reads), while ps3 contained significantly fewer matches. The C_B phase variant pair was detected in lower numbers, and only ps3 contained any matches to the D_A phase variant pair. In reads larger than 4000 nucleotides, the B_C phase variant pair remained the most common, while other phase variant combinations were detected at very low frequencies or not at all. Overall, ps3 consistently had fewer matching reads compared to ps1 and ps2.

Table 5 Barcode demultiplexing results and classification of isolate sources from healthy and symptomatic pigs. read counts and their corresponding percentages for each barcode across three primer sets are indicated. BC= barcode. BC1-30 were assigned to the reads amplified by ps1 and ps2, while BC31-BC60 were used for ps3. The total read and percentage of reads matching each barcode for each primer set within each isolate source group were also indicated

Read Count	Primer Set			Read Count	Primer Set		
	ps1	ps2	ps3		ps1	ps2	ps3
Healthy BC				**Symptomatic BC**			
BC1 or BC31	0 (0.00%)	2 (2.70%)	0 (0.00%)	BC2 or BC32	10 (4.37%)	6 (2.45%)	23 (20.00%)
BC3 or BC33	2 (2.70%)	0 (0.00%)	4 (16.67%)	BC5 or BC35	141 (61.57%)	174 (71.02%)	8 (6.96%)
BC4 or BC34	0 (0.00%)	0 (0.00%)	0 (0.00%)	BC11 or BC41	70 (30.57%)	59 (24.00%)	0 (0.00%)
BC6 or BC36	0 (0.00%)	0 (0.00%)	4 (16.67%)	BC16 or BC46	0 (0.00%)	0 (0.00%)	0 (0.00%)
BC7 or BC37	0 (0.00%)	0 (0.00%)	0 (0.00%)	BC17 or BC47	0 (0.00%)	0 (0.00%)	16 (13.91%)
BC8 or BC38	0 (0.00%)	0 (0.00%)	12 (50.00%)	BC18 or BC48	0 (0.00%)	0 (0.00%)	4 (3.48%)
BC9 or BC39	0 (0.00%)	0 (0.00%)	0 (0.00%)	BC19 or BC49	0 (0.00%)	0 (0.00%)	4 (3.48%)
BC10 or BC40	0 (0.00%)	2 (2.70%)	0 (0.00%)	BC20 or BC50	0 (0.00%)	0 (0.00%)	0 (0.00%)
BC12 or BC42	66 (89.19%)	70 (94.59%)	0 (0.00%)	BC21 or BC51	6 (2.62%)	6 (2.45%)	46 (40.00%)
BC13 or BC43	4 (5.41%)	0 (0.00%)	0 (0.00%)	BC22 or BC52	2 (0.87%)	0 (0.00%)	2 (1.74%)
BC14 or BC44	0 (0.00%)	0 (0.00%)	2 (8.33%)	BC23 or BC53	0 (0.00%)	0 (0.00%)	2 (1.74%)
BC15 or BC45	2 (2.70%)	0 (0.00%)	2 (8.33%)	BC24 or BC54	0 (0.00%)	0 (0.00%)	0 (0.00%)
BC28 or BC58	0 (0.00%)	0 (0.00%)	0 (0.00%)	BC25 or BC55	0 (0.00%)	0 (0.00%)	0 (0.00%)
BC29 or BC59	0 (0.00%)	0 (0.00%)	0 (0.00%)	BC26 or BC56	0 (0.00%)	0 (0.00%)	6 (5.22%)
BC30 or BC60	0 (0.00%)	0 (0.00%)	0 (0.00%)	BC27 or BC57	0 (0.00%)	0 (0.00%)	4 (3.48%)
Total Reads	**74**	**74**	**24**	**Total Reads**	**229**	**245**	**115**
Healthy Percentage	**24.42%**	**23.20%**	**17.27%**	**Symptomatic Percentage**	**75.58%**	**76.80%**	**82.73%**

Following the *hsdS* phase variant classification, the pipeline proceeded with a two-step barcode demultiplexing process to assign reads to their respective barcode labels. This process was performed using **Script S6** (first step) and **Script S7** (second step) and was guided by the forward primer of each primer set to ensure accuracy in barcode demultiplexing. The first step of demultiplexing involved identifying reads that contained an exact match to the reference barcode reads. Reads that successfully matched were assigned to their respective barcodes, while reads that failed to match exactly were retained in a separate file for further processing. In the second step, the discarded reads from the first step were re-screened using a barcode-matching approach that allowed up to two mismatches, increasing the likelihood of recovering reads with minor sequencing errors in their barcode regions. Barcodes BC1–30 were used for ps1 and ps2, while BC31–60 were used for ps3, ensuring proper separation of reads based on their corresponding primer sets. Once the barcode demultiplexing was completed, the demultiplexed reads were classified based on their isolate source, distinguishing between reads obtained from healthy (H) pigs and those from diseased/symptomatic (S) pigs, according to the barcode labels they carried, performed by **Script S8**. This classification allowed further analysis of phase variant distribution patterns in relation to host health status, enabling comparisons between healthy and diseased isolates. The summary of the classification can be seen in **Table 5.**

The results of barcode demultiplexing and classification of reads by isolate source show a higher proportion of reads originating from symptomatic pigs (S) compared to healthy pigs (H) across all primer sets. Overall, the percentage of reads assigned to symptomatic pigs was 75.58% for ps1, 76.80% for ps2, and 82.73% for ps3, whereas reads from healthy pigs accounted for 24.42% (ps1), 23.20% (ps2), and 17.27% (ps3). For reads from healthy isolates, only a few barcode labels showed successful amplification, with BC12 (ps1 and ps2) exhibiting the highest read count, accounting for 89.19% and 94.59% of healthy reads, respectively. Other barcodes assigned to healthy samples had minimal or no detectable reads. In contrast, reads from symptomatic isolates were more widely distributed across multiple barcodes, particularly BC2, BC5, and BC11, which were consistently detected in ps1 and ps2, as well as BC32, BC48, and BC51 for ps3. The highest read count was observed for BC5 (symptomatic isolate) in ps1 (61.57%) and ps2 (71.02%), while BC51 showed a high percentage in ps3 (40.00%). Several other barcodes also showed moderate representation, such as BC21, BC41, and BC51, though their read counts varied between primer sets. The greater representation of reads from symptomatic pigs across all primer sets suggests that there might be an influence of a higher abundance of *S. suis* genome present in the sample isolated from symptomatic pigs that could give rise to the high prevalence of *hsdS* phase variants, especially for the sample with BC2, BC5, and BC11 since these samples were isolated from the farm NL1, where severe disease cases were encountered [30–33].

After classifying reads according to their isolate source (healthy vs. symptomatic), the next step involved determining the distribution of *hsdS* phase variants within each isolate group across all primer sets. This classification aimed to reveal potential differences in *hsdS* phase variant prevalence between isolates from diseased and healthy swine, providing insights into the role of *hsdS* phase variants in *S. suis* virulence. Beforehand, reads were first trimmed at both ends using **Script S9** to remove any non-target regions beyond the primer-binding sites. Once trimmed, reads were classified into each *hsdS* phase variant according to the BLASTn results that have been generated previously using **Script S10** for *hsdS* phase variant classification. This classification allowed a direct comparison of *hsdS* phase variant distribution between healthy and symptomatic isolates, highlighting potential associations between specific *hsdS* phase variants and disease establishment.

Following the classification of reads into *hsdS* phase variants, **Script S11** and **Script S12** were executed to sort all data into a single comprehensive CSV file, facilitating streamlined analysis and visualization of phase variant distribution patterns, as presented in **Table S2**. To summarize the distribution of *hsdS* phase variants across healthy and symptomatic isolates and different primer sets, **Script S13** was executed to compile the data into **Table S3** and **Table S4**, respectively. Additionally, to assess the general phase variant preference of *S. suis*, **Script S14** was employed to generate a coalesced dataset, summarizing overall phase variant frequencies across all isolates. This final dataset, presented in **Table S5** and **Figure 6**, offers broader insights into *hsdS* phase variant selection trends within the bacterial population, potentially reflecting evolutionary pressures or functional advantages associated with specific phase variants. According to **Figure 6**, the analysis of *hsdS* phase variant distribution revealed a strong predominance of phase variant B which accounted for 95.53% of all reads analyzed. In contrast, phase variant C was detected in only 2.63% of reads, while phase variant D and phase variant A were present at even lower frequencies, 1.31% and 0.53%, respectively. The skewed distribution indicates a possible selective advantage or evolutionary pressure favoring phase variant B, regardless of the virulence of the *S. suis*.

Before conducting statistical analysis, a Shapiro-Wilk normality test (**Script S15**) was performed on the data presented in **Table S3** to evaluate whether the distribution of *hsdS* phase variants within healthy and symptomatic isolates follows a normal distribution [34–36]. For the healthy group, the Shapiro-Wilk test yielded a W statistic of 0.6505 with a p-value of 0.0026, which is below the 0.05 significance threshold. This indicates that the phase variant distribution in healthy isolates does not follow a normal distribution. Similarly, for the symptomatic group, the test produced a W statistic of 0.6458 with a p-value of 0.0022, confirming that the phase variant distribution in symptomatic isolates is also non-normally distributed. Additionally, **Table S3** revealed that the variance of the data for phase variant B exceeds the mean, further suggesting that the dataset does not meet the assumptions required for parametric statistical tests. Given these findings, non-parametric statistical methods were deemed appropriate for further analysis to account for the non-normal distribution and high variance of *hsdS* phase variant frequencies across isolate groups [34, 36].

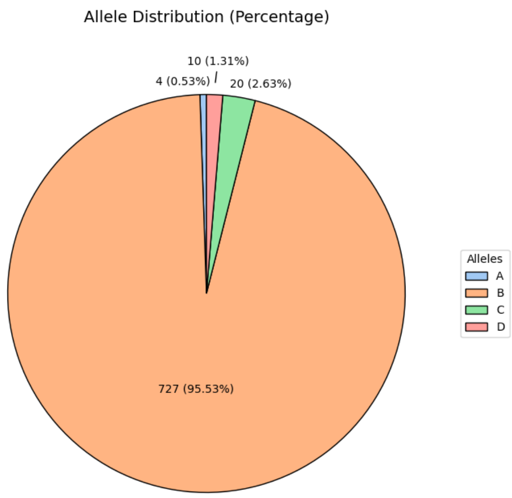

Figure 6 Pie chart illustrating the proportiona*l distribution of hsdS* phase variants from all reads analyzed in this study. The majority of reads (95.53%) were identified as phase variant B making it the most prevalent phase variant. phase variant C accounted for 2.63% of the reads, followed by phase variant D at 1.31%, while phase variant A was the least represented, constituting only 0.53% of the total reads. These findings suggest a strong preference of S. suis for phase variant B regardless of its virulence status.

The first non-parametric statistical analysis that was performed is the chi-square (χ^2) test for independence. This test was performed using **Script S16** to evaluate whether the distribution of *hsdS* phase variants depends on the host health status [37]. This test assesses whether the observed frequencies of each phase variant (A, B, C, and D) significantly deviate from the expected frequencies under the assumption that phase variant occurrence is independent of health status as seen in **Table S6A**. The null hypothesis (H_o) assumes that there is no association between phase variant type and health status, meaning the proportion of each phase variant should be similar in both healthy and symptomatic groups. Conversely, if a significant difference is observed, it would suggest that certain phase variants may be disproportionately associated with either health or disease. The χ^2 test yielded a test statistic of 2.977 with 3 degrees of freedom, resulting in a p-value of 0.395. Since this p-value is greater than the conventional significance threshold (0.05), this indicates that the *hsdS* phase variant distribution is independent of host health status. Examining the observed frequencies, phase variant B was the most prevalent in both groups, with 164 occurrences in healthy individuals and 563 in symptomatic individuals. Phase variants C and D were present at lower frequencies, while phase variant A was completely absent in healthy individuals but observed four times in symptomatic individuals. The expected frequencies, calculated under the assumption of no association, closely matched the observed counts, further supporting the conclusion that deviations were minor and not statistically significant. Overall, the results suggest that within this dataset, no strong evidence was found to indicate that any specific phase variant is associated with health status [37].

To improve the accuracy of the analysis, the read counts for each isolate group were normalized into percentages by calculating the ratio of each phase variant's read count relative to the total read count within its respective group. This transformation allowed for a more direct comparison of phase variant distributions between the healthy and symptomatic groups, mitigating the effects of differing sample sizes. A χ^2 test for independence was then conducted on this normalized frequency dataset to assess whether the frequency distribution of *hsdS* phase variants depends on the host health status or not [37]. The observed and expected phase variant frequencies for each group are presented in **Table S6B**. The χ^2 test yielded a test statistic of 1.223 with 3 degrees of freedom, resulting in a p-value of 0.7476. Since this p-value is substantially greater than 0.05, the results indicate that the distribution of the *hsdS* phase variants is independent of the health status of the host [37]. Examining the normalized frequencies, phase variant B was the most dominant in both groups, representing 95.35% of the reads in healthy individuals and 95.59% in symptomatic individuals. Phase variants C and D were present at much lower proportions, with phase variant C comprising 2.33% of reads in healthy individuals and 2.72% in symptomatic individuals, while phase variant D accounted for 2.33% in healthy individuals and 1.02% in symptomatic individuals. Phase variant A was absent in the healthy group (0%) but had a minimal presence in the symptomatic group (0.68%). These minor variations were insufficient to produce statistically significant differences, as reflected in the high p-value.

Further analysis was conducted using a Mann-Whitney U test. This non-parametric test was chosen because it is better suited for datasets with non-normal distributions and unequal variances, as previously identified in the Shapiro-Wilk test. Unlike the χ^2 test, which assesses overall association, the Mann-Whitney U test specifically evaluates whether the distributions of each phase variant differ significantly between the two groups [38–40]. By using normalized read counts, this approach also minimizes the potential bias introduced by differences in sample sizes between healthy and symptomatic isolates. The analysis was performed using **Script S17**, and the results are visualized in **Figure 7**.

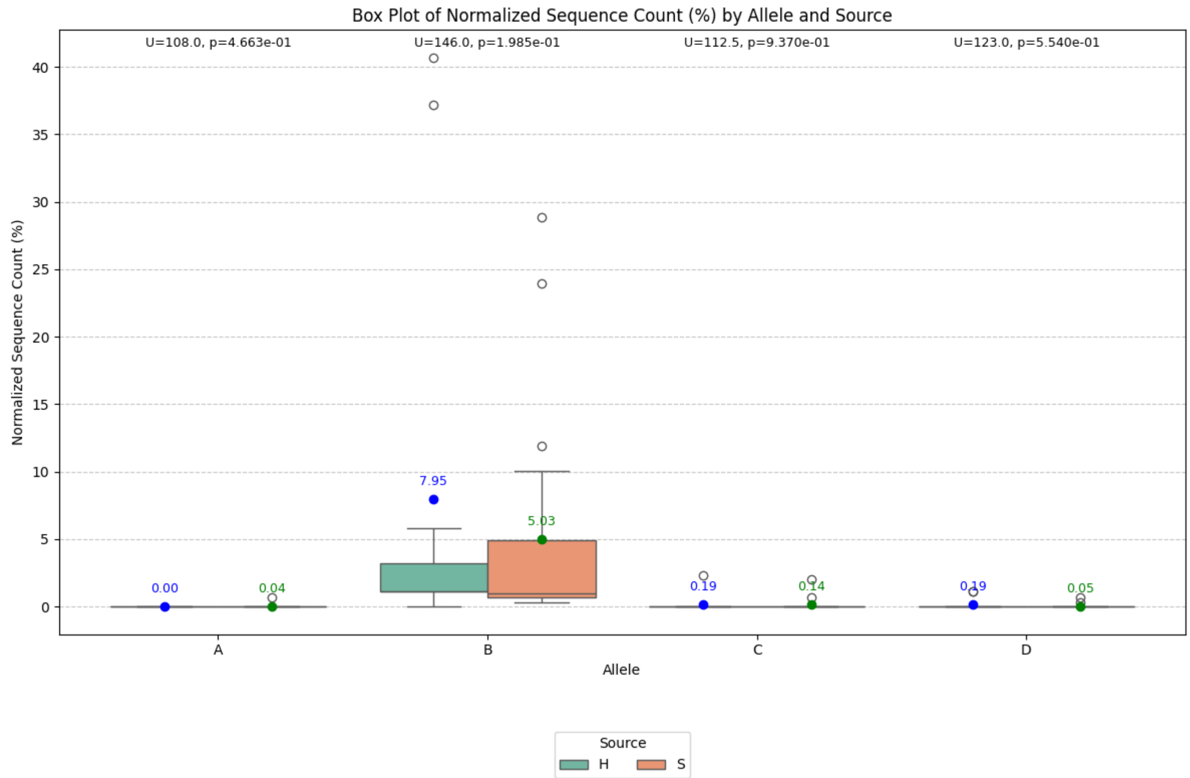

Figure 7 Box plot of the distribution of normalized read counts for each *hsdS* phase variant across two isolate source groups, H (healthy pigs) and S (symptomatic pigs). The box represents the interquartile range (IQR), while whiskers indicate the standard deviation of the read count data for each isolate source group. Outlier points (colorless dots) represent data values exceeding the expected distribution. The mean values for each group are represented by blue (H) and green (S) dots, labelled with their corresponding numerical values. Above each phase variant, the Mann-Whitney U statistic and p-value are displayed, assessing differences in phase variant distribution between healthy and symptomatic pig isolates. Notably, phase variant B is the most abundant in both groups, but its distribution does not significantly differ between healthy and symptomatic pigs, as indicated by a p-value greater than 0.05. Similarly, the distributions of phase variants A, C, and D also show no significant differences between the two groups.

The box plot illustrates the distribution of normalized read counts for each phase variant in healthy (H) and symptomatic (S) isolates, along with the results of the Mann-Whitney U test for each comparison. The U-values and corresponding p-values are displayed at the top of the plot for phase variants A, B, C, and D. The results indicate that none of the phase variants showed a statistically significant difference in distribution between the two groups, as all p-values were above 0.05. Visually, phase variant B had the highest prevalence across both groups, with some variation in its interquartile range and the presence of outliers, but the difference was not statistically significant. The distributions of phase variants C and D were relatively similar, with only slight differences in their median values. phase variant A was nearly absent in both groups, with only minor representation in the symptomatic isolates. The Mann-Whitney U test results reinforce the findings from the χ^2 analysis, indicating that no individual phase variant exhibits a significantly different distribution between healthy and symptomatic isolates. This suggests that *hsdS* phase variant frequencies are not strongly associated with health status in this dataset [38–40].

Following the Mann-Whitney U test, a Kruskal-Wallis test was conducted to further examine potential variations in *hsdS* phase variant distribution across different primer set groups (ps1, ps2, and ps3). While the previous analyses focused on comparing phase variant distributions between healthy and symptomatic isolates, this test was performed to assess whether the primer set used for sequencing influenced the observed phase variant frequencies [41, 42]. Since the dataset was previously identified as non-normally distributed with high variance, a parametric ANOVA test would not be appropriate. Instead, the Kruskal-Wallis test—a non-parametric alternative to ANOVA—was chosen, as it allows for the comparison of multiple independent groups without assuming normality [41]. The analysis was conducted using Script S18, and the results are summarized below. The Kruskal-Wallis test results did not identify any statistically significant differences in phase variant distributions across the primer sets, as all p-values were above 0.05. This suggests that the choice of primer set did not systematically influence the observed *hsdS* phase variant frequencies, reinforcing the reliability of the sequencing data.

For phase variant A, the test was not performed for the healthy group because all recorded values were either identical or absent, meaning there was no variability to analyze. In the symptomatic group, the Kruskal-Wallis test yielded an H statistic of 0.9000 and a p-value of 0.6376, indicating no significant difference in phase variant A distribution across primer sets. For phase variant B, the most prevalent phase variant in both healthy and symptomatic isolates, no significant variation was observed across primer sets. The test resulted in an H statistic of 0.4851 (healthy) and 4.2103 (symptomatic), with p-values of 0.7846 and 0.1218, respectively, suggesting that phase variant B distribution remains consistent across all primer sets. For phase variant C, the Kruskal-Wallis test produced an H statistic of 1.4000 (healthy) and 1.9000 (symptomatic), with p-values of 0.4966 and 0.3867, respectively. These values indicate that primer selection had no meaningful effect on phase variant C distribution. For phase variant D, the results followed a similar pattern, with an H statistic of 0.7700 (healthy) and 1.5667 (symptomatic), and p-values of 0.6805 and 0.4569, respectively, confirming that no significant differences in phase variant D frequencies were associated with primer selection. These results confirm that primer selection does not introduce significant bias in phase variant distribution, ensuring that any previously observed differences in phase variant frequencies are more likely to reflect true biological variation rather than methodological inconsistencies [41, 42].

3. DISCUSSION AND CONCLUSION

This study investigated the phase variant frequency distribution of *SsuRS06425* (*hsdS*), the gene encoding the DNA recognition protein of the phase-variable Type I methyltransferase system in *S. suis* across isolates from healthy and diseased swine tonsils. Using an integrated approach combining Nanopore sequencing and comprehensive bioinformatics analysis, this study aimed to assess potential differences in *hsdS* phase variant distribution between these two isolate groups and identify predominant phase variants associated with each condition.

The initial amplification of the *hsdS-hsdS'* region using primer set 1 (ps1) was largely unsuccessful, yielding smearing instead of distinct bands in most reactions, which may indicate issues during the early stage of amplification. Despite extensive optimization, amplification failures persisted, likely due to the combination of primer design limitations and structural variability within the *hsdS-hsdS'* region [5, 19, 43]. The recombination events between *hsdS* and *hsdS'* could have disrupted primer binding, while primer-related issues such as mismatched melting temperatures (T_m), secondary structure and primer dimer formation, or barcode sequence interference might have further reduced amplification efficiency. These factors could still allow amplification but at a much lower efficiency, reducing the likelihood of successful amplification and leading to inconsistent PCR outcomes [5, 43–47].

To overcome these issues, alternative primer set 2 (ps2) and 3 (ps3) were designed. ps2 accounted for potential structural rearrangements and alternative orientations of the *hsdS* gene, while ps3 flanked the *hsdS-hsdS'* region externally to ensure higher amplification success regardless of the genetic structural variability of this region [5, 44]. Both primers significantly showed improved amplification efficiency, producing visible bands in electrophoresis. However, the presence of multiple bands may suggest chimera amplicons or non-specific amplification, which could be due to the presence of inverted repeats within the region of interest, and the background DNA apart from the target region, including other parts of the *S. suis* DNA genome (sized ~1.95-2.4 Mb) [48], swine epithelial cells' genomic DNA (*Sus scrofa,* sized ~2.7 Gb) [49], and other DNA from natural microbiota of the swine tonsils [50–55]. Despite this, a distinct band corresponding to the expected amplicon size (2.5-4.0 kb) consistently appeared, which may indicate successful amplification of *hsdS-hsdS'* region. However, the intensity of this band varied between samples, possibly reflecting different amplification efficiency across samples. This could happen due to several factors, including structural variations of *hsdS-hsdS'* that could affect primer binding efficiency [1, 5, 44].

In addition, some samples exhibited larger-than-expected bands (>4.0 kb), likely due to multiple factors. One possibility is the formation of chimeric PCR products, where incomplete DNA extension and reannealing during amplification result in artificial concatemer formation. This occurs when partially synthesized DNA strands reanneal to complementary regions within the PCR cycle, acting as megaprimers instead of serving as complete templates for amplification [56, 57]. This phenomenon may be further facilitated by the presence of inverted repeat sequences between *hsdS* and *hsdS'*, which can form stable secondary structures during amplification, leading to the unintended concatenation of multiple *hsdS* and *hsdS'* fragments. Alternatively, structural variations within the *hsdS-hsdS'* region, such as insertions or duplications, could contribute to the increased amplicon size [26, 56–58]. Another potential explanation is non-specific amplification, where primers bind to unintended genomic regions due to the complexity of the environmental DNA sample [50, 52, 54].

Following the Nanopore sequencing and basecalling, the sequencing results were subjected to the primer screening using a custom pipeline. In general, 69.81% of total reads were retained, while 30.19% were discarded by the primer screening pipeline due to missing primers. One possible explanation for the absence of primer pairs in a significant fraction of reads is the presence of background DNA (has no primers) from the sample. Additionally, while Nanopore sequencing is designed to generate long reads, some may be truncated due to sequencing errors, incomplete translocation through the nanopore, or low-quality signal processing, further contributing to the loss of primer sequences in certain reads [59–63]. Despite equimolar DNA preparation, an uneven distribution of retained reads across all primer sets was observed. Specifically, ps3 exhibited the highest retention rate (55.81% for a single primer pair), whereas ps1 had the lowest (0.93% for a single primer pair). This disparity is likely due to the limited representation of ps1 sequences within the library, as only a single sample was used for ps1. Furthermore, the presence of reads containing multiple primer pairs may indicate the formation of chimeric PCR products, potentially facilitated by inverted repeat reads between *hsdS* and *hsdS'* [26, 56, 57].

Following primer screening, the retained reads underwent BLASTn classification against a custom *S. suis* genome database to identify reads with ≥80% identity to *S. suis* reference genomes. This step resulted in substantial read loss, with only 80,496 reads (0.52% of the total dataset) retained for downstream analysis. One possible explanation for this finding is that a large fraction of the sequenced DNA likely originated from non-*S. suis* sources, including swine epithelial cells and microorganisms naturally present in the tonsillar microbiome. Given the disparity in genome sizes between *S. suis* and the host genome, non-specific amplification of host DNA may have overwhelmed *S. suis*-specific reads [48–50, 52].

After BLASTn classification, the retained reads underwent size-based filtering to further refine the dataset. Reads were grouped according to length, and those smaller than 500 nucleotides were classified as junk and excluded from further analysis, as they likely covered only a fraction of the target recognition domain of the *hsdS* gene. The most substantial read loss occurred in ps3, where 95.36% of reads fell below this threshold. This high read loss in ps3 likely resulted from excessive amplification of fragmented or degraded DNA, leading to an abundance of short, incomplete reads that failed to capture the full *hsdS* gene or *hsdS-hsdS'* region [64–66]. Alternatively, it may have been caused by non-specific amplification of a shorter DNA fragment from another region of the *S. suis* genome or non-*S. suis* DNA [52, 53]. Following size filtering, an initial *hsdS* classification was performed to remove reads that did not align with any known *hsdS* phase variant or *hsdS-hsdS'* phase variant pair, further reducing the dataset. Despite passing the *S. suis* filtering step, a considerable portion of reads failed to match known *hsdS* phase variants at the 80% identity threshold, suggesting potential off-target amplification within the *S. suis* genome [52–54]. Additionally, sequencing artifacts such as low-quality reads, truncated reads, or chimeric products may have contributed to the lack of alignment. Errors in sequencing or improper strand reannealing during PCR could have resulted in incomplete or artificially rearranged reads that failed to map correctly to reference *hsdS* phase variants or *hsdS-hsdS'* phase variant pairs [57, 64, 66].

The retained reads underwent a two-step barcode demultiplexing process. First, reads with an exact barcode match were identified, followed by a secondary screening allowing up to two mismatches to recover reads with minor errors. However, Nanopore sequencing errors, such as misreading due to its relatively high error rates compared to other sequencing methods, could have altered barcode sequences, leading to misclassification or read loss [28, 59, 61, 63]. While mismatch tolerance mitigated some errors, excessive mutations caused certain barcodes to be unrepresented. Following demultiplexing, read classification by isolate source showed a higher representation from symptomatic pigs across all primer sets (ps1: 75.58%, ps2: 76.80%, ps3: 82.73%), while reads from healthy pigs accounted for 24.42%, 23.20%, and 17.27%, respectively. This suggests a higher abundance of *S. suis* genomic DNA in diseased hosts, likely due to increased bacterial loads and colonization in infected tissues, leading to greater DNA recovery from symptomatic samples [30–33].

Following barcode demultiplexing and isolate source classification, *hsdS* classification was conducted to assess the distribution of *hsdS* phase variants among isolates from healthy and symptomatic pigs. Across all primer sets and isolate groups, phase variant B was the most prevalent, while phase variants A, C, and D appeared at much lower frequencies. The dominance of phase variant B suggests a possible evolutionary preference in *S. suis*, potentially due to its role in the *hsdS*-mediated phase-variable Type I methyltransferase system. This system influences epigenetic regulation, altering gene expression patterns that may enhance cellular viability, survivability, or fitness [1, 5, 44]. Thus, phase variant B might provide an optimal regulatory function, improving bacterial growth, immune evasion, or environmental adaptability within the host, thus being favored through natural selection [72–74]. Despite phase variant B being the most frequent in both healthy and symptomatic pigs, statistical analyses were performed to determine whether *hsdS* phase variant distributions varied significantly between isolate sources (healthy vs. symptomatic) or primer sets (ps1, ps2, ps3). Chi-square (χ^2) tests, Mann-Whitney U tests, and Kruskal-Wallis tests were conducted to assess potential associations and distribution differences.

The χ^2 test found no significant association between *hsdS* phase variant distribution and host health status, suggesting no specific phase variant was disproportionately linked to disease presence [37]. The Mann-Whitney U test further supported this, showing no significant difference in phase variant prevalence between healthy and symptomatic isolates[38, 39]. Similarly, the Kruskal-Wallis test confirmed that primer choice (ps1, ps2, ps3) did not introduce amplification biases, meaning that all primer sets yielded comparable *hsdS* phase variant distributions, ensuring reliable sequencing data [41, 42]. These findings suggest that while *hsdS* phase variation occurs in *S. suis*, phase variant distribution alone does not drive virulence or disease outcomes. The high prevalence of phase variant B in both groups indicates it is not a virulence determinant but rather a functionally advantageous form of *hsdS*, essential for *S. suis* cell function and survival. This supports the idea that other genetic, environmental, or host factors play a more significant role in disease establishment than *hsdS* phase variant variation.

This study provides valuable insights into *hsdS* phase variant distribution in *S. suis* isolates from healthy and symptomatic pigs; however, further research is needed to fully understand the functional significance of *hsdS* phase variation. A key future direction is integrating functional assays to determine whether different *hsdS* phase variants influence methylation patterns, gene expression, or phenotypic traits related to virulence, immune evasion, and bacterial adaptation. While this study established phase variant frequencies, additional analyses such as RNA sequencing [75, 76], methylation profiling [77, 78], and proteomics [76, 79] could reveal changes in genome-wide expression profile which could be affected by each *hsdS* phase variant and to identify whether phase variant B's predominance confers functional advantages or contributes to *S. suis* pathogenicity. Expanding sample size and diversity will improve statistical robustness and provide a more representative view of *S. suis* populations [80]. Refining sample acquisition and DNA extraction methods could also minimize non-*S. suis* contamination, enhancing downstream analysis accuracy [81, 82]. Optimizing primer design and sequencing strategies, such as using nested PCR, or targeted enrichment techniques, could improve amplification efficiency and sequencing specificity [54, 83–86]. A shift from amplicon-based methods to whole-genome sequencing (WGS) may further uncover recombination events within the *hsdS-hsdS'* region, offering deeper insights into structural variations and regulatory mechanisms [87, 88]. From a bioinformatics perspective, advanced computational models, including machine learning-based phase variant classification, could detect subtle sequence patterns overlooked by traditional statistical methods. Moreover, the Hidden Markov Model (HMM) can also be implemented to ensure automation for future studies. By integrating functional genomics, advanced sequencing technologies, and computational innovations, future research can further elucidate *hsdS* phase variation's role in *S. suis* adaptation and pathogenicity, potentially identifying new targets for intervention and disease control [89, 90].

In conclusion, this study provides a comprehensive analysis of *hsdS* phase variant distribution in *S. suis* isolates from healthy and symptomatic pigs, offering insights into the genetic and evolutionary factors influencing phase variation in this bacterium. The predominance of phase variant B across all isolate groups suggests selective retention, structural stability, or a functional advantage in bacterial fitness, immune evasion, or gene regulation. However, statistical analyses found no significant differences in *hsdS* phase variant distribution between healthy and symptomatic isolates, indicating that phase variant composition alone does not drive disease outcomes. Additionally, the results confirmed that primer selection did not introduce significant amplification biases, ensuring sequencing reliability. Despite these findings, challenges such as non-specific amplification, barcode misclassification, and sequencing artifacts underscore the need for methodological improvements, including optimized primer design, targeted enrichment techniques, and whole-genome sequencing. While this research establishes a foundation for understanding *hsdS* phase variation in *S. suis*, future studies integrating functional assays, methylation profiling, and host-pathogen interaction analyses will be essential to uncovering the broader biological significance of *hsdS*-mediated gene regulation and its role in *S. suis* adaptation and pathogenicity.

4. MATERIALS AND METHODS

This research was conducted in two major phases: experimental laboratory work (wet lab) and computational bioinformatics analysis (dry lab). Prior to the commencement of the research, DNA samples potentially containing the genome of *S. suis* had been isolated from the swab samples of the healthy and diseased swine tonsils as part of a separate research project. Therefore, this research represents a continuation of that project.

The laboratory work was conducted to acquire data on each DNA sequence of *hsdS* phase variants from different *S. suis* isolates. This part of the research encompassed several critical steps, including sequence amplification and barcoding, DNA library preparation, and high-throughput sequencing using the Nanopore technology [91, 92]. Subsequently, the study progressed to the bioinformatics analysis, which entailed the development and application of an integrated bioinformatics pipeline to process and analyze the sequencing results. There are several key steps in this part of the research, including primer screening and classification, barcode demultiplexing, read filtering, classification of *hsdS* phase variant variants, and statistical analyses aimed at elucidating patterns within the sequencing data.

The research questions articulated for this study will be addressed through the outcomes of the bioinformatics analysis. By integrating the findings from both the experimental and computational phases, this research seeks to characterize the frequency distribution profile of *hsdS* phase variants and investigate their potential associations with the virulence and pathogenicity of *S. suis*. The methodological approach adopted herein is intended to provide a robust framework for uncovering insights into the genetic variability and functional significance of the phase-variable Type I methyltransferase system.

4.1. Acquisition of DNA Sequence Data via Nanopore Sequencing

4.1.1. Sample Selections

As mentioned earlier, the previous project successfully extracted approximately 364 DNA samples from healthy and diseased swine collected from multiple swine farms across Europe. These DNA samples were preserved at a temperature of -80ºC to ensure their long-term stability and integrity for future analysis. From the total collection, a subset of 30 samples was carefully selected for further investigation in this study, with a balanced representation of 15 samples from healthy pigs and 15 from diseased pigs. This selection aimed to enable a focused examination of potential genetic differences in *S. suis* isolates associated with the health status of the host, with particular emphasis on the differential distribution of *hsdS* phase variants. Samples were initially selected based on the abundance of the 16S rRNA sequence, which has already been quantified in a separate project using molecular methods such as quantitative PCR (qPCR). While most samples exceeded a 16S rRNA abundance of 10^6 copies/µL, some with slightly lower abundances were also included, and a few lacked recorded abundance data. This threshold was applied to ensure the selection of samples with sufficient bacterial DNA content, facilitating more accurate downstream analyses. The selected samples were transferred into the -20ºC freezer for easy access during the experiment. The list of samples used in this study can be seen in **Table S1**.

4.1.2. DNA Amplification and Barcoding

DNA amplification and barcoding were conducted simultaneously using conventional Polymerase Chain Reaction (PCR). This process utilized specialized primer sets, where the forward primers were pre-conjugated with 30 unique barcode sequences at their 5'-ends. These barcoded primers allowed for the identification and differentiation of individual samples during bioinformatics analysis. These primers were designed by Maria Juanpere Borras, MSc. and synthesized by IDT™ (Integrated DNA Technologies, Inc.). Three different primer sets were used in this research, each designed to flank the entire *hsdS*–*hsdS’* region according to different scenarios as follows:

1) Primer set 1 (ps1) was designed to amplify the *hsdS*–*hsdS’* region based on the assumption that *hsdS* contains the conserved sequence where the barcoded forward primer, denoted as BC1-30 can bind and *hsdS* is in the same orientation and the same open reading frame with *hsdM*. The reverse primer was designed to anneal to the upstream part of the *rpiA* gene, which is located adjacent to the *hsdS’* gene as shown in **Figure 8**.

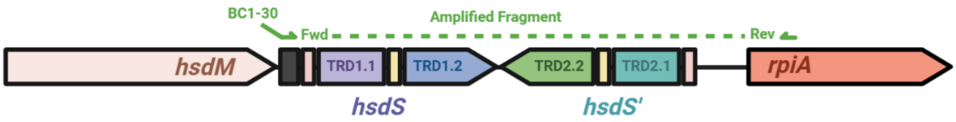

Figure 8. Scenario for ps1, where *hsdS* is in frame with *hsdM*. **Fwd**: Forward Primer, with specific barcode sequence, conjugated at its 5'-end (BC1-30), that anneals on the conserved region of *hsdS*. **Rev**: Reverse primer that anneals on the upstream part of *rpiA*. **Green dashed line**: Amplicon produced from the ps1, with a length of approximately 3.5 kb. **Created by Biorender®.**

2) Primer set 2 (ps2) was designed to account for the possibility of an inverted orientation of the *hsdS* – *hsdS’* regions. In this scenario, the conserved region of *hsdS* would also be inverted, positioning it closer to *rpiA*. If ps1 were used under these conditions, no PCR amplicons would be produced because both the forward and reverse primers of ps1 would face the same direction, preventing proper amplification. To address this potential genetic arrangement, ps2 uses the same barcoded forward primer as ps1, which binds to the conserved region of *hsdS*. However, it incorporates a different reverse primer that binds to the downstream part of *hsdM*, as illustrated in **Figure 9**.

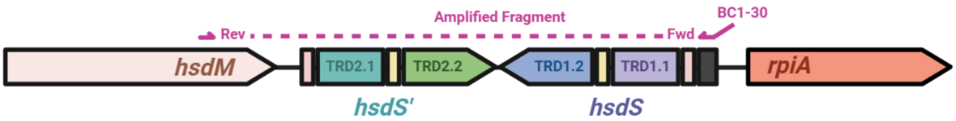

Figure 9. Scenario for ps2, where *hsdS* and *hsdS’* are inverted. **Fwd**: Forward Primer, with specific barcode sequence, conjugated at its 5'-end (BC1-30), that anneals on the conserved region of *hsdS*. **Rev**: Reverse primer that anneals on the downstream part of *hsdM*. **Pink dashed line**: Amplicon produced from the ps2, with a length of approximately 3.5 kb. **Created by Biorender®.**

3) Primer set 3 (ps3) was designed to overcome the potential genetic rearrangements between *hsdS* and *hsdS’* that could lead to the failure of amplification of this region using other primer sets. In this primer set, the forward primer binds to the same region as the reverse primer of ps1. However, in this case, the primer is referred to as the forward primer and is conjugated with the other 30 unique barcode sequences, denoted as BC31–60. The reverse primer used in this set is the same as the reverse primer from Primer ps2, which binds to the downstream part of *hsdM*. This design ensures that the entire *hsdS* – *hsdS*’ region, along with flanking portions of *hsdM* and rpiA, is successfully amplified regardless of the orientation or structural rearrangements of *hsdS* and *hsdS*’ within the genome, as shown in **Figure 10**.

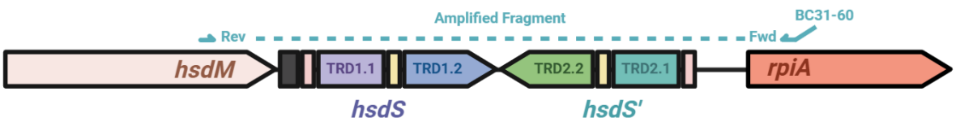

Figure 10. Scenario for ps3, amplification can be carried out regardless of the structure and arrangement of *hsdS-hsdS’*. **Fwd**: Forward Primer, with specific barcode sequence, conjugated at its 5'-end (BC1-30), that anneals on the upstream part of *rpiA*. **Rev**: Reverse primer that anneals on the downstream part of *hsdM*. **Blue dashed line**: Amplicon produced from the ps3, with a length of approximately 4 kb. **Created by Biorender®.**

Details regarding all three primer sets used in this study can be found in **Table S7**. The PCR reactions in this experiment followed the protocol for Q5® Hot Start High-Fidelity DNA polymerase (New England Biolabs, Inc.). Each PCR reaction was prepared for 50 µL with the composition in order as shown in **Table 6** below.

Table 6 PCR reaction component for this study, following the protocol for Q5® Hot Start High-Fidelity DNA polymerase

Component	Working Concentration	Quantity
Q5® Reaction Buffer 5X (NEB, Inc.)	1X	10 µL
GC enhancer 5X (NEB, Inc.)	1X	10 µL
dNTP Mix 10 mM (Promega, Inc.)	0.4 mM	2 µL
Reverse Primer 10 µM	0.5 µM	2.5 µL
Forward Primer 10 µM	0.5 µM	2.5 µL
Q5® Hot Start HiFI Polymerase 2000 $^{U}/_{\mu L}$ (NEB, Inc.)	40 $^{U}/_{\mu L}$	1 µL
DNA sample	1 $^{ng}/_{\mu L}$	50 ng
Nuclease-Free Water	-	until 50 µL

The PCR reactions were run using Thermalcycler® C1000 Touch (BioRad, Inc.) or VWR® Thermal Cyclers (VWR, Inc.). The protocol for the PCR reaction can be seen in **Table 7** below, assuming the *hsdS – hsdS'* region is approximately 3 to 5 kb in size, and the Q5® Hot Start High-Fidelity DNA polymerase is capable of extending the new DNA strand at the amplification rate of 30 $^{s}/_{kb}$ at its ideal reaction condition.

Table 7. PCR reaction condition, following the protocol for Q5® Hot Start High-Fidelity DNA polymerase

Reaction	Temperature	Duration	Cycle
Initial Denaturation	98 ºC	0:30	1
Denaturation	98 ºC	0:10	33
Annealing	52 ºC	0:30	
Extension	72 ºC	3:00	
Final Extension	72 ºC	2:00	1
Storage	4 ºC	∞	-

After the PCR was completed, 5 µL of each PCR product was mixed with 1.67 µL of Blue/Orange 6X Loading Dye (Promega, Inc.) and 5 µL of nuclease-free water (NFW). The entire mixture was then loaded into a well of a 0.8% agarose gel containing 0.5X SYBR™ Safe DNA Gel Stain (ThermoFisher Scientific, Inc.). As the molecular size marker, 5 µL of 1kb DNA Ladder (Promega, Inc.) was loaded into at least one of the wells. The electrophoresis was run at 95 V for 50 min in 1X TAE using Mini-Sub® Cell GT (BioRad, Inc.). Once the electrophoresis was finished, the gel was visualized using Molecular Imager® Gel Doc™ XR+ Imaging System (BioRad, Inc.).

The remaining PCR products were cleaned up using the INVITEK Molecular Kit (Invitrogen, ThermoFisher Scientific, Inc.). The clean-up protocol follows the "Protocol 1: Purification and concentration of DNA fragments from enzymatic reactions, e.g. PCR-products from PCR reactions, cDNA synthesis, enzyme restriction digestions" provided by the kit for a PCR mixture volume up to 50 µL. However, instead of using an elution buffer to elute the DNA, this study used nuclease-free water at room temperature. To measure the amplified DNA concentration, 2 µL of each PCR product that has been cleaned up was used and prepared according to the protocol for Qubit™ dsDNA Broad Range Assay Kit (ThermoFisher Scientific, Inc.). The dsDNA concentration was measured using Qubit™ 4 Fluorometer (ThermoFisher Scientific, Inc.) for dsDNA Broad Range setup. The remaining PCR product was stored in a 4 ºC fridge for future use. In total, 60 PCR products were obtained, corresponding to three primer sets (ps). This included one PCR product generated using ps1, 29 PCR products using ps2, and 30 PCR products using ps3, as summarized in **Table S1**.

4.1.3. DNA Library Preparation

The first step of DNA library preparation, known as end preparation, ensures that DNA molecules are compatible with subsequent adapter ligation, thereby increasing the likelihood of successful sequencing. This process involves modifying the ends of DNA fragments to make them suitable for ligation. Key steps include repairing damaged DNA ends, adding a phosphate group to the 5' ends (phosphorylation), and, in certain workflows, incorporating a single adenine (A) base at the 3' ends (A-tailing) [92, 93]. In this step, a total volume of 50 µL of DNA sample is prepared, containing a mixture of DNA amplicons to be sequenced using the Nanopore platform. The total DNA amount in this mixture must not exceed 200 fmol. Since each DNA sample has a different concentration, as measured by Qubit™, a preliminary step is required to normalize all samples to the same DNA concentration. Given that there are 60 samples in total, each sample will contribute approximately 8 µL (rounded from 8.33 µL) to the final 50 µL mixture. For a DNA size of approximately 3.5 kb, such as *hsdS–hsdS'* region, 200 fmol corresponds to roughly 450 ng of DNA. This means each sample must contribute 7.5 ng of DNA (450 ng/60 samples). To achieve this, each DNA sample must be prepared at a concentration of 9.5 ng/µL (rounded calculation from 7.5 ng / 0.8 µL). Depending on the initial concentration of the DNA sample, measured by Qubit, volumes of 10 µL or 30 µL will be adjusted to this target concentration. Specifically, if the DNA concentration in a sample is greater than 100 ng/µL, a 30 µL volume of diluted DNA should be prepared, and if the DNA concentration is less than 100 ng/µL, a 10 µL volume of diluted DNA will be sufficient. This approach minimizes the risk of pipetting errors when handling very small volumes required to dilute highly concentrated DNA samples. Once the samples are normalized, 8 µL of each sample will be taken and combined to prepare the final 50 µL mixture containing all 60 DNA amplicons.

This 50 µL mixture containing 200 fmol of DNA mixed thoroughly with 7 µL of Ultra™ II End-Prep Reaction Buffer (New England Biolabs, Inc.) and 3 µL of Ultra™ II End-Prep Enzyme Mix (New England Biolabs, Inc.), resulting in a total volume of 60 µL for the end-prep reaction. The reaction mixture was incubated in a Thermal Cycler® C1000 Touch (BioRad, Inc.) with consecutive temperature cycles of 5 minutes at 20°C followed by 5 minutes at 65°C. After incubation, the end-prep reaction mixture was transferred to a clean 1.5 mL DNA LoBind Eppendorf tube, and 60 µL of resuspended AMPure XP Beads (AXP) was added. The mixture was gently mixed and incubated on a nutating mixer at room temperature for 5 minutes. Following incubation, the beads were pelleted using a magnetic Eppendorf tube rack, and the supernatant was carefully removed. To wash the beads, 200 µL of 80% ethanol was added without disturbing the pellet, followed by removal of the ethanol. This washing step was repeated once more to ensure thorough purification. After removing all ethanol, the beads were allowed to air dry for approximately 30 seconds while on the magnetic rack. The pellet was then resuspended in 61 µL of nuclease-free water (NFW) and incubated at room temperature for 2 minutes to ensure complete elution of DNA from the beads. The beads were pelleted again using the magnetic rack, and the supernatant, now containing the purified DNA, was transferred to a clean 1.5 mL DNA LoBind Eppendorf tube. The concentration of the double-stranded DNA (dsDNA) was subsequently measured using the Qubit™ 4 Fluorometer (ThermoFisher Scientific, Inc.) with the Qubit™ dsDNA Broad Range Assay Kit, following the manufacturer's protocol.

Following the completion of the end-prep step, the process advanced to the adapter ligation and clean-up phase. The 60 µL of DNA sample obtained from the end-prep reaction was mixed thoroughly with adapter ligation reagents provided in the Ligation Sequencing Kit V14 (SQK-LSK114, Oxford Nanopore Technologies, Inc.), as detailed in **Table 8**.

Table 8. Adapter ligation components, following the protocol for Ligation Sequencing Kit V14

Component	Volume (µL)
DNA sample	60
Ligation Buffer (LNB)	25
NEBNext Quick T_4 DNA Ligase (NEB, Inc.)	10
Ligation Adapter (LA)	5
TOTAL	**100**

The reaction mixture was incubated at room temperature for 10 minutes to facilitate adapter ligation. After incubation, 40 µL of resuspended AMPure XP Beads (AXP), also included in the Ligation Sequencing Kit, was added and gently mixed. The mixture was then incubated on a nutating mixer for 5 minutes at room temperature to allow the DNA to bind to the beads. The beads were subsequently pelleted using a magnetic rack, and the supernatant was carefully removed. To wash the beads, 250 µL of Short Fragment Buffer (SFB), included in the ligation sequencing kit, was added, and the beads were resuspended by gentle pipetting. The tube was placed back on the magnetic rack to separate the beads from the SFB, and the supernatant was discarded. This washing step was repeated once to ensure the thorough removal of unbound reagents and impurities. After the final wash, the beads were air-dried for approximately 30 seconds on the magnetic rack. The dried beads were resuspended in 15 µL of Elution Buffer (EB), also supplied with the kit, and the suspension was incubated at room temperature for 10 minutes to elute the DNA. Following incubation, the beads were pelleted using the magnetic rack, and the supernatant containing the purified DNA was transferred to a new 1.5 mL DNA LoBind Eppendorf tube. The concentration of the double-stranded DNA (dsDNA) was then measured using the Qubit™ 4 Fluorometer (ThermoFisher Scientific, Inc.) with the Qubit™ dsDNA Broad Range Assay Kit, according to the manufacturer’s protocol.

4.1.4. Nanopore Sequencing

Before loading the DNA library into the SpotON flow cell of the MinION device (Oxford Nanopore Technologies, Inc.), the flow cell was primed to ensure optimal sequencing performance [93, 94]. The priming mix, containing Flow Cell Flush (FCF) and Flow Cell Tether (FCT) along with Bovine Serum Albumin (BSA), was prepared using components from the Sequencing Kit (SQK-LSK114, Oxford Nanopore Technologies, Inc.), as outlined in **Table 9**.

Table 9.Flow cell priming mix, following the protocol for Ligation Sequencing Kit V14

Component	Volume (µL)
Flow Cell Flush (FCF)	1170
Flow Cell Tether (FCT)	30
BSA 20 $^{mg}/_{mL}$ (Thermo Fisher Scientific, Inc.)	12.5
TOTAL	**1212.5**

The flow cell was then securely placed in the MinION device (Oxford Nanopore Technologies, Inc.) under the clip, ensuring proper thermal and electrical contact. The device's MinKNOW software (Oxford Nanopore Technologies, Inc.) was used to assess the number of sequencing pores available. If at least 95% of the pores were confirmed to be available, the priming process was initiated. To begin, the priming port was carefully opened by sliding the cover clockwise. Any air bubbles present in the flow cell were gently removed by drawing them back through the priming port. Once the flow cell was clear of bubbles, 800 µL of the priming mix was slowly introduced through the priming port, taking care to avoid introducing additional bubbles. The flow cell was then incubated for a minimum of 5 minutes to allow proper equilibration. After the initial incubation, the SpotON sample port cover was gently opened to access the sample port. An additional 200 µL of the priming mix was loaded into the priming port, ensuring no air bubbles were introduced, until a slight drop of buffer appeared at the sample port. This step completed the flow cell priming process. The prepared DNA library was then mixed with sequencing reagents from the kit, as specified in **Table 10**.

Table 10. DNA library mix, following the protocol for Ligation Sequencing Kit V14

Component	Volume (µL)
Sequencing Buffer (SB)	37.5
Library Beads (LB)	25.5
DNA library	12
TOTAL	**75**

The mixture was gently pipetted to ensure homogeneity before loading. The entire DNA library mixture was introduced into the flow cell through the SpotON sample port in a dropwise manner, taking care to avoid overfilling and overflowing the flow cell. Once loading was complete, the SpotON sample port cover was securely closed back, followed by the priming port cover. The flow cell was then placed back into the MinION device (Oxford Nanopore Technologies, Inc.) which subsequently was connected to a dedicated supercomputer to initiate sequencing. The sequencing run was set to proceed for up to 72 hours or until no further sequencing reads were detected by the device.

4.2. Bioinformatics Pipeline Development: Sequencing Data Processing

4.2.1. Basecalling: Processing Electrical Signal Data into DNA Sequence Data

To develop the bioinformatics pipeline, a FASTQ file containing sequencing results was required as input. This file provides crucial information, including the sequence ID, the DNA sequence, and the quality score for each nucleotide. However, Nanopore sequencing does not directly generate a FASTQ file; instead, it produces raw data in the POD5 format. The POD5 file primarily stores raw electrical signal data captured during the sequencing process. To generate the required FASTQ file, the raw data underwent a basecalling process, which translates the electrical signals from Nanopore sequencing into nucleotide sequences along with associated quality scores [92, 95].

In this study, basecalling was performed after the sequencing run was completed, utilizing the standalone Dorado basecaller v7.6.8 program (Oxford Nanopore Technologies, Inc.). The Dorado basecaller was selected for its ability to process POD5 files efficiently and generate high-quality FASTQ outputs [95]. The basecalling process required approximately 48 hours to complete, reflecting the time needed to process the entirety of the sequencing dataset. Once the FASTQ file was generated, it was securely transferred to a remote server (`/local/muham020`) hosted by `dayhoff.bioinformatics.nl` and renamed to `methsystem_batch1_sup.fastq` for subsequent processing and analysis. The file transfer was conducted using WinSCP, a secure file transfer client, ensuring data integrity during the transmission. This workflow provided the foundation for the bioinformatics pipeline development and downstream analyses.

4.2.2. Primer Screening and Classification

The bioinformatics pipeline was developed using Python 3 (v3.9.2) and executed in a Linux-based environment within a Conda 3 (v24.11.2) environment. This setup ensured compatibility with essential libraries, such as Biopython, and reproducibility of the workflow by managing dependencies in an isolated computational environment. The pipeline development began following the basecalling process, where raw POD5 files were converted into FASTQ format using the Dorado basecaller. Subsequent preprocessing steps prepared the sequencing data for downstream analyses. These steps included converting the FASTQ file (`methsystem_batch1_sup.fastq`) into a FASTA format file (`original_seq.fasta`) and generating its corresponding reverse complement FASTA file (`reverse_comp.fasta`). Preprocessing was performed using a custom Python script (**Script S1**) that leveraged the `Biopython` library's `SeqIO` module to extract nucleotide sequences while discarding quality scores. The read count of the resulting FASTA file was validated to ensure data integrity. To enable robust primer screening, the reverse complement of each read in the original FASTA file was computed using the `reverse_complement` command from the `Biopython SeqIO` module. The reverse complement reads were then written to a separate FASTA file (`reverse_comp.fasta`). The script also confirmed that the number of reads in the reverse complement FASTA file matched the original FASTA file. Both the original and reverse complement FASTA files were essential for comprehensive primer screening in subsequent steps.

The primer screening pipeline was developed using the Biopython library to identify and classify reads based on the presence and orientation of specific primers. This pipeline systematically processed DNA reads, screening for both forward and reverse primers within user-defined regions, ensuring that only reads meeting the specified criteria were retained for downstream analysis. The preprocessing step utilized two input files: the original FASTA file (`original_seq.fasta`) and the reverse complement FASTA file (`reverse_comp.fasta`). These files were generated in earlier steps to facilitate comprehensive screening of primer matches in both orientations.

The **Script S2** was specifically designed for screening ps1, where the forward primer was 5'-CCTGAACAACTGAAAGCAAG-3' and the reverse primer was 5'-GCAGGTTGGGATTAAGGC-3'. The pipeline began with file verification, where the number of reads in the original and reverse complement FASTA files were counted and validated using a custom read-counting function. This step ensured that the input files were complete and consistent for subsequent analyses. The pipeline then screened for the forward primer within a 200-nucleotide threshold region from both the 5' and 3' ends of each read. Reads containing the forward primer at the 5' end were retained as-is, while those with the primer at the 3' end were reversed to ensure the correct orientation. Reads matching the forward primer were saved to an output file (`result_1_ps1_with_fwd.fasta`), while unmatched reads were stored separately for reference (`discarded_ps1_no_primers.fasta`). Next, reads containing the forward primer were screened for the reverse primer located at the 3' end, ensuring correct orientation. Reads meeting this criterion were saved in a new file (`result_1_ps1_with_both_primers.fasta`), while those with incorrectly positioned primers were stored in a separate file (`discarded_ps1_incorrect_rev_position.fasta`). Afterward, reads were classified based on the number of primer pairs present. Reads with exactly one forward and one reverse primer were categorized as having a single pair of primers and saved in `result_1_ps1_one_primers_pair.fasta`. Reads containing multiple primer pairs were stored separately in `discarded_ps1_multiple_primers.fasta`.

The same pipeline structure was adapted for screening ps2 and ps3 by substituting the forward and reverse primers. For ps2, the forward primer was 5'-CCTGAACAACTGAAAGCAAG-3' and the reverse primer was 5'-GGACAGCTCGTTCTT-3'. For ps3, the forward primer was 5'-GCCTTAATCCCAACCTGC-3' and the reverse primer was 5’-GGACAGCTCGTTCTT-3’. In each case, the respective forward and reverse primers were used as input parameters for the pipeline, and the same read of steps was performed. Outputs for ps2 and ps3 were stored in separate files following the same naming conventions, ensuring clear differentiation between the results of each primer set. For each primer set, the pipeline generated a detailed report summarizing read counts at each stage of processing. The reports included the initial number of reads in the input files, the number of reads retained after forward and reverse primer screening, and the final counts of reads classified by primer pair configuration.

4.2.3. Read Filtering: S. suis Origin

To identify reads originating from *S. suis*, a read filtering step was performed using BLASTn. The purpose of this step was to compare the reads obtained from the primer screening pipeline to a reference genome database and retain only those with significant matches to *S. suis*. This filtering ensured the specificity and relevance of the reads for downstream analysis. The reference genome database, `all_genomes.fna`, which contained genomic sequences from *S. suis*, was prepared as a BLAST database using the `makeblastdb` command. The following bash command was used:

```
makeblastdb -in all_genomes.fna -dbtype nucl -out all_genomes
```

This command generated the necessary database index files (`all_genomes.nhr`, `all_genomes.nin`, and `all_genomes.nsq`) to facilitate efficient sequence alignment with BLASTn. Subsequently, the following command was executed in a bash shell to compare the reads from ps1 (`result_1_ps1_one_primers_pair.fasta`) against the reference database using BLASTn on screen.

```
nice -n 16 blastn -query result_1_ps1_one_primers_pair.fasta \
                  -db all_genomes.fna \
                  -out blast_result_ps1.txt \
                  -outfmt "6 qseqid sseqid pident length mismatch gapopen qstart qend sstart send evalue bitscore" \
                  -max_target_seqs 1 \
                  -perc_identity 80 \
                  -num_threads 24
```

The BLASTn parameters were configured as follows:

- Query File (`result_1_ps1_one_primers_pair.fasta`), containing reads filtered using ps1.
- Database (`all_genomes.fna`), the reference genome database.
- Output File (`blast_result_ps1.txt`), storing alignment results in a tabular format.
- Output Format: Custom format 6, reporting query ID, subject ID, percentage identity, alignment length, mismatches, gaps, start and end positions, e-value, and bit score.
- Filtering Criteria:
  - Maximum of one target sequence per query (-max_target_seqs 1).
  - Minimum percentage identity of 80% (-perc_identity 80).
- Performance Optimization:
  - Multithreading with 24 threads (-num_threads 24).
  - Resource prioritization using the nice command with a priority of 16 (nice -n 16).

This command produced an output file (`blast_result_ps1.txt`) containing the alignment results for each query sequence, identifying potential matches to *S. suis*. The same procedure was performed for the reads filtered using ps2 and ps3, with adjustments made to the query file names and output file names, following the primer set code (ps2 and ps3). These steps ensured that reads from all three primer sets were screened against the reference database, providing comprehensive results for downstream validation and analysis.

Following the BLASTn analysis, a custom Python script (**Script S3**) was utilized to extract reads corresponding to those identified in the BLAST results, isolating reads of *S. suis* origin. This process began with two key input files: the BLAST result file (`blast_result_ps1.txt`), which contained alignment details from the BLASTn analysis, and the FASTA input file (`result_1_ps1_one_primers_pair.fasta`), generated from the primer screening pipeline for ps1. The outputs of this process included a FASTA file containing matched reads (`result_2_ps1_from_suis.fasta`), a FASTA file of discarded reads not matching the BLAST results (`discarded_ps1_not_suis.fasta`), and a summary report file (`report_2_ps1_suis_screening.txt`). The script first parsed the BLAST result file to extract query sequence identifiers (`qseqid`) of all matching reads, storing these identifiers in a set for efficient lookup. It then iterated through each read in the input FASTA file, comparing the read identifiers against the set of matches from the BLAST results. Reads with identifiers present in the BLAST results were categorized as matched and written to the matched output FASTA file. Conversely, reads whose identifiers were not found in the BLAST results were categorized as discarded and written to a separate output FASTA file.

To ensure data integrity, the script implemented several validation checks. It counted the total number of reads in the input FASTA file and verified that this count matched the sum of reads in the matched and discarded output files. Additionally, the script cross-referenced the number of matched reads with the line count of the BLAST result file to confirm consistency between the BLAST results and the output. A detailed report summarizing these statistics was generated and saved in the report file. The report documented key details such as the number of reads in the BLAST results, total input reads, counts of matched and discarded reads, and the successful validation of read counts.

To maintain reproducibility and ensure organized outputs, the workflow stored all files in structured directories. The same procedure was applied to reads generated from ps2 and ps3 by adapting the file names for each dataset. For example, reads for ps2 were processed using the BLAST result file `blast_result_ps2.txt` and the input FASTA file `result_1_ps2_one_primers_pair.fasta`, with corresponding output files named appropriately. Similarly, reads for ps3 were processed using `blast_result_ps3.txt` and `result_1_ps3_one_primers_pair.fasta`. All outputs and reports for ps2 and ps3 were stored in separate directories, ensuring proper traceability and organization for each primer set.

4.2.4. Read Filtering: hsdS Phase Variants Identification

Subsequently, classification of the read based on the *hsdS* phase variants was conducted, but prior to that, a size-based classification was performed. This step aimed to categorize reads into defined size ranges, enabling downstream analysis of reads within biologically relevant length groups. A custom Python script (**Script S4**) was developed for size classification using the Biopython library. The reads were divided into the following size categories:

- ST500 : Reads smaller than 500 base pairs (bp).
- 500-1000 : Reads between 500 and 1000 bp.
- 1000-2000 : Reads between 1000 and 2000 bp.
- 2000-4000 : Reads between 2000 and 4000 bp.
- BT4000 : Reads bigger than 4000 bp.

The input for this process was the FASTA file containing reads from *S. suis* (`result_2_ps1_from_suis.fasta`), generated in the previous read filtering step for ps1. The classified reads were organized into a directory specific to this primer set (`/size_classified_ps1`). The script processed each read in the input FASTA file, determining its length and assigning it to the appropriate size category using predefined thresholds. Each read was then written to a corresponding output FASTA file, named based on its size category (e.g., `ps1_suis_ST_500.fasta`, `ps1_suis_500-1000.fasta`). The script also generated a summary report, saved in a text file (`note_ps1_size_classification.txt`), documenting the number of reads in each size category and verifying that the total number of output reads matched the input read count.

Following the size-based classification, reads were further categorized based on their *hsdS* phase variants. This process involved aligning reads to one of two databases using BLASTn. Reads smaller than 2000 bp, which were expected to consist of partial or full-length segments of the *hsdS* gene, were aligned against the `alleles_all.fasta` database. This database contains four distinct *hsdS* phase variants: A, B, C, and D. Reads larger than 2000 bp, which were likely to encompass the entire *hsdS* gene along with partial or complete *hsdS’* reads, were aligned against the `alleles_combined.fna` database. The `alleles_combined.fna` database includes four combinations of *hsdS*-*hsdS’* phase variants: AD, BC, CB, and DA. The databases (`alleles_all.fna` and `alleles_combined.fna`) were prepared using the `makeblastdb` command, following the same structure as for the *S. suis* database.

The BLASTn search was performed using the same structure as the commands used for *S. suis* classification. Reads from size-based classification were used as queries, and the appropriate database was specified based on the read size. For example, reads smaller than 2000 bp (such as those in `ps1_suis_500-1000.fasta`) were aligned against `alleles_all.fna` using the command:

```
nice -n 16 blastn -query ps1_suis_500-1000.fasta \
                  -db alleles_all.fna \
                  -out blast_result_ps1_500-1000.txt \
                  -outfmt "6 qseqid sseqid pident length mismatch gapopen qstart qend sstart send evalue bitscore" \
                  -max_target_seqs 1 \
                  -perc_identity 80 \
                  -num_threads 24
```

Similarly, reads larger than 2000 bp (such as those in `ps1_suis_2000-4000.fasta`) were aligned against `alleles_combined.fna` with a command following the same structure but using the corresponding query, database, and output names. This classification process served a dual purpose: it reduced computational complexity by aligning reads to appropriate databases based on their lengths, and it enhanced the accuracy of gene identification by tailoring the BLASTn search to the expected read structures. By leveraging these distinct databases and alignment strategies, this step provided a robust foundation for the accurate classification of reads into their respective *hsdS* phase variant categories, enabling a detailed understanding of their distribution and variations within the dataset.

For ps2 and ps3, the same size classification pipeline and BLASTn alignment procedures were applied, with minor adaptations to accommodate the respective input files (`result_2_ps2_from_suis.fasta` and `result_2_ps3_from_suis.fasta`). Outputs for these primer sets were stored in separate directories (`/size_classified_ps2` and `/size_classified_ps3`), and results were organized into size-specific FASTA files following the same naming conventions. The subsequent *hsdS* phase variant classification used the same prepared `alleles_all.fna` and `alleles_combined.fna` databases, ensuring consistency across all datasets. Queries, databases, and output file names were modified accordingly for each primer set to maintain proper traceability.

Following BLASTn alignment, reads were further classified based on their alignment results to specific *hsdS* phase variants or *hsdS-hsdS'* phase variant combinations. A custom Python script (**Script S5**) was used to parse the BLASTn output files and segregate reads into separate FASTA files corresponding to their identified phase variants. For ps1, the input files included the size-classified FASTA files (`ps1_suis_500-1000.fasta`, `ps1_suis_1000-2000.fasta`, `ps1_suis_2000-4000.fasta`, and `ps1_suis_BT4000.fasta`) and the respective BLASTn output files (`blast_result_ps1_500-1000.txt`, `blast_result_ps1_1000-2000.txt`, `blast_result_ps1_2000-4000.txt`, and `blast_result_ps1_BT4000.txt`). The results were organized into an output directory specific to ps1 (`result_3_ps1_hsdS_classified`).

The script categorized reads into phase variant-specific FASTA files based on the BLASTn alignment data. For reads within the FASTA files for 500-1000 bp and 1000-2000 bp, classification was conducted against the `alleles_all.fasta` database, which contained the four individual *hsdS* phase variants: A, B, C, and D. Reads in these size ranges were written to corresponding phase variant-specific files (e.g., `result_3_ps1_hsdS_A_500-1000.fasta`, `result_3_ps1_hsdS_B_1000-2000.fasta`). For reads larger than 2000 bp, classification was conducted against the `alleles_combined.fasta` database, which contained combinations of *hsdS-hsdS'* phase variants: AD, BC, CB, and DA. These reads were also written to separate files corresponding to their identified combination (e.g., `result_3_ps1_hsdS_AD_2000-4000.fasta`, `result_3_ps1_hsdS_CB_BT4000.fasta`). Reads that did not match any phase variant in the BLASTn results were categorized as unmatched and written to a separate file (`discarded_ps1_not_hsdS.fasta`). For each classification step, the script generated a detailed report summarizing the input file, total reads processed, the number of reads matching each phase variant, and unmatched reads. This report was saved in a log file (`report_ps1_hsdS_classification.txt`).

The same pipeline was applied for ps2 and ps3, using their respective size-classified FASTA files and BLASTn output files. For ps2, results were stored in the directory `result_3_ps2_hsdS_classified`, and for ps3, results were saved in `result_3_ps3_hsdS_classified`. The script's structure and functionality remained consistent across all primer sets, with the input and output file names adapted accordingly to maintain proper traceability and organization of results. This workflow enabled the precise classification of reads into their respective *hsdS* or *hsdS-hsdS'* phase variants categories, ensuring accurate and reproducible results for downstream analyses.

4.2.5. Barcode Demultiplexing

Following the identification and classification of reads into specific *hsdS* and *hsdS-hsdS'* phase variants, barcode demultiplexing was performed to further categorize the reads based on barcodes associated with each sample. For ps1 (**Script S6**), the compiled reads from the *hsdS* classification step (`compiled_result_3_ps1_hsdS_classified.fasta`) were processed against the barcode reference file BC1-30.fasta, which contained 30 unique barcode sequences. The forward primer for ps1 (5’-CCTGAACAACTGAAAGCAAG-3’) was used to guide the demultiplexing process, ensuring the correct orientation of reads before barcode scanning.

The demultiplexing process consisted of two main steps. First, reads were scanned for the presence of the forward primer within the first or last 200 bases. Reads containing the forward primer were retained, with their orientation adjusted if necessary (e.g., reversed or reverse complemented). Reads lacking the forward primer were discarded and written to a separate FASTA file (`discarded_no_fwd_ps1.fasta`). In the second step, retained reads were scanned for barcodes within the region preceding or overlapping the forward primer. Reads with an exact match to a barcode were written to barcode-specific FASTA files (e.g., `with_BC1_ps1.fasta`, `with_BC2_ps1.fasta`), while unassigned reads were written to `no_BC_ps1.fasta`. A comprehensive report summarizing read counts and barcode assignments was generated (`report_4_ps1_bc_demultiplexing.txt`).

To account for reads containing barcodes with up to two mismatches relative to the reference barcodes, a second round of barcode screening was performed (**Script S7**). This step aimed to capture reads with minor mutations in their barcode regions that might have been missed during the exact match screening. The input for this step was the unassigned reads from the first round (`no_BC_ps1.fasta`), and the same barcode reference file (`BC1-30.fasta`) was used. Retained reads were first scanned for the forward primer within the first or last 200 bases, and their orientation was corrected as needed. Reads without the forward primer were discarded and written to a separate file (`discarded_no_fwd_ps1_mutated.fasta`). Barcode scanning was then performed, allowing for a maximum of two mismatches between the detected barcode and the reference sequence. A custom similarity function based on the `SequenceMatcher` algorithm was used to identify barcodes with mutations. Reads matching a barcode within the two-mismatch tolerance were assigned to barcode-specific FASTA files (e.g., `with_BC1_ps1_mutated.fasta`, `with_BC2_ps1_mutated.fasta`), while unassigned reads were written to `no_BC_ps1_mutated.fasta`. A summary report was generated (`report_4_ps1_bc_demultiplexing_mutated.txt`), documenting all read counts and verifying consistency across the process.

For ps2 and ps3, the same demultiplexing pipeline was adapted with minor modifications. For ps2, the input files were `compiled_result_3_ps2_`*`hsdS`*`_classified.fasta` for the first round and no_BC_ps2.fasta for the second round. The forward primer (5’-CCTGAACAACTGAAAGCAAG-3’) and the same barcode reference file (`BC1-30.fasta`) were used. For ps3, the input files were `compiled_result_3_ps3_`*`hsdS`*`_classified.fasta` for the first round and no_BC_ps3.fasta for the second round. The forward primer (5’-GCCTTAATCCCAACCTGC-3’) and a barcode reference file specific to ps3 (`BC31-60.fasta`) were used. This two-step barcode demultiplexing approach, incorporating exact matches and mismatch-tolerant scanning, ensured accurate classification of reads while minimizing potential data loss due to barcode mutations.

4.2.6. Isolates Source Classification

Following barcode demultiplexing, the classified reads were further categorized based on their isolate source, distinguishing whether they originated from healthy or symptomatic pigs. This classification step was performed using the barcode information obtained from the demultiplexing process. For ps1 (**Script S8**), reads in barcode-specific FASTA files (from both exact matches and mutated barcode assignments) were used as input. Barcodes corresponding to healthy pigs were identified as BC1, BC3, BC4, BC6, BC7, BC8, BC9, BC10, BC12, BC13, BC14, BC15, BC28, BC29, and BC30, while barcodes associated with symptomatic pigs were identified as BC2, BC5, BC11, BC16, BC17, BC18, BC19, BC20, BC21, BC22, BC23, BC24, BC25, BC26, and BC27. These barcode groups were defined based on experimental metadata linking the barcodes to sample origins.

The classification process began by copying reads from barcode-specific FASTA files into separate directories: one for reads originating from healthy pigs (`from_healthy_pigs`) and another for reads from symptomatic pigs (`from_sick_pigs`). For each barcode, the number of reads was counted, and summary reports were generated to document the read distribution across the two groups. Reads from both the initial barcode demultiplexing (`result_4_ps1_bc_demultiplexing`) and the subsequent mismatch-tolerant barcode assignment (`result_4_ps1_bc_demultiplexing_mutated`) were included in this classification to ensure completeness.

For ps2, the same pipeline was applied using the barcode-specific FASTA files generated in the corresponding demultiplexing steps. The barcodes for ps2 matched those of ps1 (e.g., BC1, BC2, ..., BC30) for identifying healthy and symptomatic pig samples. For ps3, the barcodes include BC31, BC33, BC34, ..., BC60 corresponded to healthy pigs, and BC32, BC35, BC41, ..., BC57 corresponded to symptomatic pigs. This adjustment maintained consistency with the barcode assignments across all primer sets.

The classification process tracked the number of reads in each barcode group and calculated the total reads originating from healthy and symptomatic pigs. Summary statistics, including percentages of reads from each source, were reported in two separate files: `report_5_ps1_from_healthy_pigs.txt for healthy pigs` and `report_5_ps1_from_sick_pigs.txt` for symptomatic pigs. These reports included a breakdown of read counts per barcode, as well as their relative contributions to the total reads in each group. Similar reports were generated for ps2 and ps3, with barcode mappings adapted appropriately for each primer set. This step ensured a clear distinction between reads originating from healthy and symptomatic pigs, facilitating downstream analyses to explore potential associations between read variations and the health status of the host animals.

4.2.7. Read Trimming

Following the classification of reads based on their isolate sources (healthy or symptomatic pig), a read-trimming step was performed to remove non-target regions and retain only the region between the forward and reverse primers. For ps1 (**Script S9**), reads from the directories `from_healthy_pigs` and `from_sick_pigs`, generated during the source classification step, were processed. The forward primer used for ps1 (5'-CCTGAACAACTGAAAGCAAG-3') and the reverse primer (5'-GCAGGTTGGGATTAAGGC-3') guided the trimming process, ensuring that only the region between the primers was retained in each read.

The trimming process was implemented in Python using the `Biopython` library. For each read, the script searched for the forward primer, identified its end position, and located the start of the reverse primer. Reads with valid primer positions, where the reverse primer occurred after the forward primer, were trimmed to include only the region between these primers. Trimmed reads were written to new FASTA files in directories `from_healthy_pigs` and `from_sick_pigs` under the output directory `result_6_ps1_trimmed_sequences`. Reads that did not meet these criteria (e.g., missing one of the primers or with incorrectly ordered primers) were retained in their original form to ensure no data was unintentionally lost during processing. Summary reports were generated for both healthy and symptomatic pig samples, detailing the number of reads processed, the number of reads successfully trimmed, and the barcode-specific trimming statistics. These results were saved in a report file (`report_6_ps1_sequence_trimming.txt`).

For ps2, the same read trimming pipeline was adapted, with input files corresponding to the source classification results for ps2. The forward primer (5'-CCTGAACAACTGAAAGCAAG-3') and reverse primer (5'-GGACAGCTCGTTCTT-3') specific to ps2 were used to guide the trimming process. Similarly, for ps3, the pipeline was applied to the respective source classification files, using the forward primer (5'-GCCTTAATCCCAACCTGC-3') and reverse primer (5'-GGACAGCTCGTTCTT-3') associated with ps3. The file naming conventions for ps2 and ps3 were adjusted accordingly, and the output directories (`result_6_ps2_trimmed_sequences` and `result_6_ps3_trimmed_sequences`) were structured to maintain consistency and traceability. This read trimming step ensured that only target regions between the primers were retained, providing high-quality, standardized data for downstream analyses.

4.2.8. Final hsdS Phase Variants Identification

To finalize the identification and classification of *hsdS* phase variants, reads from the trimming step were categorized into specific *hsdS* or *hsdS-hsdS'* phase variant groups based on their alignment results from BLASTn. For ps1 (**Script S10**), trimmed reads from the healthy pigs group (`from_healthy_pigs`) were processed using size-specific BLAST results, which were aligned to the `alleles_all.fasta` and `alleles_combined.fasta` databases. The input reads were divided into predefined size ranges: 500-1000 bp, 1000-2000 bp, 2000-4000 bp, and greater than 4000 bp (BT4000), with each size range aligned to a corresponding BLAST result file (e.g., `blast_result_ps1_500-1000.txt`, `blast_result_ps1_1000-2000.txt`). Based on the BLAST alignment, reads were assigned to the relevant *hsdS* phase variants (A, B, C, D) or their combinations (AD, BC, CB, DA).

Each read was checked against the BLAST result files using its read ID to determine the best match to a phase variant. Reads that matched a phase variant were written to phase variant-specific FASTA files, such as `hsdS_A_500-1000.fasta` or `hsdS_BC_2000-4000.fasta`, under an output directory organized by barcode (e.g., `with_BC1_normal_ps1`). The pipeline also distinguished reads based on their barcode mutation status (normal or mutated) and created corresponding subdirectories for organization. The script recorded phase variant frequencies, and the number of reads processed for each size range and barcode, generating detailed statistics about the distribution of phase variants. These results were saved in a comprehensive report file (`report_7_ps1_final_hsdS_classification_healthy.txt`), which included information on input file processing, skipped files, and the number of reads assigned to each phase variant.

The same pipeline was adapted for reads from the symptomatic pigs group in ps1. Trimmed reads from the symptomatic pigs group were processed in the same manner, with results saved in a separate output directory (`result_7_ps1_final_hsdS_classification/from_sick_pigs`). For ps2, the pipeline was applied to the trimmed reads and corresponding BLAST results for ps2 (e.g., `blast_result_ps2_500-1000.txt`, `blast_result_ps2_1000-2000.txt`). Similarly, the pipeline for ps3 processed trimmed reads from ps3, using size-specific BLAST results generated for ps3 (e.g., `blast_result_ps3_500-1000.txt`, `blast_result_ps3_2000-4000.txt`). Output directories and file names for ps2 and ps3 were adjusted accordingly to maintain consistency and ensure traceability of results.

This comprehensive pipeline ensured the accurate classification of reads into their respective *hsdS* and *hsdS-hsdS'* phase variants across all groups and primer sets. By organizing reads based on size range, barcodes, and phase variant types, the workflow provided detailed insights into the distribution and variation of phase variants.

4.2.9. Statistical Analysis and Data Visualization

Generation of *hsdS* Phase Variant Frequency Tables

The results from the final *hsdS* classification step were compiled and integrated to facilitate statistical analysis and provide a comprehensive overview of phase variant frequency distributions. For each primer set and source group (healthy and symptomatic pigs), phase variant distribution reports were processed to extract detailed information about the frequency of *hsdS* phase variants (A, B, C, and D) across all barcodes. This process summarized both within-group (barcode-specific) and global patterns of phase variant distribution, enabling downstream statistical and comparative analyses.

A custom Python script (**Script S11**) processed the phase variant distribution reports for ps1 (`report_7_ps1_final_hsdS_classification_healthy.txt` and `report_7_ps1_final_hsdS_classification_sick.txt`), ps2 (`report_7_ps2_final_hsdS_classification_healthy.txt` and `report_7_ps2_final_hsdS_classification_sick.txt`), and ps3 (`report_7_ps3_final_hsdS_classification_healthy.txt` and `report_7_ps3_final_hsdS_classification_sick.txt`) in the form of CSV files. Each report contained information about the total number of reads assigned to specific phase variants per barcode. The script extracted these counts, calculated the proportion of each phase variant within its barcode group (as a percentage of the total reads for that barcode), and computed the global frequency of each phase variant across all barcodes. The processed data were organized into rows with columns for primer set, source group, barcode, phase variant type, read count, within-group frequency (percentage), and global frequency (percentage). These datasets were saved as individual CSV files (e.g., `table_report_7_ps1_final_hsdS_classification_healthy.csv`) in the directory result_8_statistical_analysis.

To streamline the analysis further, a second custom Python script (**Script S12**) integrated the individual CSV files into a single comprehensive dataset. This process involved two steps: filtering out rows with zero read counts and combining the cleaned data. First, the script scanned each CSV file, removed rows where the read count was zero, and saved the filtered files with a modified name (prefixed with `table_essential_`) in the same directory. This ensured that only meaningful data were retained for analysis. Second, the script combined the filtered datasets for all primer sets and source groups into a single integrated CSV file. During this step, redundant columns (e.g., "No.") were removed, and a new "No." column with sequential numbering was added to uniquely identify each row in the consolidated dataset. The final integrated CSV file was saved as `result_8.1_table_combined.csv` in the same directory.

This integration step provided a holistic view of phase variant frequency distributions across all primer sets and health statuses, enabling efficient statistical analysis and comparisons of phase variant variation patterns between groups. The entire workflow was implemented using the `pandas` library in a Linux-based computational environment, ensuring accuracy, reproducibility, and scalability for handling large datasets. By consolidating the data into a single, standardized format, this step facilitated streamlined access for further analyses and interpretation.

Generation of *hsdS* Phase Variants Frequency Distribution Tables

To analyze and summarize the distribution of *hsdS* phase variants across all primer sets (ps1, ps2, ps3) and isolate sources (healthy (H) and symptomatic (S) pigs), a series of summary tables were generated from the integrated phase variant frequency dataset (`result_8.1_table_combined.csv`). This analysis provided a comprehensive and detailed view of phase variant distributions, enabling robust comparisons between groups, primer sets, and phase variant types (A, B, C, D). A custom Python script (**Script S13**) was developed to process the dataset, calculate key metrics, and compile the results into structured outputs.

The first output, `8.1_table_hsdS_distribution_source.csv`, focused on summarizing phase variant distributions within each source group (healthy and symptomatic pigs) across all primer sets. For each phase variant, the script calculated metrics including the total read count, the percentage frequency relative to all reads in the group, the number of barcodes associated with the phase variant, and statistical measures such as the mean, median, standard deviation, and variance of read count across barcodes. Additionally, a list of associated barcodes was recorded for each phase variant. This table provided a detailed insight into the prevalence and variability of each phase variant within healthy and symptomatic pig groups, highlighting differences in phase variant representation between the two sources.

The second output, `8.2_table_hsdS_distribution_primers.csv`, examined phase variant distributions by primer set and isolate's source. The dataset was grouped by primer set, source group, and phase variant type, and the same metrics as in the first table were calculated. This analysis allowed for precise comparisons of phase variant distributions across primer sets (ps1, ps2, ps3) and health statuses, identifying patterns and trends unique to each primer set and source group combination.

To provide a holistic and aggregated view of the dataset, a final comprehensive analysis (**Script S14**) was performed by combining data from the first summary table (`8.1_table_hsdS_distribution_source.csv`). This step produced two additional outputs. The first, `8.3_comprehensive_allele_analysis.csv`, provided a detailed breakdown of metrics for each phase variant across both healthy and symptomatic groups. Metrics included total read counts, global frequencies as a percentage of all reads, barcode counts, and statistical measures such as mean, median, standard deviation, and variance for read counts across barcodes. The analysis also retained separate metrics for healthy and symptomatic groups, along with aggregated barcode lists for each phase variant, offering a granular view of phase variant distribution patterns.

The second output, `8.3_summary_allele_analysis.csv`, focused on a global summary of phase variant distributions. It provided a simplified overview, highlighting the total read counts, global frequencies, total barcode counts, and overall statistical measures such as global mean, median, standard deviation, and variance for each phase variant. This table condensed the key findings from the comprehensive analysis into a format that was easier to interpret briefly while retaining all critical information. The script also contains commands to generate histogram (`allele_distribution_histogram.png`) and pie chart (`allele_distribution_pie_chart.png`) to visualize the *hsdS* phase variants distribution summary in *S. suis*. All output tables and charts were saved in the directory `result_8_statistical_analysis/8.1_hsdS_allele_distribution_table`, ensuring traceability and accessibility for downstream analysis. This computational approach ensured the accuracy, reproducibility, and scalability of the analysis, facilitating the handling and integration of large datasets. By integrating data from all primer sets and isolate sources, this analysis provided a standardized and comprehensive summary of *hsdS* phase variant distributions, serving as a critical resource for further statistical and comparative studies.

Normality Test

A normality test was conducted to assess whether the distribution of *hsdS* phase variant frequencies was consistent with a normal distribution for both isolate sources: healthy (H) and symptomatic (S) pigs. The Shapiro-Wilk test was chosen for this analysis, as it is well-suited for evaluating the normality of small to moderate-sized datasets [35, 96]. The input data for this test was derived from the summary table of phase variant distributions by source (`8.1_table_hsdS_distribution_source.csv`), which contained frequency data for phase variants (A, B, C, D) across the healthy and symptomatic groups.

A custom Python script (**Script S15**) was used to perform the Shapiro-Wilk test on the frequency percentages for each source group. The script first filtered the dataset to separate frequency data for healthy and symptomatic groups. For each group, the Shapiro-Wilk test calculated a test statistic (W) and a corresponding p-value. A p-value greater than 0.05 indicated that the data was likely normally distributed, while a p-value less than or equal to 0.05 suggested that the data did not follow a normal distribution.

The script automatically interpreted the test results for each group, generating a detailed report (`report_normality_test_H_S_detailed.txt`) summarizing the test statistic, p-value, and the conclusion regarding normality for each source group. If the data for both groups were found to be normally distributed, parametric tests were recommended for subsequent analyses. If one or both groups deviated from normality, non-parametric tests were suggested as a more appropriate option. In cases where the dataset for a group contained insufficient data points (fewer than three), the script issued a warning and advised proceeding with caution, favoring non-parametric tests.

The results of the analysis were saved in the output directory `result_8_statistical_analysis/8.2_normality_nonparametric_test`. The workflow was implemented using the `pandas` and `scipy.stats` libraries. This approach ensured a clear and reproducible assessment of the data's normality, providing critical guidance for the selection of appropriate statistical methods in downstream analyses.

Chi-Square (χ^2) Test

To evaluate the association between *hsdS* phase variant distributions and isolate sources (healthy (H) and symptomatic (S) pigs), a χ^2 test of independence was conducted [37]. This statistical test aimed to determine whether observed differences in the distribution of *hsdS* phase variants (A, B, C, and D) between the two sources were statistically significant. The input data for this analysis were derived from two datasets: the integrated phase variant frequency table (`result_8.1_table_combined.csv`) and the source-specific distribution table (`8.1_table_hsdS_distribution_source.csv`). The analysis was carried out using a custom Python script (**Script S16**), which performed the χ^2 test and generated visualizations to aid in data interpretation.

The χ^2 test was applied separately to three key variables: Sequence Count, Barcode Count, and Frequency (%), using a contingency table constructed for each variable. For each test, the script computed the χ^2 statistic, degrees of freedom, p-value, and expected frequencies under the null hypothesis of independence. A p-value less than 0.05 indicated a statistically significant association between phase variant distribution and isolate source. Detailed test results, including observed and expected frequencies, were saved to text files (e.g., `chi_square_sequence_count_report.txt`, `chi_square_barcode_count_report.txt`, `chi_square_frequency_report.txt`) in the directory `result_8_statistical_analysis/8.3_hsdS_analysis`.

In addition to statistical tests, the script generated histograms to visualize the distribution of *hsdS* phase variants across isolate sources. These histograms were created for each variable (Sequence Count, Barcode Count, and Frequency (%)) and included both stacked and grouped formats. The visualizations were designed to provide a clear comparison of phase variant distributions between healthy and symptomatic groups and across phase variants. Specifically, the script generated:

- Stacked histograms showing phase variant distributions by population group (healthy and symptomatic) for each variable.
- Grouped histograms for the same data, highlighting individual contributions of phase variants within each source group.
- Stacked histograms showing distributions of source groups (healthy and symptomatic) by phase variant for each variable.
- Grouped histograms for source group distributions by phase variant, providing another perspective on the data.

The histograms were saved as image files (e.g., `stacked_histogram_sequence_count.png`, `grouped_histogram_frequency.png`) in the output directory. Labels were added to the bars to indicate specific counts, enhancing interpretability.

This comprehensive analysis provided insights into the patterns of *hsdS* phase variant distributions between healthy and symptomatic pigs, supporting the statistical conclusions drawn from the χ^2 test. The workflow was implemented with the `pandas`, `matplotlib`, `seaborn`, and `scipy.stats` libraries in a Linux-based computational environment. This approach ensured robust and reproducible statistical testing while offering clear and informative visual representations of the data. The results served as a critical foundation for understanding the relationship between *hsdS* phase variant distributions and the isolate source characteristics.

Mann-Whitney U Test

To assess differences in *hsdS* phase variant distributions between healthy (H) and symptomatic (S) pig populations, the Mann-Whitney U test was performed [38, 39]. This non-parametric statistical test was chosen because it does not assume a normal distribution, making it appropriate for the data characteristics observed in the normality test. The analysis aimed to identify significant differences in phase variant distributions between the two isolate sources.

A custom Python script (**Script S17**) was developed to perform the Mann-Whitney U test for each *hsdS* phase variant (A, B, C, D) based on the Sequence Count values. The script separated the dataset by phase variant and source group, and the Mann-Whitney U test was applied to compare read counts between the healthy and symptomatic groups for each phase variant. The U statistic and p-value were calculated for each test, and the null hypothesis of no difference between the groups was rejected if the p-value was below 0.05. Detailed results for each phase variant, including the U statistic, p-value, and medians of the healthy and symptomatic groups, were saved as text files (e.g., `mann_whitney_A.txt`) in the directory `result_8_statistical_analysis/8.4_mann_whitney_u_test`.

To enhance interpretability, the data was also normalized by calculating the percentage contribution of each read count relative to the total read count for each source group. This normalized metric, referred to as Normalized Sequence Count (%), was analyzed using the same Mann-Whitney U test approach. Results for the normalized data were saved alongside those for the raw read counts, providing complementary perspectives on the phase variant distribution patterns.

In addition to statistical testing, the script generated visualizations to compare the distributions of read counts between healthy and symptomatic groups. For each metric (raw and normalized), two types of plots were created for all phase variants:

- Box plots, which displayed the distribution of read counts for each phase variant, grouped by source. The median and interquartile ranges were highlighted, and mean values were overlaid as distinct points with labels for clarity.
- Violin plots, which combined elements of box plots and kernel density estimates to show the distribution shape and spread of read counts for each phase variant by source group.

Annotations were added to all plots to display the U statistic and p-value for each phase variant, aiding in the interpretation of statistical significance. Both raw and normalized visualizations were saved as image files (e.g., `box_plot_raw.png`, `violin_plot_normalized.png`) in the output directory. This dual approach of statistical testing and visualization allowed for a comprehensive assessment of phase variant distribution differences between healthy and symptomatic pig populations.

The analysis was implemented with the `pandas`, `scipy.stats`, `matplotlib`, and `seaborn` libraries in a Linux-based computational environment. By combining robust statistical methods with clear visual representations, this workflow provided a detailed understanding of the relationship between *hsdS* phase variant distributions and the isolate sources, supporting further biological and epidemiological interpretations.

Kruskal Wallis Test

To further explore the distribution of *hsdS* phase variants across primer sets (ps1, ps2, and ps3) and between isolate sources (healthy (H) and symptomatic (S) pigs), a Kruskal-Wallis H test was conducted [41, 42]. This non-parametric statistical test was used to determine if there were significant differences in *hsdS* phase variant distributions across multiple independent groups. The test is particularly suitable for analyzing non-normally distributed data and was applied to normalized read counts to account for variations in total read counts between groups.

A custom Python script (**Script S18**) was developed to perform the Kruskal-Wallis test for each *hsdS* phase variant (A, B, C, D) in two scenarios: (1) within the same source group across the three primer sets, and (2) across isolate sources (H vs. S) for each phase variant. The input dataset for this analysis was the integrated phase variant frequency table (`result_8.1_table_combined.csv`), which included normalized read counts calculated as percentages of the total read count for each source group.

For comparisons within the same source group, the dataset was filtered by phase variant and source group (e.g., healthy pigs), and read counts were grouped by primer set. The Kruskal-Wallis test was then applied to these groups to assess differences in read counts across primer sets. To ensure meaningful comparisons, tests were only conducted if there were data from more than one primer set and if the read counts were not identical across all groups. Results, including the Kruskal-Wallis statistic (H), p-value, and significance ($p < 0.05$), were recorded for each test.

For comparisons between source groups (H vs. S), normalized read counts for each phase variant were extracted for healthy and symptomatic pigs, and the Kruskal-Wallis test was applied to evaluate differences between the two groups. This analysis provided insights into the impact of the isolate source on phase variant distribution patterns.

The script generated two outputs: a comprehensive CSV file (`kruskal_wallis_results.csv`) containing the test results for all comparisons and a detailed text report (`kruskal_wallis_report.txt`) summarizing the test statistics, p-values, and interpretations for each phase variant. These outputs were saved in the directory `result_8_statistical_analysis/8.5_kruskal_wallis_across_primers`.

This workflow was implemented with the `pandas` and `scipy.stats` libraries in a Linux-based computational environment. By using the Kruskal-Wallis test, this analysis provided a robust framework for identifying significant differences in phase variant distributions across primer sets and isolate sources, contributing to a deeper understanding of the relationships between primer design, isolate source, and *hsdS* phase variant variation.

5. REFERENCES

1. Roodsant, T.J., Van Der Putten, B., Brizuela, J., Coolen, J.P.M., Baltussen, T.J.H., Schipper, K., Pannekoek, Y., Van Der Ark, K.C.H., Schultsz, C.: The streptococcal phase-variable type i restriction modificationsystem SsuCC20p dictates the methylome of Streptococcus suis impacting the transcriptome and virulence in a zebrafishlarvae infection model. mBio. 15, (2024). https://doi.org/10.1128/mbio.02259-23

2. Dang, H., Nghia, T., Hoa, N.T., Linh, L.D., Campbell, J., Diep, T.S., Van, N., Chau, V., Thi, N., Mai, H., Hien, T.T., Spratt, B., Farrar, J., Schultsz, C.: Human Case of Streptococcus suis Serotype 16 Infection. (2008)

3. Obradovic, M.R., Segura, M., Segalés, J., Gottschalk, M.: Review of the speculative role of co-infections in Streptococcus suis-associated diseases in pigs, (2021)

4. Wang, H., Fan, Q., Wang, Y., Yi, L., Wang, Y.: Rethinking the control of Streptococcus suis infection: Biofilm formation, (2024)

5. Atack, J.M., Weinert, L.A., Tucker, A.W., Husna, A.U., Wileman, T.M., F Hadjirin, N., Hoa, N.T., Parkhill, J., Maskell, D.J., Blackall, P.J., Jennings, M.P.: Streptococcus suis contains multiple phase-variable methyltransferases that show a discrete lineage distribution. Nucleic Acids Res. 46, 11466–11476 (2018). https://doi.org/10.1093/nar/gky913

6. Lun, Z.-R., Wang, Q.-P., Chen, X.-G., Li, A.-X., Zhu, X.-Q.: Streptococcus suis: an emerging zoonotic pathogen. (2007). https://doi.org/10.1016/S1473

7. Gottschalk, M., Xu, J., Calzas, C., Segura, M.: Streptococcus suis: A new emerging or an old neglected zoonotic pathogen?, (2010)

8. Agoston, Z., Terhes, G., Hannauer, P., Gajdacs, M.O., Urban, E.: Fatal case of bacteremia caused by Streptococcus suis in a splenectomized man and a review of the European literature, (2020)

9. Gottschalk, M., Segura, M., Xu, J.: Streptococcus suis infections in humans: the Chinese experience and the situation in North America., (2007)

10. Uruén, C., García, C., Fraile, L., Tommassen, J., Arenas, J.: How Streptococcus suis escapes antibiotic treatments, (2022)

11. Lunha, K., Chumpol, W., Samngamnim, S., Jiemsup, S., Assavacheep, P., Yongkiettrakul, S.: Antimicrobial Susceptibility of Streptococcus suis Isolated from Diseased Pigs in Thailand, 2018–2020. Antibiotics. 11, (2022). https://doi.org/10.3390/antibiotics11030410

12. Wang, C.-Z., Wang, M.-G., Chu, Y.-F., Sun, R.-Y., Li, J.-G., Li, X.-A., Sun, J., Liu, Y.-H., Zhou, Y.-F., Liao, X.-P.: Antibiotic Resistance Patterns and Molecular Characterization of Streptococcus suis Isolates from Swine and Humans in China. Microbiol Spectr. 11, (2023). https://doi.org/10.1128/spectrum.00309-23

13. Dechêne-Tempier, M., Jouy, E., Bayon-Auboyer, M.H., Bougeard, S., Chauvin, C., Libante, V., Payot, S., Marois-Créhan, C.: Antimicrobial resistance profilesof Streptococcus suis isolated from pigs, wild boars, and humans in France between 1994 and 2020. J Clin Microbiol. 61, (2023). https://doi.org/10.1128/jcm.00164-23

14. Roodsant, T.J., Van Der Putten, B.C.L., Tamminga, S.M., Schultsz, C., Van Der Ark, K.C.H.: Identification of Streptococcus suis putative zoonotic virulence factors: A systematic review and genomic meta-analysis, (2021)

15. Tram, G., Jennings, M.P., Blackall, P.J., Atack, J.M.: Streptococcus suis pathogenesis—A diverse array of virulence factors for a zoonotic lifestyle. In: Advances in Microbial Physiology. pp. 217–257. Academic Press (2021)

16. Segura, M., Calzas, C., Grenier, D., Gottschalk, M.: Initial steps of the pathogenesis of the infection caused by Streptococcus suis: fighting against nonspecific defenses, (2016)

17. Esgleas, M., Li, Y., Hancock, M.A., Harel, J., Dubreuil, J.D., Gottschalk, M.: Isolation and characterization of α-enolase, a novel fibronectin-binding protein from Streptococcus suis. Microbiology (N Y). 154, 2668–2679 (2008). https://doi.org/10.1099/mic.0.2008/017145-0

18. Phillips, Z.N., Husna, A.U., Jennings, M.P., Seib, K.L., Atack, J.M.: Phasevarions of bacterial pathogens-phase-variable epigenetic regulators evolving from restriction-modification systems, (2019)

19. Phillips, Z.N., Tram, G., Seib, K.L., Atack, J.M.: Phase-variable bacterial loci: How bacteria gamble to maximise fitness in changing environments, (2019)

20. Tram, G., Jen, F.E.-C., Phillips, Z.N., Timms, J., Husna, A.-U., Jennings, M.P., Blackall, P.J., Atack, J.M.: Streptococcus suis Encodes Multiple Allelic Variants of a Phase-Variable Type III DNA Methyltransferase, ModS, That Control Distinct Phasevarions. mSphere. 6, (2021). https://doi.org/10.1128/msphere.00069-21

21. Manso, A.S., Chai, M.H., Atack, J.M., Furi, L., De Ste Croix, M., Haigh, R., Trappetti, C., Ogunniyi, A.D., Shewell, L.K., Boitano, M., Clark, T.A., Korlach, J., Blades, M., Mirkes, E., Gorban, A.N., Paton, J.C., Jennings, M.P., Oggioni, M.R.: A random six-phase switch regulates pneumococcal virulence via global epigenetic changes. Nat Commun. 5, (2014). https://doi.org/10.1038/ncomms6055

22. Moldovan, E., Moldovan, V.: Controls in real-time polymerase chain reaction based techniques, (2020)

23. Lu, T., Stroot, P.G., Oerther, D.B.: Reverse Transcription of 16S rRNA to Monitor Ribosome-Synthesizing Bacterial Populations in the Environment. Appl Environ Microbiol. 75, 4589–4598 (2009). https://doi.org/10.1128/AEM.02970-08

24. Clarridge, J.E.: Impact of 16S rRNA gene sequence analysis for identification of bacteria on clinical microbiology and infectious diseases, (2004)

25. Bosshard, P.P., Abels, S., Zbinden, R., Böttger, E.C., Altwegg, M.: Ribosomal DNA sequencing for identification of aerobic gram-positive rods in the clinical laboratory (an 18-month evaluation). J Clin Microbiol. 41, 4134–4140 (2003). https://doi.org/10.1128/JCM.41.9.4134-4140.2003

26. Qin, Y., Wu, L., Zhang, Q., Wen, C., Van Nostrand, J.D., Ning, D., Raskin, L., Pinto, A., Zhou, J.: Effects of error, chimera, bias, and GC content on the accuracy of amplicon sequencing. mSystems. 8, (2023). https://doi.org/10.1128/msystems.01025-23

27. Schloss, P.D., Gevers, D., Westcott, S.L.: Reducing the effects of PCR amplification and sequencing Artifacts on 16s rRNA-based studies. PLoS One. 6, (2011). https://doi.org/10.1371/journal.pone.0027310

28. Delahaye, C., Nicolas, J.: Sequencing DNA with nanopores: Troubles and biases. PLoS One. 16, (2021). https://doi.org/10.1371/journal.pone.0257521

29. Chen, H.J., Zhang, Y.R., Wang, B., Liao, R., Duan, X.H., Yang, C.Y., Chen, J., Hao, Y.T., Shu, Y.S., Cai, L.L., Leng, X., Qian, N.S., Sun, D.W., Niu, B., Zhou, Q.: Characterization and mitigation of artifacts derived from NGS library preparation due to structure-specific sequences in the human genome. BMC Genomics. 25, (2024). https://doi.org/10.1186/s12864-024-10157-w

30. Wang, Q., Cai, R., Huang, A., Wang, X., Qu, W., Shi, L., Li, C., Yan, H.: Comparison of Oropharyngeal Microbiota in Healthy Piglets and Piglets With Respiratory Disease. Front Microbiol. 9, (2018). https://doi.org/10.3389/fmicb.2018.03218

31. Li, Z., Wang, X., Di, D., Pan, R., Gao, Y., Xiao, C., Li, B., Wei, J., Liu, K., Qiu, Y., Ma, Z.: Comparative analysis of the pulmonary microbiome in healthy and diseased pigs. Molecular Genetics and Genomics. 296, 21–31 (2021). https://doi.org/10.1007/s00438-020-01722-5

32. Pirolo, M., Espinosa-Gongora, C., Bogaert, D., Guardabassi, L.: The porcine respiratory microbiome: recent insights and future challenges, (2021)

33. Su, Y., Yao, W., Perez-Gutierrez, O.N., Smidt, H., Zhu, W.Y.: Changes in abundance of Lactobacillus spp. and Streptococcus suis in the stomach, jejunum and ileum of piglets after weaning. FEMS Microbiol Ecol. 66, 546–555 (2008). https://doi.org/10.1111/j.1574-6941.2008.00529.x

34. Khatun, N.: Applications of Normality Test in Statistical Analysis. Open J Stat. 11, 113–122 (2021). https://doi.org/10.4236/ojs.2021.111006

35. Hanusz, Z., Tarasinska, J., Zielinski, W.: Shapiro-Wilk Test With Known Mean. REVSTAT-Statistical Journal. 14, 89–100 (2016). https://doi.org/10.57805/revstat.v14i1.180

36. Rani Das, K.: A Brief Review of Tests for Normality. American Journal of Theoretical and Applied Statistics. 5, 5 (2016). https://doi.org/10.11648/j.ajtas.20160501.12

37. McHugh, M.L.: The Chi-square test of independence. Biochem Med (Zagreb). 23, 143–149 (2013). https://doi.org/10.11613/BM.2013.018

38. Birnbaum, Z.W.: On a Use of the Mann-Whitney Statistic. (1956)

39. Hart, A.: Mann-Whitney test is not just a test of medians: differences in spread can be important. BMJ: Education and Debate. 323, 391–393 (2001)

40. Conroy, R.M.: What hypotheses do “nonparametric” two-group tests actually test? Stata Journal. 12, 182–190 (2012). https://doi.org/10.1177/1536867x1201200202

41. Kruskal, W.H., Wallis, W.A.: Use of Ranks in One-Criterion Variance Analysis. (1952)

42. Ostertagová, E., Ostertag, O., Kováč, J.: Methodology and application of the Kruskal-Wallis test. Applied Mechanics and Materials. 611, 115–120 (2014). https://doi.org/10.4028/www.scientific.net/AMM.611.115

43. Lorenz, T.C.: Polymerase chain reaction: Basic protocol plus troubleshooting and optimization strategies. Journal of Visualized Experiments. (2012). https://doi.org/10.3791/3998

44. Xu, B., Zhang, P., Li, W., Liu, R., Tang, J., Fan, H.: hsdS, belonging to the type i restriction-modification system, contributes to the streptococcus suis serotype 2 survival ability in phagocytes. Front Microbiol. 8, 1524 (2017). https://doi.org/10.3389/fmicb.2017.01524

45. Poritz, M.A., Ririe, K.M.: Getting things backwards to prevent primer dimers, (2014)

46. Asif, S., Khan, M., Waqar Arshad, M., Shabbir, M.I.: PCR Optimization for Beginners: A Step by Step Guide. Research in Molecular Medicine. 9, 81–102 (2021). https://doi.org/10.32598/rmm.9.2.1189.1

47. Burpo, F.J.: A critical review of PCR primer design algorithms and cross-hybridization case study.

48. Estrada, A.A., Gottschalk, M., Gebhart, C.J., Marthaler, D.G.: Comparative analysis of Streptococcus suis genomes identifies novel candidate virulence-associated genes in North American isolates. Vet Res. 53, 23 (2022). https://doi.org/10.1186/s13567-022-01039-8

49. Humphray, S.J., Scott, C.E., Clark, R., Marron, B., Bender, C., Camm, N., Davis, J., Jenks, A., Noon, A., Patel, M., Sehra, H., Yang, F., Rogatcheva, M.B., Milan, D., Chardon, P., Rohrer, G., Nonneman, D., de Jong, P., Meyers, S.N., Archibald, A., Beever, J.E., Schook, L.B., Rogers, J.: A high utility integrated map of the pig genome. Genome Biol. 8, (2007). https://doi.org/10.1186/gb-2007-8-7-r139

50. Walker, S.P., Barrett, M., Hogan, G., Flores Bueso, Y., Claesson, M.J., Tangney, M.: Non-specific amplification of human DNA is a major challenge for 16S rRNA gene sequence analysis. Sci Rep. 10, (2020). https://doi.org/10.1038/s41598-020-73403-7

51. Marquet, M., Zöllkau, J., Pastuschek, J., Viehweger, A., Schleußner, E., Makarewicz, O., Pletz, M.W., Ehricht, R., Brandt, C.: Evaluation of microbiome enrichment and host DNA depletion in human vaginal samples using Oxford Nanopore's adaptive sequencing. Sci Rep. 12, (2022). https://doi.org/10.1038/s41598-022-08003-8

52. Bedarf, J.R., Beraza, N., Khazneh, H., Özkurt, E., Baker, D., Borger, V., Wüllner, U., Hildebrand, F.: Much ado about nothing? Off-target amplification can lead to false-positive bacterial brain microbiome detection in healthy and Parkinson's disease individuals. Microbiome. 9, (2021). https://doi.org/10.1186/s40168-021-01012-1

53. Tan, E., Erwin, B., Dames, S., Ferguson, T., Buechel, M., Irvine, B., Voelkerding, K., Niemz, A.: Specific versus nonspecific isothermal DNA amplification through thermophilic polymerase and nicking enzyme activities. Biochemistry. 47, 9987–9999 (2008). https://doi.org/10.1021/bi800746p

54. Puskas, L.G., Bottka, S.: Reduction of Mispriming in Amplification Reactions with Restricted PCR. (1995)

55. Dinesh Kumar, K., Senthil Kumar, G., Santhiya, S.T.: Nonspecific PCR amplification of CRYBB2- pseudogene leads to misconception of natural variation as mutation, (2012)

56. Lasken, R.S., Stockwell, T.B.: Mechanism of chimera formation during the Multiple Displacement Amplification reaction. BMC Biotechnol. 7, (2007). https://doi.org/10.1186/1472-6750-7-19

57. Lu, N., Qiao, Y., Lu, Z., Tu, J.: Chimera: The spoiler in multiple displacement amplification, (2023)

58. Smyth, R.P., Schlub, T.E., Grimm, A., Venturi, V., Chopra, A., Mallal, S., Davenport, M.P., Mak, J.: Reducing chimera formation during PCR amplification to ensure accurate genotyping. Gene. 469, 45–51 (2010). https://doi.org/10.1016/j.gene.2010.08.009

59. Burden, D.L., Meyer, J.J., Michael, R.D., Anderson, S.C., Burden, H.M., Peña, S.M., Leong-Fern, K.J., Van Ye, L.A., Meyer, E.C., Keranen-Burden, L.M.: Confirming Silent Translocation through Nanopores with Simultaneous Single-Molecule Fluorescence and Single-Channel Electrical Recordings. Anal Chem. 95, 18020–18028 (2023). https://doi.org/10.1021/acs.analchem.3c02329

60. Aksimentiev, A., Heng, J.B., Timp, G., Schulten, K.: Microscopic kinetics of DNA translocation through synthetic nanopores. Biophys J. 87, 2086–2097 (2004). https://doi.org/10.1529/biophysj.104.042960

61. Muzard, J., Martinho, M., Mathé, J., Bockelmann, U., Viasnoff, V.: DNA translocation and unzipping through a nanopore: Some geometrical effects. Biophys J. 98, 2170–2178 (2010). https://doi.org/10.1016/j.bpj.2010.01.041

62. MacKenzie, M., Argyropoulos, C.: An Introduction to Nanopore Sequencing: Past, Present, and Future Considerations, (2023)

63. Branton, D., Deamer, D.W., Marziali, A., Bayley, H., Benner, S.A., Butler, T., Di Ventra, M., Garaj, S., Hibbs, A., Huang, X., Jovanovich, S.B., Krstic, P.S., Lindsay, S., Ling, X.S., Mastrangelo, C.H., Meller, A., Oliver, J.S., Pershin, Y. V., Ramsey, J.M., Riehn, R., Soni, G. V., Tabard-Cossa, V., Wanunu, M., Wiggin, M., Schloss, J.A.: The potential and challenges of nanopore sequencing, (2008)

64. Kanagawa', T.: REVIEW Bias and Artifacts in Multitemplate Polymerase Chain Reactions (PCR). (2003)

65. Golenberg, E.M., Bickel, A., Weihs, P.: Effect of highly fragmented DNA on PCR. (1996)

66. Roux, K.H.: Optimization and troubleshooting in PCR. Cold Spring Harb Protoc. 4, (2009). https://doi.org/10.1101/pdb.ip66

67. Sauvage, T., Cormier, A., Delphine, P.: A comparison of Oxford nanopore library strategies for bacterial genomics. BMC Genomics. 24, (2023). https://doi.org/10.1186/s12864-023-09729-z

68. Garcia-Pedemonte, D., Carcereny, A., Gregori, J., Quer, J., Garcia-Cehic, D., Guerrero, L., Ceretó-Massagué, A., Abid, I., Bosch, A., Costafreda, M.I., Pintó, R.M., Guix, S.: Comparison of Nanopore and Synthesis-Based Next-Generation Sequencing Platforms for SARS-CoV-2 Variant Monitoring in Wastewater. Int J Mol Sci. 24, (2023). https://doi.org/10.3390/ijms242417184

69. Illumina: Effects of Index Misassignment on Multiplexing and Downstream Analysis.

70. Koch, C., Reilly-O'Donnell, B., Gutierrez, R., Lucarelli, C., Ng, F.S., Gorelik, J., Ivanov, A.P., Edel, J.B.: Nanopore sequencing of DNA-barcoded probes for highly multiplexed detection of microRNA, proteins and small biomarkers. Nat Nanotechnol. 18, 1483–1491 (2023). https://doi.org/10.1038/s41565-023-01479-z

71. Ezpeleta, J., Garcia Labari, I., Villanova, G.V., Bulacio, P., Lavista-Llanos, S., Posner, V., Krsticevic, F., Arranz, S., Tapia, E.: Robust and scalable barcoding for massively parallel long-read sequencing. Sci Rep. 12, (2022). https://doi.org/10.1038/s41598-022-11656-0

72. Sibly, R.M., Curnow, R.N.: Allele frequencies and selection coefficients in locally adapted populations. J Theor Biol. 565, (2023). https://doi.org/10.1016/j.jtbi.2023.111463

73. Müller, R., Kaj, I., Mugal, C.F.: A Nearly Neutral Model of Molecular Signatures of Natural Selection after Change in Population Size. Genome Biol Evol. 14, (2022). https://doi.org/10.1093/gbe/evac058

74. Vitti, J.J., Grossman, S.R., Sabeti, P.C.: Detecting natural selection in genomic data, (2013)

75. Wang, Z., Gerstein, M., Snyder, M.: RNA-Seq: A revolutionary tool for transcriptomics, (2009)

76. Vogel, C., Marcotte, E.M.: Insights into the regulation of protein abundance from proteomic and transcriptomic analyses. Nat Rev Genet. 13, 227–232 (2012). https://doi.org/10.1038/nrg3185

77. Yong, W.S., Hsu, F.M., Chen, P.Y.: Profiling genome-wide DNA methylation, (2016)

78. Mehrmohamadi, M., Sepehri, M.H., Nazer, N., Norouzi, M.R.: A Comparative Overview of Epigenomic Profiling Methods, (2021)

79. Kammers, K., Cole, R.N., Tiengwe, C., Ruczinski, I.: Detecting significant changes in protein abundance. EuPA Open Proteom. 7, 11–19 (2015). https://doi.org/10.1016/j.euprot.2015.02.002

80. Faber, J., Fonseca, L.M.: How sample size influences research outcomes, (2014)

81. Basset, P., Castella, V.: Positive impact of DNA contamination minimization procedures taken within the laboratory. Forensic Sci Int Genet. 38, 232–235 (2019). https://doi.org/10.1016/j.fsigen.2018.11.013

82. Kampmann, M.L., Børsting, C., Morling, N.: Decrease DNA contamination in the laboratories. Forensic Sci Int Genet Suppl Ser. 6, e577–e578 (2017). https://doi.org/10.1016/j.fsigss.2017.09.223

83. Illumina: Effects of Index Misassignment on Multiplexing and Downstream Analysis.

84. Na, Y.S., Kim, H.L., Kim, C.M., Yun, N.R., Kim, D.M.: Usefulness of nested polymerase chain reaction with clinical specimens for the diagnosis of leptospirosis: A case series and a review of the literature. J Korean Med Sci. 35, (2020). https://doi.org/10.3346/JKMS.2020.35.E301

85. Green, M.R., Sambrook, J.: Nested polymerase chain reaction (PCR). Cold Spring Harb Protoc. 2019, (2019). https://doi.org/10.1101/pdb.prot095182

86. Marsilio, F., Di Martino, B., Decaro, N., Buonavoglia, C.: A novel nested PCR for the diagnosis of calicivirus infections in the cat. Vet Microbiol. 105, 1–7 (2005). https://doi.org/10.1016/j.vetmic.2004.09.017

87. Guo, G., Kong, X., Du, D., Wei, D., Yu, Y., Zhang, W.: Genome-wide association study identifies the virulence-associated marker in Streptococcus suis serotype 2. Infection, Genetics and Evolution. 92, (2021). https://doi.org/10.1016/j.meegid.2021.104894

88. Aiewsakun, P., Ruangchai, W., Jaemsai, B., Bodharamik, T., Meemetta, W., Senapin, S.: Whole genome sequence and characterisation of Streptococcus suis 3112, isolated from snakeskin gourami, Trichopodus pectoralis. BMC Genomics. 25, (2024). https://doi.org/10.1186/s12864-024-10736-x

89. Miyake, J., Kaneshita, Y., Asatani, S., Tagawa, S., Niioka, H., Hirano, T.: Graphical classification of DNA sequences of HLA alleles by deep learning. Hum Cell. 31, 102–105 (2018). https://doi.org/10.1007/s13577-017-0194-6

90. Yang, A., Zhang, W., Wang, J., Yang, K., Han, Y., Zhang, L.: Review on the Application of Machine Learning Algorithms in the Sequence Data Mining of DNA, (2020)

91. Oxford Nanopore Technologies, Inc.: Advantages of nanopore sequencing, https://nanoporetech.com/platform/technology/advantages-of-nanopore-sequencing

92. Oxford Nanopore Technologies, Inc.: How Nanopore Sequencing Works, https://nanoporetech.com/platform/technology

93. Oxford Nanopore Technologies, Inc.: Ligation sequencing V14 - PCR barcoding (SQK-LSK114 with EXP-PBC001 or EXP-PBC096), https://nanoporetech.com/document/ligation-sequencing-gdna-pcr-barcoding-sqk-lsk114-with-exp

94. Oxford Nanopore Technologies, Inc.: Flow Cells, https://nanoporetech.com/platform/technology/flow-cells-and-nanopores

95. Oxford Nanopore Technologies, Inc.: How Basecalling Works, https://nanoporetech.com/platform/technology/basecalling

96. Dudley, R.: The Shapiro-Wilk Test for Normality. , Cambridge, Massachussetts (2012)

6. SUPPLEMENTARY FIGURES, TABLES, AND SCRIPTS

Table S1 Summary of DNA samples for the analysis of *S. suis* genomes isolated from healthy (blue rows) and diseased swine (pink rows). Samples were labeled with unique barcode sequences (BC1–30 for ps1 and ps2, and BC31–60 for ps3). The ps1 was initially tested on all 30 samples, with successful amplification only observed for a sample with BC4. The ps2, a replacement for ps1, was tested on the remaining 29 samples except BC4, with the same barcodes (BC1–30). The ps3, designed to deal with structural rearrangement of *hsdS-hsdS'*, was tested using barcodes BC31–60 on samples previously tested with ps2, excluding BC6, BC7, and BC9, since these three samples did not work with ps3. Therefore, to maintain a total of 30 samples, 3 additional samples that are not tested with ps2 are included (BC36, 37, and 39).

No.	Barcode			Box	Slot	Name	Age	Farm	Health Status	16S rRNA (copy/μL)	Qubit (ng/μL)		
	ps1	ps2	ps3								ps1	ps2	ps3
1	-	1	31	161B	H3	PR31TM1	Week-1	ES2	Healthy	1.92 x 10^6	-	143	18.1
2	-	2	32	161D	E6	PS9372	Week+3	NL1	Diseased	1.49 x 10^6	-	89.6	65.4
3	-	3	33	161A	H5	PR32TM1	Week-1	ES2	Healthy	2.04 x 10^6	-	59.6	125
4	4	-	34	161B	G5	PQ20TM1	Week-1	ES1	Healthy	N/A*	46.6	-	167
5	-	5	35	161D	H2	PS9989	Week+3	NL1	Diseased	1.12 x 10^6	-	222	144
6	-	6	-	161C	I4	PB30TM1	Week-1	ES3	Healthy	8.89 x 10^5	-	94.2	-
7	-	-	36	161C	E8	PR1ST1	Week+1	ES2	Healthy	1.45 x 10^6	-	-	44
8	-	7	-	161B	B5	PQ11T3	Week+3	ES1	Healthy	N/A	-	88	-
9	-	-	37	161C	F5	PR15T0	Week_0	ES2	Healthy	1.07 x 10^6	-	-	10.6
10	-	8	38	161B	C7	PR38TM1	Week-1	ES2	Healthy	4,76 x 10^6	-	73.8	172
11	-	9	-	161A	I9	PQ03TM1	Week-1	ES1	Healthy	N/A	-	138	-
12	-	-	39	161D	G7	JT19	Week_0	ES4	Healthy	1.17 x 10^6	-	-	130
13	-	10	40	161D	E7	PS6946	Week+3	NL1	Healthy	5.395 x 10^5	-	141	41.4
14	-	11	41	161D	H5	PS9379	Week+3	NL1	Diseased	3.93 x 10^6	-	194	82
15	-	12	42	161B	B7	PR08TM1	Week-1	ES2	Healthy	4.81 x 10^6	-	36	66
16	-	13	43	161B	C2	PB32TM1	Week-1	ES3	Healthy	2.14 x 10^6	-	34.8	27.2
17	-	14	44	161B	I3	PR15TM1	Week-1	ES2	Healthy	3.59 x 10^6	-	85	44.8
18	-	15	45	161B	D3	PR40TM1	Week-1	ES2	Healthy	1.65 x 10^6	-	18.1	83
19	-	16	46	161B	I6	PB33TM1	Week-1	ES3	Diseased	3.16 x 10^6	-	40	149
20	-	17	47	I61C	A1	PR25TM1	Week-1	ES2	Diseased	1.49 x 10^6	-	136	43.6
21	-	18	48	I61C	A2	PR24TM1	Week-1	ES2	Diseased	2.95 x 10^6	-	184	32.4
22	-	19	49	I61C	A4	PR14TM1	Week-1	ES2	Diseased	1.05 x 10^6	-	110	34.8
23	-	20	50	I61C	A7	PR04TM1	Week-1	ES2	Diseased	2.30 x 10^6	-	27.2	63.2
24	-	21	51	I61C	B4	PR36TM1	Week-1	ES2	Diseased	7.4 x 10^6	-	46.6	184
25	-	22	52	I61C	C6	PR05TM1	Week-1	ES2	Diseased	5.61 x 10^6	-	29.6	22
26	-	23	53	I61C	C9	PB15NM1	Week-1	ES3	Diseased	1.04 x 10^6	-	38	20.4
27	-	24	54	I61C	D1	PR35T0	Week_0	ES2	Diseased	1.97 x 10^6	-	21.4	16.4
28	-	25	55	I61C	E3	PR14T0	Week_0	ES2	Diseased	1.85 x 10^6	-	92.6	59.6
29	-	26	56	I61C	E5	PR25T0	Week_0	ES2	Diseased	4.03 x 10^6	-	17.9	43.6
30	-	27	57	I61C	E6	PR24T0	Week_0	ES2	Diseased	2.43 x 10^6	-	59.4	51
31	-	28	58	I61D	C3	PR12T1	Week+1	ES2	Healthy	1.06 x 10^6	-	46.4	23.8
32	-	29	59	I61D	C9	PR34T1	Week+1	ES2	Healthy	7 x 10^5	-	31	11.8
33	-	30	60	I61D	D9	JT14	Week_0	ES4	Healthy	8.65 x 10^5	-	63.6	9.56

***16S rRNA abundance record is not available.**

Table S2 Comprehensive hierarchical classification of *hsdS* phase variants across primer sets, isolate sources, and barcodes represented as read count, within-group frequency (%), and global frequency (%). Within-group frequency was calculated by dividing the read count for a specific phase variant and the total phase variant within the same barcode. Global frequency was calculated by dividing the read count for a specific phase variant and the total phase variant within the same isolate source group. **H**= isolate source from healthy pigs, **S**= Isolate source from symptomatic pigs.

No.	Primer Set	Source	Barcode	Phase Variant	Read Count	Within-Group Frequency (%)	Global Frequency (%)
1	ps1	H	3	A	0	0	0
2	ps1	H	3	B	2	100	2.7
3	ps1	H	3	C	0	0	0
4	ps1	H	3	D	0	0	0
5	ps1	H	12	A	0	0	0
6	ps1	H	12	B	64	96.97	86.49
7	ps1	H	12	C	0	0	0
8	ps1	H	12	D	2	3.03	2.7
9	ps1	H	13	A	0	0	0
10	ps1	H	13	B	4	100	5.41
11	ps1	H	13	C	0	0	0
12	ps1	H	13	D	0	0	0
13	ps1	H	15	A	0	0	0
14	ps1	H	15	B	2	100	2.7
15	ps1	H	15	C	0	0	0
16	ps1	H	15	D	0	0	0
17	ps1	S	2	A	0	0	0
18	ps1	S	2	B	10	100	4.37
19	ps1	S	2	C	0	0	0
20	ps1	S	2	D	0	0	0
21	ps1	S	5	A	0	0	0
22	ps1	S	5	B	141	100	61.57
23	ps1	S	5	C	0	0	0
24	ps1	S	5	D	0	0	0
25	ps1	S	11	A	0	0	0
26	ps1	S	11	B	70	100	30.57
27	ps1	S	11	C	0	0	0
28	ps1	S	11	D	0	0	0
29	ps1	S	21	A	0	0	0
30	ps1	S	21	B	6	100	2.62
31	ps1	S	21	C	0	0	0
32	ps1	S	21	D	0	0	0
33	ps1	S	22	A	0	0	0
34	ps1	S	22	B	2	100	0.87
35	ps1	S	22	C	0	0	0
36	ps1	S	22	D	0	0	0
37	ps2	H	1	A	0	0	0
38	ps2	H	1	B	2	100	2.7
39	ps2	H	1	C	0	0	0
40	ps2	H	1	D	0	0	0
41	ps2	H	10	A	0	0	0

42	ps2	H	10	B	2	100	2.7
43	ps2	H	10	C	0	0	0
44	ps2	H	10	D	0	0	0
45	ps2	H	12	A	0	0	0
46	ps2	H	12	B	70	100	94.59
47	ps2	H	12	C	0	0	0
48	ps2	H	12	D	0	0	0
49	ps2	S	2	A	0	0	0
50	ps2	S	2	B	6	100	2.45
51	ps2	S	2	C	0	0	0
52	ps2	S	2	D	0	0	0
53	ps2	S	5	A	0	0	0
54	ps2	S	5	B	170	97.7	69.39
55	ps2	S	5	C	0	0	0
56	ps2	S	5	D	4	2.3	1.63
57	ps2	S	11	A	0	0	0
58	ps2	S	11	B	59	100	24.08
59	ps2	S	11	C	0	0	0
60	ps2	S	11	D	0	0	0
61	ps2	S	21	A	0	0	0
62	ps2	S	21	B	6	100	2.45
63	ps2	S	21	C	0	0	0
64	ps2	S	21	D	0	0	0
65	ps3	H	33	A	0	0	0
66	ps3	H	33	B	4	100	16.67
67	ps3	H	33	C	0	0	0
68	ps3	H	33	D	0	0	0
69	ps3	H	36	A	0	0	0
70	ps3	H	36	B	0	0	0
71	ps3	H	36	C	4	100	16.67
72	ps3	H	36	D	0	0	0
73	ps3	H	38	A	0	0	0
74	ps3	H	38	B	10	83.33	41.67
75	ps3	H	38	C	0	0	0
76	ps3	H	38	D	2	16.67	8.33
77	ps3	H	44	A	0	0	0
78	ps3	H	44	B	2	100	8.33
79	ps3	H	44	C	0	0	0
80	ps3	H	44	D	0	0	0
81	ps3	H	45	A	0	0	0
82	ps3	H	45	B	2	100	8.33
83	ps3	H	45	C	0	0	0
84	ps3	H	45	D	0	0	0
85	ps3	S	32	A	0	0	0
86	ps3	S	32	B	11	47.83	9.57
87	ps3	S	32	C	12	52.17	10.43
88	ps3	S	32	D	0	0	0

89	ps3	S	35	A	4	50	3.48
90	ps3	S	35	B	4	50	3.48
91	ps3	S	35	C	0	0	0
92	ps3	S	35	D	0	0	0
93	ps3	S	47	A	0	0	0
94	ps3	S	47	B	12	75	10.43
95	ps3	S	47	C	4	25	3.48
96	ps3	S	47	D	0	0	0
97	ps3	S	48	A	0	0	0
98	ps3	S	48	B	4	100	3.48
99	ps3	S	48	C	0	0	0
100	ps3	S	48	D	0	0	0
101	ps3	S	49	A	0	0	0
102	ps3	S	49	B	4	100	3.48
103	ps3	S	49	C	0	0	0
104	ps3	S	49	D	0	0	0
105	ps3	S	51	A	0	0	0
106	ps3	S	51	B	46	100	40
107	ps3	S	51	C	0	0	0
108	ps3	S	51	D	0	0	0
109	ps3	S	52	A	0	0	0
110	ps3	S	52	B	2	100	1.74
111	ps3	S	52	C	0	0	0
112	ps3	S	52	D	0	0	0
113	ps3	S	53	A	0	0	0
114	ps3	S	53	B	2	100	1.74
115	ps3	S	53	C	0	0	0
116	ps3	S	53	D	0	0	0
117	ps3	S	56	A	0	0	0
118	ps3	S	56	B	4	66.67	3.48
119	ps3	S	56	C	0	0	0
120	ps3	S	56	D	2	33.33	1.74
121	ps3	S	57	A	0	0	0
122	ps3	S	57	B	4	100	3.48
123	ps3	S	57	C	0	0	0
124	ps3	S	57	D	0	0	0

Table S3 Overview of *hsdS* phase variants distribution across primer sets and isolate source group. The Read Count column represents the total number of reads corresponding to a specific phase variant within the same isolate source group and primer set. Frequency (%) indicates the proportion of a given *hsdS* phase variant within the respective isolate source group and primer set. Barcode Count reflects the total number of barcodes covering the specific phase variant, as detailed in the Barcode List column. Additionally, statistical measures—including Mean, Median, Standard Deviation, and Variance—are provided for the read count of each phase variant within the same isolate source group and primer set.

No.	Primer Set	Source	Phase Variant	Read Count	Frequency (%)	Barcode Count	Mean	Median	St Dev	Variance	Barcode List
1	ps1	H	A	0	0	0	0	0	0	0	()
2	ps1	H	B	72	97.3	4	18	3	30.68	941.33	(BC3,BC12,BC13,BC15)
3	ps1	H	C	0	0	0	0	0	0	0	()
4	ps1	H	D	2	2.7	1	2	0	1	1	(BC12)
5	ps1	S	A	0	0	0	0	0	0	0	()
6	ps1	S	B	229	100	5	45.8	10	60.07	3608.2	(BC2,BC5,BC11,BC21,BC22)
7	ps1	S	C	0	0	0	0	0	0	0	()
8	ps1	S	D	0	0	0	0	0	0	0	()
9	ps2	H	A	0	0	0	0	0	0	0	()
10	ps2	H	B	74	100	3	24.67	2	39.26	1541.33	(BC1,BC10,BC12)
11	ps2	H	C	0	0	0	0	0	0	0	()
12	ps2	H	D	0	0	0	0	0	0	0	()
13	ps2	S	A	0	0	0	0	0	0	0	()
14	ps2	S	B	241	98.37	4	60.25	32.5	77.31	5977.58	(BC2,BC5,BC11,BC21)
15	ps2	S	C	0	0	0	0	0	0	0	()
16	ps2	S	D	4	1.63	1	4	0	2	4	(BC5)
17	ps3	H	A	0	0	0	0	0	0	0	()
18	ps3	H	B	18	75	4	4.5	2	3.85	14.8	(BC33,BC38,BC44,BC45)
19	ps3	H	C	4	16.67	1	4	0	1.79	3.2	(BC36)
20	ps3	H	D	2	8.33	1	2	0	0.89	0.8	(BC38)
21	ps3	S	A	4	3.48	1	4	0	1.26	1.6	(BC35)
22	ps3	S	B	93	80.87	10	9.3	4	13.35	178.23	(BC32,BC35,BC47,BC48,BC49,BC51,BC52,BC53,BC56,BC57)
23	ps3	S	C	16	13.91	2	8	0	3.86	14.93	(BC32,BC47)
24	ps3	S	D	2	1.74	1	2	0	0.63	0.4	(BC56)

Table S4 Coalesced table for all primer sets, highlighting the distribution of *hsdS* phase variants across two isolate source groups. The total read count from symptomatic pigs is significantly higher than healthy pigs. Across both groups, phase variant B of *hsdS* exhibited the highest frequency. The Read Count column represents the total number of reads corresponding to a specific phase variant within the same isolate source group. Frequency (%) indicates the proportion of a given phase variant within each isolate source group. Barcode Count summarizes the total number of barcodes covering the specific phase variant, as detailed in the Barcode List column. Additionally, statistical measures—including Mean, Median, Standard Deviation, and Variance—are provided for the read count of each phase variant within the same isolate source group and primer set.

No.	Primer Set	Source	Phase Variant	Read Count	Frequency (%)	Barcode Count	Mean	Median	St Dev	Variance	Barcode List
1	ALL	H	A	0	0	0	0	0	0	0	()
2	ALL	H	B	164	95.35	10	16.4	2	25.06	628.24	(BC3,BC12,BC13,BC15,BC1,BC10,BC33,BC38,BC44,BC45)
3	ALL	H	C	4	2.33	1	4	0	1.15	1.33	(BC36)
4	ALL	H	D	4	2.33	2	2	0	0.78	0.61	(BC12,BC38)
5	ALL	S	A	4	0.68	1	4	0	0.92	0.84	(BC35)
6	ALL	S	B	563	95.59	15	37.53	6	48.97	2397.8	(BC2,BC5,BC11,BC21,BC22,BC32,BC35,BC47,BC48,BC49,BC51,BC52,BC53,BC56,BC57)
7	ALL	S	C	16	2.72	2	8	0	2.85	8.14	(BC32,BC47)
8	ALL	S	D	6	1.02	2	3	0	1	1.01	(BC5,BC56)

Table S5 A comprehensive overview of *hsdS* phase variant distribution across all primer sets and isolate sources. phase variant B is the most predominant, accounting for 95.53% of all reads, followed by phase variants C (2.63%) and D (1.31%), while phase variant A (0.53%) is the least represented. The Read Count Total column indicates the total number of reads identified for each phase variant. Global Frequency (%) represents the proportion of each phase variant in the dataset. Barcode Count Total summarizes the total number of unique barcodes associated with each phase variant, with specific barcode identifiers listed in the Barcode List All column. Additionally, statistical parameters—including Global Mean, Median, Standard Deviation (StDev), and Variance—are provided to describe the distribution of read counts for each phase variant across different barcodes.

No.	Primer Set	Source	Phase Variant	Read Count Total	Global Frequency (%)	Barcode Count Total	Global Mean	Global Median	Global StDev	Global Variance	Barcode List All
1	ALL	ALL	A	4	0.53	1	4	0	0.46	0.2116	(),(BC35)
2	ALL	ALL	B	727	95.53	25	29.08	4	37.02	1370.4804	(BC3,BC12,BC13,BC15,BC1,BC10,BC33,BC38,BC44,BC45), (BC2,BC5,BC11,BC21,BC22,BC32,BC35,BC47,BC48,BC49,BC51,BC52,BC53,BC56,BC57)
3	ALL	ALL	C	20	2.63	3	6.67	0	2	4	(BC36),(BC32,BC47)
4	ALL	ALL	D	10	1.31	4	2.5	0	0.89	0.7921	(BC12,BC38),(BC5,BC56)

Table S6 Chi-square observed and expected (A) read count and (B) frequency (%) of each *hsdS* phase variant from two different isolate source groups.

A

Observed Read Count			Expected Read Count		
Phase Variant	**H (Healthy)**	**S (Symptomatic)**	**Phase Variant**	**H (Healthy)**	**S (Symptomatic)**
A	0	4	**A**	0.90	3.10
B	164	563	**B**	164.32	562.68
C	4	16	**C**	4.52	15.48
D	4	6	**D**	2.26	7.74

B

Observed Frequency (%)			Expected Frequency (%)		
Phase Variant	**H (Healthy)**	**S (Symptomatic)**	**Phase Variant**	**H (Healthy)**	**S (Symptomatic)**
A	0.00	0.68	**A**	0.34	0.34
B	95.35	95.59	**B**	95.47	95.47
C	2.33	2.72	**C**	2.52	2.52
D	2.33	1.02	**D**	1.68	1.68

Table S7 Primer list used in this study, including the barcode sequences that are conjugated to the 5'-end of the forward primer. **(A)** Primer name and its corresponding sequence from 5'-end to 3'-end. **(B)** Primers combination for ps1, ps2, and ps3. **(C)** List of the forward primers used for ps1 and ps2, conjugated with 30 unique barcode sequences at its 5'-end (BC1-30). **(D)** List of the forward primers used for ps3, conjugated with 30 unique barcode sequences at its 5'-end (BC31-60)

A

Primer Name	Primer Sequence (5' – 3')
Met_system_FWD_consv	CCTGAACAACTGAAAGCAAG
Met_system_REV_rpiA	GCCTTAATCCCAACCTGC
Met_system_FWD_*hsdM*	CAGMAAGAACGAGCTGTCC*

***M can be A or C**

B

Primer	Primer Set		
	ps1	ps2	ps3
Forward**	FWD_consv	FWD_consv	REV_rpiA
Reverse	REV_rpiA	FWD_*hsdM*	FWD_*hsdM*

****with barcode sequence**

C

Primer Name	Primer Sequence (5' – 3')
BC1_FWD_consv	CACAAAGACACCGACAACTTTCTT**CCTGAACAACTGAAAGCAAG**
`BC2_FWD_consv	ACAGACGACTACAAACGGAATCGA**CCTGAACAACTGAAAGCAAG**
BC3_FWD_consv	CCTGGTAACTGGGACACAAGACTC**CCTGAACAACTGAAAGCAAG**
BC4_FWD_consv	TAGGGAAACACGATAGAATCCGAA**CCTGAACAACTGAAAGCAAG**
BC5_FWD_consv	AAGGTTACACAAACCCTGGACAAG**CCTGAACAACTGAAAGCAAG**
BC6_FWD_consv	GACTACTTTCTGCCTTTGCGAGAA**CCTGAACAACTGAAAGCAAG**
BC7_FWD_consv	AAGGATTCATTCCCACGGTAACAC**CCTGAACAACTGAAAGCAAG**
BC8_FWD_consv	ACGTAACTTGGTTTGTTCCCTGAA**CCTGAACAACTGAAAGCAAG**
BC9_FWD_consv	AACCAAGACTCGCTGTGCCTAGTT**CCTGAACAACTGAAAGCAAG**
BC10_FWD_consv	GAGAGGACAAAGGTTTCAACGCTT**CCTGAACAACTGAAAGCAAG**
BC11_FWD_consv	TCCATTCCCTCCGATAGATGAAAC**CCTGAACAACTGAAAGCAAG**
BC12_FWD_consv	TCCGATTCTGCTTCTTTCTACCTG**CCTGAACAACTGAAAGCAAG**
BC13_FWD_consv	AGAACGACTTCCATACTCGTGTGA**CCTGAACAACTGAAAGCAAG**
BC14_FWD_consv	AACGAGTCTCTTGGGACCCATAGA**CCTGAACAACTGAAAGCAAG**
BC15_FWD_consv	AGGTCTACCTCGCTAACACCACTG**CCTGAACAACTGAAAGCAAG**
BC16_FWD_consv	CGTCAACTGACAGTGGTTCGTACT**CCTGAACAACTGAAAGCAAG**
BC17_FWD_consv	ACCCTCCAGGAAAGTACCTCTGAT**CCTGAACAACTGAAAGCAAG**
BC18_FWD_consv	CCAAACCCAACAACCTAGATAGGC**CCTGAACAACTGAAAGCAAG**
BC19_FWD_consv	GTTCCTCGTGCAGTGTCAAGAGAT**CCTGAACAACTGAAAGCAAG**
BC20_FWD_consv	TTGCGTCCTGTTACGAGAACTCAT**CCTGAACAACTGAAAGCAAG**
BC21_FWD_consv	GAGCCTCTCATTGTCCGTTCTCTA**CCTGAACAACTGAAAGCAAG**
BC22_FWD_consv	ACCACTGCCATGTATCAAAGTACG**CCTGAACAACTGAAAGCAAG**
BC23_FWD_consv	CTTACTACCCAGTGAACCTCCTCG**CCTGAACAACTGAAAGCAAG**
BC24_FWD_consv	GCATAGTTCTGCATGATGGGTTAG**CCTGAACAACTGAAAGCAAG**
BC25_FWD_consv	GTAAGTTGGGTATGCAACGCAATG**CCTGAACAACTGAAAGCAAG**
BC26_FWD_consv	CATACAGCGACTACGCATTCTCAT**CCTGAACAACTGAAAGCAAG**
BC27_FWD_consv	CGACGGTTAGATTCACCTCTTACA**CCTGAACAACTGAAAGCAAG**
BC28_FWD_consv	TGAAACCTAAGAAGGCACCGTATC**CCTGAACAACTGAAAGCAAG**
BC29_FWD_consv	CTAGACACCTTGGGTTGACAGACC**CCTGAACAACTGAAAGCAAG**
BC30_FWD_consv	TCAGTGAGGATCTACTTCGACCCA**CCTGAACAACTGAAAGCAAG**

D

Primer Name	Primer Sequence (5' – 3')
BC31 _REV_rpiA	TGCGTACAGCAATCAGTTACATTG**GCCTTAATCCCAACCTGC**
BC32 _REV_rpiA	CCAGTAGAAGTCCGACAACGTCAT**GCCTTAATCCCAACCTGC**
BC33 _REV_rpiA	CAGACTTGGTACGGTTGGGTAACT**GCCTTAATCCCAACCTGC**
BC34 _REV_rpiA	GGACGAAGAACTCAAGTCAAAGGC**GCCTTAATCCCAACCTGC**
BC35 _REV_rpiA	CTACTTACGAAGCTGAGGGACTGC**GCCTTAATCCCAACCTGC**
BC36 _REV_rpiA	ATGTCCCAGTTAGAGGAGGAAACA**GCCTTAATCCCAACCTGC**
BC37 _REV_rpiA	GCTTGCGATTGATGCTTAGTATCA**GCCTTAATCCCAACCTGC**
BC38 _REV_rpiA	ACCACAGGAGGACGATACAGAGAA**GCCTTAATCCCAACCTGC**
BC39 _REV_rpiA	CCACAGTGTCAACTAGAGCCTCTC**GCCTTAATCCCAACCTGC**
BC40 _REV_rpiA	TAGTTTGGATGACCAAGGATAGCC**GCCTTAATCCCAACCTGC**
BC41 _REV_rpiA	GGAGTTCGTCCAGAGAAGTACACG**GCCTTAATCCCAACCTGC**
BC42 _REV_rpiA	CTACGTGTAAGGCATACCTGCCAG**GCCTTAATCCCAACCTGC**
BC43 _REV_rpiA	CTTTCGTTGTTGACTCGACGGTAG**GCCTTAATCCCAACCTGC**
BC44 _REV_rpiA	AGTAGAAAGGGTTCCTTCCCACTC**GCCTTAATCCCAACCTGC**
BC45 _REV_rpiA	GATCCAACAGAGATGCCTTCAGTG**GCCTTAATCCCAACCTGC**
BC46 _REV_rpiA	GCTGTGTTCCACTTCATTCTCCTG**GCCTTAATCCCAACCTGC**
BC47 _REV_rpiA	GTGCAACTTTCCCACAGGTAGTTC**GCCTTAATCCCAACCTGC**
BC48 _REV_rpiA	CATCTGGAACGTGGTACACCTGTA**GCCTTAATCCCAACCTGC**
BC49 _REV_rpiA	ACTGGTGCAGCTTTGAACATCTAG**GCCTTAATCCCAACCTGC**
BC50 _REV_rpiA	ATGGACTTTGGTAACTTCCTGCGT**GCCTTAATCCCAACCTGC**
BC51 _REV_rpiA	GTTGAATGAGCCTACTGGGTCCTC**GCCTTAATCCCAACCTGC**
BC52 _REV_rpiA	TGAGAGACAAGATTGTTCGTGGAC**GCCTTAATCCCAACCTGC**
BC53 _REV_rpiA	AGATTCAGACCGTCTCATGCAAAG**GCCTTAATCCCAACCTGC**
BC54 _REV_rpiA	CAAGAGCTTTGACTAAGGAGCATG**GCCTTAATCCCAACCTGC**
BC55 _REV_rpiA	TGGAAGATGAGACCCTGATCTACG**GCCTTAATCCCAACCTGC**
BC56 _REV_rpiA	TCACTACTCAACAGGTGGCATGAA**GCCTTAATCCCAACCTGC**
BC57 _REV_rpiA	GCTAGGTCAATCTCCTTCGGAAGT**GCCTTAATCCCAACCTGC**
BC58 _REV_rpiA	CAGGTTACTCCTCCGTGAGTCTGA**GCCTTAATCCCAACCTGC**
BC59 _REV_rpiA	TCAATCAAGAAGGGAAAGCAAGGT**GCCTTAATCCCAACCTGC**
BC60 _REV_rpiA	CATGTTCAACCAAGGCTTCTATGG**GCCTTAATCCCAACCTGC**

Script S1 Python script to transform FASTQ file to FASTA files of the original sequence and its reverse complement

```python
from Bio import SeqIO
from Bio.Seq import Seq


def count_sequences(file, file_type):
    """Count the number of sequences in a file."""
    return sum(1 for _ in SeqIO.parse(file, file_type))


def convert_fastq_to_fasta(input_fastq, output_fasta):
    """Convert FASTQ to FASTA and count sequences."""
    SeqIO.convert(input_fastq, "fastq", output_fasta, "fasta")
    print(f"Converted {input_fastq} to {output_fasta}")
    fasta_count = count_sequences(output_fasta, "fasta")
    print(f"Sequence count in {output_fasta}: {fasta_count}")


def generate_reverse_complement(input_fasta, output_fasta):
    """Generate reverse complement sequences."""
    reverse_complement = []
    for record in SeqIO.parse(input_fasta, "fasta"):
        record.seq = record.seq.reverse_complement()
        reverse_complement.append(record)
    SeqIO.write(reverse_complement, output_fasta, "fasta")
    print(f"Generated reverse complement file: {output_fasta}")
    reverse_count = count_sequences(output_fasta, "fasta")
    print(f"Sequence count in {output_fasta}: {reverse_count}")


if __name__ == "__main__":
    input_fastq = "methsystem_batch1_sup.fastq"
    original_fasta = "original_seq.fasta"
    reverse_comp_fasta = "reverse_comp.fasta"

    # Convert FASTQ to FASTA
    convert_fastq_to_fasta(input_fastq, original_fasta)

    # Generate Reverse Complement FASTA
    generate_reverse_complement(original_fasta, reverse_comp_fasta)
```

Script S2 Python script to screen the presence of forward and reverse primers for primer set 1 within the sequencing results

```
import os
from Bio import SeqIO
from Bio.Seq import Seq


def ensure_directory(path):
    """Ensure the directory exists, create it if it does not."""
    if not os.path.exists(path):
        os.makedirs(path)


def count_sequences(file, file_type):
    """Count the number of sequences in a file."""
    return sum(1 for _ in SeqIO.parse(file, file_type))


def flip_sequence(record):
    """Flip the sequence to reverse its orientation."""
    record.seq = record.seq.reverse_complement()
    return record


def screen_for_primer(sequences, primer, threshold, direction, flip=False):
    """Screen sequences for the presence of a primer within the threshold region."""
    matched = []
    unmatched = []
    primer = primer.lower()
    if flip:
        primer = str(Seq(primer).reverse_complement())

    for record in sequences:
        sequence = str(record.seq).lower()
        if direction == "start":
            region = sequence[:threshold]
        elif direction == "end":
            region = sequence[-threshold:]
        else:
            raise ValueError("Invalid direction. Must be 'start' or 'end'.")

        if primer in region:
            if direction == "end" and flip:
                record = flip_sequence(record)
            matched.append(record)
        else:
            unmatched.append(record)

    return matched, unmatched
```

```python
def count_primers_in_sequence(sequence, primer):
    """Count the total occurrences of a primer in a given sequence."""
    return sequence.lower().count(primer.lower())

def classify_sequences_by_primer_count(input_fasta, output_one_pair_fasta,
output_multiple_pairs_fasta, fwd_primer, rev_primer, threshold):
    """Classify sequences based on the number of primers and write them to separate files."""
    one_pair_records = []
    multiple_pairs_records = []

    for record in SeqIO.parse(input_fasta, "fasta"):
        sequence = str(record.seq).lower()
        fwd_primer_count = count_primers_in_sequence(sequence, fwd_primer)
        rev_primer_count = count_primers_in_sequence(sequence, rev_primer)

        fwd_position_valid = sequence.find(fwd_primer) < threshold if fwd_primer_count == 1 else
False
        rev_position_valid = sequence.rfind(rev_primer) > (len(sequence) - threshold) if
rev_primer_count == 1 else False

        if fwd_primer_count == 1 and rev_primer_count == 1 and fwd_position_valid and
rev_position_valid:
            one_pair_records.append(record)
        else:
            multiple_pairs_records.append(record)

    SeqIO.write(one_pair_records, output_one_pair_fasta, "fasta")
    SeqIO.write(multiple_pairs_records, output_multiple_pairs_fasta, "fasta")

    return len(one_pair_records), len(multiple_pairs_records)

def primer_screening_pipeline():
    # Paths and directory setup
    base_dir = "/local/muham020/suismetsys/trial_pipeline"
    output_dir = os.path.join(base_dir, "result_1_ps1_primers_screening")
    ensure_directory(output_dir)

    # File paths
    original_fasta = os.path.join(base_dir, "original_seq.fasta")
    reverse_comp_fasta = os.path.join(base_dir, "reverse_comp.fasta")
    report_file = os.path.join(output_dir, "report_primers_screening.txt")

    # Step 1: Verify Files
    fasta_count = count_sequences(original_fasta, "fasta")
    reverse_count = count_sequences(reverse_comp_fasta, "fasta")
    print(f"Sequence counts: {fasta_count} in original FASTA, {reverse_count} in reverse complement
FASTA.")
```

```
    # Step 2: Screen for Forward Primer
    fwd_primer = "cctgaacaactgaaagcaag"
    threshold = 200
    with_fwd_fasta = os.path.join(output_dir, "result_1_ps1_with_fwd.fasta")
    discarded_fasta = os.path.join(output_dir, "discarded_ps1_no_primers.fasta")

    sequences = list(SeqIO.parse(original_fasta, "fasta")) + list(SeqIO.parse(reverse_comp_fasta,
"fasta"))
    matched_start, unmatched = screen_for_primer(sequences, fwd_primer, threshold, "start")
    matched_end, unmatched = screen_for_primer(unmatched, fwd_primer, threshold, "end", flip=True)

    with_fwd = matched_start + matched_end
    SeqIO.write(with_fwd, with_fwd_fasta, "fasta")
    SeqIO.write(unmatched, discarded_fasta, "fasta")

    # Step 3: Screen for Reverse Primer
    rev_primer = "gcaggttgggattaaggc"
    with_both_fasta = os.path.join(output_dir, "result_1_ps1_with_both_primers.fasta")
    uncorrect_rev_fasta = os.path.join(output_dir, "discarded_ps1_incorrect_rev_position.fasta")

    unmatched = list(SeqIO.parse(with_fwd_fasta, "fasta"))
    matched_start, unmatched = screen_for_primer(unmatched, rev_primer, threshold, "start")
    SeqIO.write(matched_start, uncorrect_rev_fasta, "fasta")
    matched_end, unmatched = screen_for_primer(unmatched, rev_primer, threshold, "end")

    with_both = matched_end
    SeqIO.write(with_both, with_both_fasta, "fasta")

    # Step 4: Classify Sequences Based on Primer Count
    one_pair_fasta = os.path.join(output_dir, "result_1_ps1_one_primers_pair.fasta")
    multiple_pairs_fasta = os.path.join(output_dir, "discarded_ps1_multiple_primers.fasta")
    one_pair_count, multiple_pairs_count = classify_sequences_by_primer_count(
        with_both_fasta, one_pair_fasta, multiple_pairs_fasta, fwd_primer, rev_primer, threshold
    )

    # Generate report
    with open(report_file, "w") as report:
        report.write(f"Sequence counts: {fasta_count} in original FASTA, {reverse_count} in reverse
complement FASTA.\n")
        report.write(f"Final sequence counts:\n")
        report.write(f"With only one primer pair: {one_pair_count}\n")
        report.write(f"With multiple primer pairs: {multiple_pairs_count}\n")

if __name__ == "__main__":
    primer_screening_pipeline()
```

Script S3 Python script for sequence classification of sequences that match with S. suis genome according to the BLASTn result, specific for sequences with primer set 1

```
import os
import sys
from Bio import SeqIO

# Define input and output paths
blast_result = "/local/muham020/suismetsys/trial_pipeline/blast_result_ps1.txt"
fasta_input =
"/local/muham020/suismetsys/trial_pipeline/result_1_ps1_primers_screening/result_1_ps1_one_primers_p
air.fasta"
output_dir = "/local/muham020/suismetsys/trial_pipeline/result_2_ps1_suis_screening"
output_fasta = os.path.join(output_dir, "result_2_ps1_from_suis.fasta")
discarded_fasta = os.path.join(output_dir, "discarded_ps1_not_suis.fasta")
report_file = os.path.join(output_dir, "report_2_ps1_suis_screening.txt")

# Create output directory if it doesn't exist
os.makedirs(output_dir, exist_ok=True)

# Read blast results to get matching qseqids
qseqids = set()
with open(blast_result, 'r') as blast_file:
    for line in blast_file:
        qseqid = line.split()[0]  # Extract qseqid
        qseqids.add(qseqid)

# Count lines, words, and bytes in the BLAST result file
blast_wc = os.popen(f'wc {blast_result}').read().strip()

# Read input fasta and separate sequences
matched_seqs = []
discarded_seqs = []
total_sequences = 0

with open(fasta_input, 'r') as fasta_file:
    for record in SeqIO.parse(fasta_file, "fasta"):
        total_sequences += 1
        if record.id in qseqids:
            matched_seqs.append(record)
        else:
            discarded_seqs.append(record)

# Write output fasta files
SeqIO.write(matched_seqs, output_fasta, "fasta")
SeqIO.write(discarded_seqs, discarded_fasta, "fasta")

# Count sequences in output files
```

```python
matched_count = len(matched_seqs)
discarded_count = len(discarded_seqs)

# Write report
with open(report_file, 'w') as report:
    report.write(f"BLAST result file details: {blast_wc}\n")
    report.write(f"Total sequences in input FASTA file: {total_sequences}\n")
    report.write(f"Sequences in output (matched): {matched_count}\n")
    report.write(f"Sequences in output (discarded): {discarded_count}\n")
    report.write(f"Sum of matched and discarded sequences: {matched_count + discarded_count}\n")

# Confirmations
assert total_sequences == matched_count + discarded_count, "Sequence count mismatch in input and
outputs."
assert matched_count == int(blast_wc.split()[0]), "Matched sequence count mismatch with BLAST result
lines."

print("BLAST result file details:", blast_wc)
print("Total sequences in input FASTA file:", total_sequences)
print("Sequences in output (matched):", matched_count)
print("Sequences in output (discarded):", discarded_count)
print("Sum of matched and discarded sequences:", matched_count + discarded_count)
print("Sequence count verified. Script execution complete.")
```

Script S4 Python script for size classification of sequences with primer set 1

```
import os
from Bio import SeqIO

# Define file paths and directories
input_fasta =
'/local/muham020/suismetsys/trial_pipeline/result_2_ps1_suis_screening/result_2_ps1_from_suis.fasta'
output_dir = '/local/muham020/suismetsys/trial_pipeline/size_classified_ps1'

# Create the output directory if it does not exist
if not os.path.exists(output_dir):
    os.makedirs(output_dir)

# Define output file paths for each size category
output_files = {
    'ST500': os.path.join(output_dir, 'ps1_suis_ST_500.fasta'),  # < 500 bp
    '500-1000': os.path.join(output_dir, 'ps1_suis_500-1000.fasta'),  # 500 to 1000 bp
    '1000-2000': os.path.join(output_dir, 'ps1_suis_1000-2000.fasta'),  # 1000 to 2000 bp
    '2000-4000': os.path.join(output_dir, 'ps1_suis_2000-4000.fasta'),  # 2000 to 4000 bp
    'BT4000': os.path.join(output_dir, 'ps1_suis_BT4000.fasta')  # > 4000 bp
}

# Note file to store logs
note_file_path = os.path.join(output_dir, 'note_ps1_size_classification.txt')
note_file = open(note_file_path, 'w')

# Open output files for writing
output_handles = {size: open(path, 'w') for size, path in output_files.items()}

# Size classification function
def classify_sequence_by_size(seq_length):
    """Classify the sequence into a size category."""
    if seq_length < 500:
        return 'ST500'
    elif 500 <= seq_length < 1000:
        return '500-1000'
    elif 1000 <= seq_length < 2000:
        return '1000-2000'
    elif 2000 <= seq_length < 4000:
        return '2000-4000'
    else:  # seq_length >= 4000
        return 'BT4000'

# Initialize counters for input and output files
input_sequence_count = 0
output_sequence_counts = {'ST500': 0, '500-1000': 0, '1000-2000': 0, '2000-4000': 0, 'BT4000': 0}
```

```python
# Process the input FASTA file
for record in SeqIO.parse(input_fasta, 'fasta'):
    input_sequence_count += 1  # Count each sequence in the input file
    sequence_length = len(record.seq)

    # Classify the sequence into one of the size categories
    size_category = classify_sequence_by_size(sequence_length)

    # Write the sequence to the appropriate output file
    SeqIO.write(record, output_handles[size_category], 'fasta')
    output_sequence_counts[size_category] += 1  # Count the sequence for the specific size category

# Close all file handles
for handle in output_handles.values():
    handle.close()

# Print and log summary of counts
summary_lines = []
summary_lines.append("Size classification completed.")
summary_lines.append(f"Input file contained {input_sequence_count} sequences.")
for size_category, count in output_sequence_counts.items():
    summary_lines.append(f"{size_category}: {count} sequences written to {output_files[size_category]}")

# Verify that the total count of output sequences matches the input count
total_output_count = sum(output_sequence_counts.values())
if total_output_count == input_sequence_count:
    summary_lines.append("Success: Total output count matches input count.")
else:
    summary_lines.append(f"Warning: Total output count ({total_output_count}) does not match input count ({input_sequence_count}).")

summary_lines.append(f"Classified sequences can be found in the directory: {output_dir}")

# Print and write summary to note file
for line in summary_lines:
    print(line)
    note_file.write(line + '\n')

# Close the note file
note_file.close()
```

Script S5 Python script for sequence classification of those sequences that match for each phase variant, according to the BLASTn result, specifically for sequences with primer set 1

```
import os
from Bio import SeqIO
from contextlib import redirect_stdout

# File paths and directories
base_dir = "/local/muham020/suismetsys/trial_pipeline"

# Input files
input_fasta_file_1 = os.path.join(base_dir, "size_classified_ps1", "ps1_suis_1000-2000.fasta")
blast_output_file_1 = os.path.join(base_dir, "blast_result_ps1_1000-2000.txt")
input_fasta_file_2 = os.path.join(base_dir, "size_classified_ps1", "ps1_suis_2000-4000.fasta")
blast_output_file_2 = os.path.join(base_dir, "blast_result_ps1_2000-4000.txt")
input_fasta_file_3 = os.path.join(base_dir, "size_classified_ps1", "ps1_suis_500-1000.fasta")
blast_output_file_3 = os.path.join(base_dir, "blast_result_ps1_500-1000.txt")
input_fasta_file_4 = os.path.join(base_dir, "size_classified_ps1", "ps1_suis_BT4000.fasta")
blast_output_file_4 = os.path.join(base_dir, "blast_result_ps1_BT4000.txt")

# Output directory
output_dir = os.path.join(base_dir, "result_3_ps1_hsdS_classified")

# Output files for first classification
output_allele_files_1 = {
    "A": os.path.join(output_dir, "result_3_ps1_hsdS_A_1000-2000.fasta"),
    "B": os.path.join(output_dir, "result_3_ps1_hsdS_B_1000-2000.fasta"),
    "C": os.path.join(output_dir, "result_3_ps1_hsdS_C_1000-2000.fasta"),
    "D": os.path.join(output_dir, "result_3_ps1_hsdS_D_1000-2000.fasta"),
}

# Output files for second classification
output_allele_files_2 = {
    "A_D": os.path.join(output_dir, "result_3_ps1_hsdS_AD_2000-4000.fasta"),
    "B_C": os.path.join(output_dir, "result_3_ps1_hsdS_BC_2000-4000.fasta"),
    "C_B": os.path.join(output_dir, "result_3_ps1_hsdS_CB_2000-4000.fasta"),
    "D_A": os.path.join(output_dir, "result_3_ps1_hsdS_DA_2000-4000.fasta"),
}

# Output files for third classification
output_allele_files_3 = {
    "A": os.path.join(output_dir, "result_3_ps1_hsdS_A_500-1000.fasta"),
    "B": os.path.join(output_dir, "result_3_ps1_hsdS_B_500-1000.fasta"),
    "C": os.path.join(output_dir, "result_3_ps1_hsdS_C_500-1000.fasta"),
    "D": os.path.join(output_dir, "result_3_ps1_hsdS_D_500-1000.fasta"),
}

# Output files for fourth classification
```

```
output_allele_files_4 = {
    "A_D": os.path.join(output_dir, "result_3_ps1_hsdS_AD_BT4000.fasta"),
    "B_C": os.path.join(output_dir, "result_3_ps1_hsdS_BC_BT4000.fasta"),
    "C_B": os.path.join(output_dir, "result_3_ps1_hsdS_CB_BT4000.fasta"),
    "D_A": os.path.join(output_dir, "result_3_ps1_hsdS_DA_BT4000.fasta"),
}

# Discarded sequences
output_discarded_file = os.path.join(output_dir, "discarded_ps1_not_hsdS.fasta")
report_file = os.path.join(output_dir, "report_ps1_hsdS_classification.txt")

# Step 1: Create output directory
os.makedirs(output_dir, exist_ok=True)

# Redirect all terminal output to report file
with open(report_file, "a") as notes_file, redirect_stdout(notes_file):

    def process_classification(input_fasta, blast_output, allele_files, discarded_file, allele_map):
        # Parse BLAST output
        blast_matches = {}
        with open(blast_output, "r") as blast_file:
            for line in blast_file:
                parts = line.strip().split()
                if len(parts) >= 2:
                    sequence_id = parts[0]
                    allele = parts[1]
                    blast_matches[sequence_id] = allele

        # Read input FASTA and filter sequences
        initial_sequence_count = 0
        allele_sequences = {key: [] for key in allele_files.keys()}
        discarded_sequences = []

        for record in SeqIO.parse(input_fasta, "fasta"):
            initial_sequence_count += 1
            seq_id = record.id
            if seq_id in blast_matches:
                allele = blast_matches[seq_id]
                if allele in allele_map:
                    allele_sequences[allele_map[allele]].append(record)
            else:
                discarded_sequences.append(record)

        # Write output files
        allele_counts = {}
        for allele, records in allele_sequences.items():
            SeqIO.write(records, allele_files[allele], "fasta")
            allele_counts[allele] = len(records)
```

```
        SeqIO.write(discarded_sequences, discarded_file, "fasta")

        # Report
        print(f"Input FASTA file: {input_fasta}")
        print(f"Total sequences in input file: {initial_sequence_count}")
        total_matched = sum(allele_counts.values())
        print(f"Total matching sequences: {total_matched}")
        for allele, count in allele_counts.items():
            print(f"Sequences matching allele {allele}: {count}")
        print(f"Total unmatched sequences: {len(discarded_sequences)}")

    # Classification for all inputs
    process_classification(input_fasta_file_1, blast_output_file_1, output_allele_files_1,
output_discarded_file, {"A": "A", "B": "B", "C": "C", "D": "D"})
    process_classification(input_fasta_file_2, blast_output_file_2, output_allele_files_2,
output_discarded_file, {"A_D": "A_D", "B_C": "B_C", "C_B": "C_B", "D_A": "D_A"})
    process_classification(input_fasta_file_3, blast_output_file_3, output_allele_files_3,
output_discarded_file, {"A": "A", "B": "B", "C": "C", "D": "D"})
    process_classification(input_fasta_file_4, blast_output_file_4, output_allele_files_4,
output_discarded_file, {"A_D": "A_D", "B_C": "B_C", "C_B": "C_B", "D_A": "D_A"})

    print("\nAll output files have been generated successfully!")
```

Script S6 Python script for exact-match barcode demultiplexing, specifically for sequences with primer set 1

```
import os
from Bio import SeqIO
from Bio.Seq import Seq

# Define file paths
working_dir = "/local/muham020/suismetsys/trial_pipeline"
compiled_file = os.path.join(working_dir, "result_3_ps1_hsdS_classified",
"compiled_result_3_ps1_hsdS_classified.fasta")
barcode_file = os.path.join(working_dir, "BC1-30.fasta")
discarded_file = os.path.join(working_dir, "discarded_no_fwd_ps1.fasta")
output_dir = os.path.join(working_dir, "result_4_ps1_bc_demultiplexing")
report_file = os.path.join(output_dir, "report_4_ps1_bc_demultiplexing.txt")

# Ensure output directory exists
os.makedirs(output_dir, exist_ok=True)

# Define forward primer
forward_primer = "CCTGAACAACTGAAAGCAAG"

# Load barcode sequences
barcode_sequences = {record.id: str(record.seq) for record in SeqIO.parse(barcode_file, "fasta")}

# Create empty output files for each barcode
for bc_id in barcode_sequences:
    output_file = os.path.join(output_dir, f"with_{bc_id}_ps1.fasta")
    with open(output_file, "w") as of:
        pass  # Create empty file

# Count initial sequences
total_sequences = 0
sequences_with_fwd_primer = 0
sequences_to_discard = 0
intermediate_sequences = []

# Step 1: Initial counting and forward primer scanning
for record in SeqIO.parse(compiled_file, "fasta"):
    total_sequences += 1
    seq = str(record.seq)

    # Check if forward primer is in first 200 bases
    if forward_primer in seq[:200]:
        sequences_with_fwd_primer += 1
        intermediate_sequences.append(record)
    else:
        # Check for forward primer in the last 200 bases (reversed)
        if forward_primer in seq[-200:][::-1]:
            sequences_with_fwd_primer += 1
```

```
            record.seq = Seq(seq[::-1])  # Reverse the sequence
            intermediate_sequences.append(record)
        else:
            # Check for reverse complement of the forward primer in first 200 or last 200 bases
            reverse_comp_primer = str(Seq(forward_primer).reverse_complement())
            if reverse_comp_primer in seq[:200] or reverse_comp_primer in seq[-200:]:
                sequences_with_fwd_primer += 1
                record.seq = record.seq.reverse_complement()  # Reverse complement the sequence
                intermediate_sequences.append(record)
            else:
                # Discard sequence if no primer found
                sequences_to_discard += 1
                with open(discarded_file, "a") as df:
                    SeqIO.write(record, df, "fasta")

# Check if total sequence counts match
assert total_sequences == sequences_with_fwd_primer + sequences_to_discard, "Mismatch in sequence
counts after primer scanning"

# Step 2: Barcode scanning and demultiplexing
barcode_counts = {bc_id: 0 for bc_id in barcode_sequences}
discarded_barcodes = 0

for record in intermediate_sequences:
    seq = str(record.seq)

    # Read from start of the sequence to the 3' end of the forward primer
    primer_end_index = seq.find(forward_primer) + len(forward_primer)
    if primer_end_index < len(seq):
        sequence_to_scan = seq[:primer_end_index]
    else:
        sequence_to_scan = seq  # In case the forward primer spans the whole sequence

    barcode_found = False
    for bc_id, bc_seq in barcode_sequences.items():
        if bc_seq in sequence_to_scan:
            barcode_counts[bc_id] += 1
            output_file = os.path.join(output_dir, f"with_{bc_id}_ps1.fasta")
            with open(output_file, "a") as of:
                SeqIO.write(record, of, "fasta")
            barcode_found = True
            break

    if not barcode_found:
        discarded_barcodes += 1
        no_bc_file = os.path.join(output_dir, "no_BC_ps1.fasta")
        with open(no_bc_file, "a") as nbf:
            SeqIO.write(record, nbf, "fasta")
```

```
# Verify if total sequences match
final_total = sum(barcode_counts.values()) + discarded_barcodes
assert final_total == total_sequences, "Mismatch in total sequence counts after barcode
demultiplexing"

# Report counts
with open(report_file, "w") as rf:
    rf.write(f"Initial total sequences: {total_sequences}\n")
    rf.write(f"Sequences with forward primer: {sequences_with_fwd_primer}\n")
    rf.write(f"Sequences discarded (no forward primer found): {sequences_to_discard}\n")
    rf.write(f"Sequences discarded (no barcode found): {discarded_barcodes}\n")
    for bc_id, count in barcode_counts.items():
        rf.write(f"Sequences with barcode {bc_id}: {count}\n")
    rf.write(f"Total sequences after barcode demultiplexing: {final_total}\n")

# Print report summary in terminal
print("Report written to:", report_file)
```

Script S7 Python script for barcode demultiplexing with 2 mismatches allowed as the second round of demultiplexing, specifically for sequences with primer set 1

```
import os
from Bio import SeqIO
from Bio.Seq import Seq
from difflib import SequenceMatcher

# Define file paths
working_dir = "/local/muham020/suismetsys/trial_pipeline"
input_file = os.path.join(working_dir, "result_4_ps1_bc_demultiplexing", "no_BC_ps1.fasta")
barcode_file = os.path.join(working_dir, "BC1-30.fasta")
discarded_file = os.path.join(working_dir, "discarded_no_fwd_ps1_mutated.fasta")
output_dir = os.path.join(working_dir, "result_4_ps1_bc_demultiplexing_mutated")
report_file = os.path.join(output_dir, "report_4_ps1_bc_demultiplexing_mutated.txt")

# Ensure output directory exists
os.makedirs(output_dir, exist_ok=True)

# Define forward primer
forward_primer = "CCTGAACAACTGAAAGCAAG"

# Load barcode sequences
barcode_sequences = {record.id: str(record.seq) for record in SeqIO.parse(barcode_file, "fasta")}

# Create empty output files for each barcode and mutated versions
for bc_id in barcode_sequences:
    output_file = os.path.join(output_dir, f"with_{bc_id}_ps1_mutated.fasta")
    with open(output_file, "w") as of:
        pass  # Create empty file

# Count initial sequences
total_sequences = 0
sequences_with_fwd_primer = 0
sequences_to_discard = 0
intermediate_sequences = []

# Function to check similarity with tolerance for mutations
def is_similar(seq1, seq2, max_diff=2):
    matcher = SequenceMatcher(None, seq1, seq2)
    match_blocks = matcher.get_matching_blocks()
    mismatches = len(seq1) + len(seq2) - 2 * sum([block.size for block in match_blocks])
    return mismatches <= max_diff

# Step 1: Primer scanning
for record in SeqIO.parse(input_file, "fasta"):
    total_sequences += 1
    seq = str(record.seq)
```

```python
    # Check if forward primer is in first 200 bases
    if forward_primer in seq[:200]:
        sequences_with_fwd_primer += 1
        intermediate_sequences.append(record)
    else:
        # Check for reverse complement of the forward primer
        reverse_comp_primer = str(Seq(forward_primer).reverse_complement())
        if reverse_comp_primer in seq[-200:]:
            sequences_with_fwd_primer += 1
            record.seq = record.seq.reverse_complement()
            intermediate_sequences.append(record)
        else:
            sequences_to_discard += 1
            with open(discarded_file, "a") as df:
                SeqIO.write(record, df, "fasta")

# Barcode scanning and demultiplexing
barcode_counts = {bc_id: 0 for bc_id in barcode_sequences}
discarded_barcodes = 0

for record in intermediate_sequences:
    seq = str(record.seq)

    # Extract sequence region to scan for barcode
    primer_end_index = seq.find(forward_primer) + len(forward_primer)
    if primer_end_index < len(seq):
        sequence_to_scan = seq[:primer_end_index]
    else:
        sequence_to_scan = seq

    barcode_found = False
    for bc_id, bc_seq in barcode_sequences.items():
        # Check exact match first
        if bc_seq in sequence_to_scan:
            barcode_counts[bc_id] += 1
            output_file = os.path.join(output_dir, f"with_{bc_id}_ps1_mutated.fasta")
            with open(output_file, "a") as of:
                SeqIO.write(record, of, "fasta")
            barcode_found = True
            break
        # Check for mutations with tolerance
        else:
            for i in range(len(sequence_to_scan) - len(bc_seq) + 1):
                sub_seq = sequence_to_scan[i:i+len(bc_seq)]
                if is_similar(sub_seq, bc_seq, max_diff=2):
                    barcode_counts[bc_id] += 1
                    output_file = os.path.join(output_dir, f"with_{bc_id}_ps1_mutated.fasta")
```

```
                    with open(output_file, "a") as of:
                        SeqIO.write(record, of, "fasta")
                    barcode_found = True
                    break
        if barcode_found:
            break

    if not barcode_found:
        discarded_barcodes += 1
        no_bc_file = os.path.join(output_dir, "no_BC_ps1_mutated.fasta")
        with open(no_bc_file, "a") as nbf:
            SeqIO.write(record, nbf, "fasta")

# Verify sequence counts
final_total = sum(barcode_counts.values()) + discarded_barcodes
assert final_total == total_sequences, "Mismatch in total sequence counts after barcode
demultiplexing"

# Report counts
with open(report_file, "w") as rf:
    rf.write(f"Initial total sequences: {total_sequences}\n")
    rf.write(f"Sequences with forward primer: {sequences_with_fwd_primer}\n")
    rf.write(f"Sequences discarded (no forward primer found): {sequences_to_discard}\n")
    rf.write(f"Sequences discarded (no barcode found): {discarded_barcodes}\n")
    for bc_id, count in barcode_counts.items():
        rf.write(f"Sequences with barcode {bc_id}: {count}\n")
    rf.write(f"Total sequences after barcode demultiplexing: {final_total}\n")

# Print report summary in terminal
print("Report written to:", report_file)
```

Script S8 Python script for isolates classification based on barcodes (from healthy (H) or symptomatic (S) pigs), specifically for sequences with primer set 1

```python
import os
import shutil

# Define directories
base_dir = '/local/muham020/suismetsys/trial_pipeline'
input_dir = os.path.join(base_dir, 'result_4_ps1_bc_demultiplexing')
input_mutated_dir = os.path.join(base_dir, 'result_4_ps1_bc_demultiplexing_mutated')
output_dir = os.path.join(base_dir, 'result_5_ps1_healthy_sick_classification')
healthy_dir = os.path.join(output_dir, 'from_healthy_pigs')
sick_dir = os.path.join(output_dir, 'from_sick_pigs')

# Create directories if they do not exist
os.makedirs(healthy_dir, exist_ok=True)
os.makedirs(sick_dir, exist_ok=True)

# Classification mapping
healthy_barcodes = {1, 3, 4, 6, 7, 8, 9, 10, 12, 13, 14, 15, 28, 29, 30}
sick_barcodes = {2, 5, 11, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27}

# Function to count sequences in a FASTA file
def count_sequences(file_path):
    count = 0
    with open(file_path, 'r') as f:
        for line in f:
            if line.startswith('>'):
                count += 1
    return count

# Initialize reports
healthy_report = []
sick_report = []
healthy_file_counts = {}
sick_file_counts = {}

# Function to process files
def process_files(input_folder, mutated=False):
    total_sequences = 0
    healthy_sequences = 0
    sick_sequences = 0

    for barcode in range(1, 31):
        filename = f'with_BC{barcode}_ps1.fasta'
        if mutated:
            filename = f'with_BC{barcode}_ps1_mutated.fasta'
        filepath = os.path.join(input_folder, filename)
```

```
        if os.path.exists(filepath):
            seq_count = count_sequences(filepath)
            total_sequences += seq_count

            # Classify files
            if barcode in healthy_barcodes:
                shutil.copy(filepath, healthy_dir)
                healthy_sequences += seq_count
                healthy_file_counts[barcode] = healthy_file_counts.get(barcode, 0) + seq_count
                healthy_report.append(f'{filename}: {seq_count} sequences')
            elif barcode in sick_barcodes:
                shutil.copy(filepath, sick_dir)
                sick_sequences += seq_count
                sick_file_counts[barcode] = sick_file_counts.get(barcode, 0) + seq_count
                sick_report.append(f'{filename}: {seq_count} sequences')

    return total_sequences, healthy_sequences, sick_sequences

# Process regular and mutated files
reg_total, reg_healthy, reg_sick = process_files(input_dir, mutated=False)
mut_total, mut_healthy, mut_sick = process_files(input_mutated_dir, mutated=True)

# Calculate overall totals
total_healthy = reg_healthy + mut_healthy
total_sick = reg_sick + mut_sick
total_all = total_healthy + total_sick

# Calculate percentages
healthy_percentage = (total_healthy / total_all) * 100
sick_percentage = (total_sick / total_all) * 100

# Write reports
with open(os.path.join(healthy_dir, 'report_5_ps1_from_healthy_pigs.txt'), 'w') as f:
    f.write('Healthy Pigs Report:\n')
    f.write(f'Total sequences (original files): {reg_healthy}\n')
    f.write(f'Total sequences (mutated files): {mut_healthy}\n')
    f.write(f'Overall total sequences: {total_healthy}\n')
    f.write(f'Percentage of healthy pigs: {healthy_percentage:.2f}%\n\n')
    f.write('Details per file:\n')
    for barcode, count in healthy_file_counts.items():
        percentage = (count / total_healthy) * 100
        f.write(f'Barcode {barcode}: {count} sequences ({percentage:.2f}%)\n')

with open(os.path.join(sick_dir, 'report_5_ps1_from_sick_pigs.txt'), 'w') as f:
    f.write('Sick Pigs Report:\n')
    f.write(f'Total sequences (original files): {reg_sick}\n')
    f.write(f'Total sequences (mutated files): {mut_sick}\n')
```

```
    f.write(f'Overall total sequences: {total_sick}\n')
    f.write(f'Percentage of sick pigs: {sick_percentage:.2f}%\n\n')
    f.write('Details per file:\n')
    for barcode, count in sick_file_counts.items():
        percentage = (count / total_sick) * 100
        f.write(f'Barcode {barcode}: {count} sequences ({percentage:.2f}%)\n')

print("Classification and reporting completed!")
```

Script S9 Python script for sequence ends trimming, specifically for sequences with primer set 1

```python
import os
from Bio import SeqIO
from Bio.Seq import Seq

# Define base directory
base_dir = '/local/muham020/suismetsys/trial_pipeline'
classified_dir = os.path.join(base_dir, 'result_5_ps1_healthy_sick_classification')
trimmed_dir = os.path.join(base_dir, 'result_6_ps1_trimmed_sequences')

# Define output directories
healthy_trimmed_dir = os.path.join(trimmed_dir, 'from_healthy_pigs')
sick_trimmed_dir = os.path.join(trimmed_dir, 'from_sick_pigs')

# Create directories if not exist
os.makedirs(healthy_trimmed_dir, exist_ok=True)
os.makedirs(sick_trimmed_dir, exist_ok=True)

# Define primers
forward_primer = "CCTGAACAACTGAAAGCAAG"
reverse_primer = "GCAGGTTGGGATTAAGGC"

# Initialize reports
healthy_report = []
sick_report = []

# Function to trim sequences
def trim_sequences(input_file, output_file):
    trimmed_count = 0
    total_count = 0
    with open(output_file, "w") as of:
        for record in SeqIO.parse(input_file, "fasta"):
            total_count += 1
            seq = str(record.seq)
            fwd_start = seq.find(forward_primer) + len(forward_primer)
            rev_end = seq.rfind(reverse_primer)
            if fwd_start < rev_end:  # Trim only if reverse primer is found after forward primer
                record.seq = Seq(seq[fwd_start:rev_end])
                trimmed_count += 1
            SeqIO.write(record, of, "fasta")
    return total_count, trimmed_count

# Process files in healthy and sick directories
def process_files(category, source_dir, output_dir, report):
    file_data = []
    for filename in os.listdir(source_dir):
        if filename.startswith("with_BC") and (filename.endswith("_ps1.fasta") or
filename.endswith("_ps1_mutated.fasta")):
```

```python
            input_file = os.path.join(source_dir, filename)
            trimmed_file = filename.replace(".fasta", "_trimmed.fasta")
            output_file = os.path.join(output_dir, trimmed_file)

            # Trim sequences
            total_sequences, trimmed_count = trim_sequences(input_file, output_file)

            # Collect details
            file_data.append((filename, total_sequences, trimmed_count))

    # Sort data by barcode number
    file_data.sort(key=lambda x: int(x[0].split('_')[1][2:]))

    # Append to report
    for data in file_data:
        report.append(f"{data[0]}: {data[2]}/{data[1]} sequences trimmed")

# Process healthy and sick files
process_files('healthy', os.path.join(classified_dir, 'from_healthy_pigs'), healthy_trimmed_dir,
healthy_report)
process_files('sick', os.path.join(classified_dir, 'from_sick_pigs'), sick_trimmed_dir, sick_report)

# Write reports
with open(os.path.join(trimmed_dir, 'report_6_ps1_sequence_trimming.txt'), 'w') as f:
    f.write("Sequence Trimming Report\n\n")
    f.write(f"Primers Used:\nForward: {forward_primer}\nReverse: {reverse_primer}\n\n")

    f.write("Healthy Pigs Report:\n")
    f.write('\n'.join(healthy_report))
    f.write('\n\n')

    f.write("Sick Pigs Report:\n")
    f.write('\n'.join(sick_report))

print("Trimming completed and reports generated!")
```

```python
import os
from Bio import SeqIO
from contextlib import redirect_stdout

# File paths and directories
base_dir = "/local/muham020/suismetsys/trial_pipeline"
input_dir = os.path.join(base_dir, "result_6_ps1_trimmed_sequences/from_healthy_pigs")
blast_dir = os.path.join(base_dir, "blast_results")
output_base_dir = os.path.join(base_dir, "result_7_ps1_final_hsdS_classification/from_healthy_pigs")
report_file = os.path.join(base_dir,
"result_7_ps1_final_hsdS_classification/report_7_ps1_final_hsdS_classification_healthy.txt")

# Ensure base output directory exists
os.makedirs(output_base_dir, exist_ok=True)

# Blast files
blast_files = [
    ("500-1000", os.path.join(blast_dir, "blast_result_ps1_500-1000.txt")),
    ("1000-2000", os.path.join(blast_dir, "blast_result_ps1_1000-2000.txt")),
    ("2000-4000", os.path.join(blast_dir, "blast_result_ps1_2000-4000.txt")),
    ("BT4000", os.path.join(blast_dir, "blast_result_ps1_BT4000.txt"))
]

# Initialize counters
total_input_sequences = 0
total_output_sequences = 0
output_file_counts = {}
skipped_files = []
input_file_counts = {}
allele_counts = {"A": 0, "B": 0, "C": 0, "D": 0}

# Redirect output to report file
with open(report_file, "w") as report, redirect_stdout(report):

    for file_name in sorted(os.listdir(input_dir)):
        if not file_name.endswith(".fasta"):
            continue

        # Determine BC number and type (mutated/normal)
        bc_number = file_name.split("_")[1][2:]
        mutated_status = "mutated" if "mutated" in file_name else "normal"

        input_file = os.path.join(input_dir, file_name)
        sequences = list(SeqIO.parse(input_file, "fasta"))
        input_count = len(sequences)
        total_input_sequences += input_count
```

```
        if input_count == 0:
            skipped_files.append(file_name)
            continue
        input_file_counts[file_name] = input_count

        # Process each sequence
        for seq_record in sequences:
            seq_id = seq_record.id
            found = False

            for blast_range, blast_file in blast_files:
                with open(blast_file, "r") as blast:
                    for line in blast:
                        parts = line.strip().split()
                        if len(parts) < 2:
                            continue

                        blast_seq_id = parts[0]
                        allele = parts[1]  # Second column for allele

                        if seq_id == blast_seq_id:
                            found = True

                            # Create output directory if it does not exist
                            output_dir = os.path.join(output_base_dir,
f"with_BC{bc_number}_{mutated_status}_ps1")
                            os.makedirs(output_dir, exist_ok=True)

                            # Create output file
                            output_file = os.path.join(output_dir,
f"hsdS_{allele}_{blast_range}.fasta")
                            with open(output_file, "a") as out_fasta:
                                SeqIO.write(seq_record, out_fasta, "fasta")

                            # Count sequences per output file
                            if output_file not in output_file_counts:
                                output_file_counts[output_file] = 0
                            output_file_counts[output_file] += 1

                            # Count allele types
                            if allele in ["A", "A_D"]:
                                allele_counts["A"] += 1
                            elif allele in ["B", "B_C"]:
                                allele_counts["B"] += 1
                            elif allele in ["C", "C_B"]:
                                allele_counts["C"] += 1
                            elif allele in ["D", "D_A"]:
                                allele_counts["D"] += 1
```

```python
                            total_output_sequences += 1
                            break
                if found:
                    break

    # Final report summary
    print("Input File Summary:")
    for file in sorted(input_file_counts.keys()):
        print(f"Sequences found in {file} = {input_file_counts[file]}")

    print("\nSkipped files (no sequences):")
    for file in sorted(skipped_files):
        print(f"{file}")

    print(f"\nTotal sequence in input fasta files: {total_input_sequences}")

    print("\nOutput Files:")
    for output_dir, _, files in sorted(os.walk(output_base_dir)):
        if files:
            print(f"Directory {output_dir}")
            for file in sorted(files):
                output_path = os.path.join(output_dir, file)
                count = output_file_counts.get(output_path, 0)
                print(f"Sequences found in {file} = {count}")

    print(f"\nTotal sequences written to output files: {total_output_sequences}")

    # Allele statistics
    print("\nAllele Statistics:")
    for allele, count in allele_counts.items():
        percentage = (count / total_output_sequences * 100) if total_output_sequences > 0 else 0
        print(f"Allele {allele}: {count} sequences ({percentage:.2f}%)")

print("Processing completed. Check the report file for details.")
```

Script S10 Python script for generation of table (.csv) out of the report from the final *hsdS* classification for all primer sets

```python
import pandas as pd
import os
import re

def parse_report_and_generate_csv(report_path, output_dir, output_file_name):
    with open(report_path, 'r') as report:
        lines = report.readlines()

    # Initialize variables
    data = []
    barcode_totals = {}  # Total counts per barcode
    allele_totals = {'A': 0, 'B': 0, 'C': 0, 'D': 0}  # Global totals for alleles
    barcode_allele_counts = {}  # To store allele counts per barcode

    # Extract primer set and source from the file name
    primer_set = ""
    source = ""

    if 'ps1' in report_path:
        primer_set = 'ps1'
    elif 'ps2' in report_path:
        primer_set = 'ps2'
    elif 'ps3' in report_path:
        primer_set = 'ps3'

    if 'healthy' in report_path:
        source = 'H'
    elif 'sick' in report_path:
        source = 'S'

    current_barcode = None

    # Parse the report line by line
    for line in lines:
        line = line.strip()

        # Identify directories for barcodes
        dir_match = re.match(r'Directory .*/with_(BC\d+)_.*', line)
        if dir_match:
            current_barcode = int(dir_match.group(1)[2:])  # Extract BC number
            if current_barcode not in barcode_allele_counts:
                barcode_allele_counts[current_barcode] = {'A': 0, 'B': 0, 'C': 0, 'D': 0}

        # Extract sequence counts from output files
        seq_match = re.match(r'Sequences found in hsdS_([A-D]_.*)\.fasta = (\d+)', line)
        if seq_match and current_barcode is not None:
            allele = seq_match.group(1)[0]  # Extract the first letter for allele
```

```
            count = int(seq_match.group(2))

            # Add to the barcode-specific and global allele counts
            barcode_allele_counts[current_barcode][allele] += count
            allele_totals[allele] += count

    # Process data into rows for the output
    global_total_sequences = sum(allele_totals.values())
    for barcode in sorted(barcode_allele_counts.keys()):
        barcode_total = sum(barcode_allele_counts[barcode].values())

        for allele in ['A', 'B', 'C', 'D']:
            count = barcode_allele_counts[barcode].get(allele, 0)
            within_group_freq = (count / barcode_total * 100) if barcode_total > 0 else 0
            global_freq = (count / global_total_sequences * 100) if global_total_sequences > 0 else
0


            data.append({
                'No.': len(data) + 1,
                'Primer Set': primer_set,
                'Source': source,
                'Barcode': barcode,
                'Allele': allele,
                'Sequence Count': count,
                'Within Group Frequency (%)': round(within_group_freq, 2),
                'Global Frequency (%)': round(global_freq, 2)
            })

    # Convert to DataFrame
    df = pd.DataFrame(data)

    # Ensure output directory exists
    os.makedirs(output_dir, exist_ok=True)

    # Save the CSV
    output_file_path = os.path.join(output_dir, output_file_name)
    df.to_csv(output_file_path, index=False)

    print(f"CSV file has been generated and saved to: {output_file_path}")

if __name__ == "__main__":
    # Define paths and process multiple report files
    base_dir = '/local/muham020/suismetsys/trial_pipeline'
    report_files = [
        'result_7_ps1_final_hsdS_classification/report_7_ps1_final_hsdS_classification_healthy.txt',
        'result_7_ps1_final_hsdS_classification/report_7_ps1_final_hsdS_classification_sick.txt',
        'result_7_ps2_final_hsdS_classification/report_7_ps2_final_hsdS_classification_healthy.txt',
        'result_7_ps2_final_hsdS_classification/report_7_ps2_final_hsdS_classification_sick.txt',
```

```python
    'result_7_ps3_final_hsdS_classification/report_7_ps3_final_hsdS_classification_healthy.txt',
    'result_7_ps3_final_hsdS_classification/report_7_ps3_final_hsdS_classification_sick.txt'
]

output_dir = os.path.join(base_dir, 'result_8_statistical_analysis')
os.makedirs(output_dir, exist_ok=True)

for report in report_files:
    report_path = os.path.join(base_dir, report)
    output_file_name = f"table_{report.split('/')[-1].replace('.txt', '.csv')}"
    parse_report_and_generate_csv(report_path, output_dir, output_file_name)
```

Script S11 Python script to combine all .csv files that have been generated into a single integrated table (.csv)

```python
import pandas as pd
import os

def remove_empty_rows(input_dir, output_dir):
    # List all CSV files in the directory
    for file_name in os.listdir(input_dir):
        if file_name.endswith(".csv"):
            input_file = os.path.join(input_dir, file_name)
            output_file = os.path.join(output_dir, f"table_essential_{file_name}")

            # Read the CSV file
            df = pd.read_csv(input_file)

            # Remove rows where Sequence Count is 0
            df_filtered = df[df['Sequence Count'] > 0]

            # Save the filtered CSV
            df_filtered.to_csv(output_file, index=False)
            print(f"Filtered file saved to: {output_file}")

def combine_specific_csv_files(input_dir, output_file):
    specific_files = [
        'table_report_7_ps1_final_hsdS_classification_healthy.csv',
        'table_report_7_ps1_final_hsdS_classification_sick.csv',
        'table_report_7_ps2_final_hsdS_classification_healthy.csv',
        'table_report_7_ps2_final_hsdS_classification_sick.csv',
        'table_report_7_ps3_final_hsdS_classification_healthy.csv',
        'table_report_7_ps3_final_hsdS_classification_sick.csv'
    ]

    combined_data = []

    # Iterate through the specific input files
    for file_name in specific_files:
        input_file = os.path.join(input_dir, file_name)
        if os.path.exists(input_file):
            # Read the CSV and append its content
            df = pd.read_csv(input_file)
            if 'No.' in df.columns:
                df = df.drop(columns=['No.'])  # Remove existing 'No.' column if it exists
            if not df.empty:
                combined_data.append(df)

    # Check if there is data to combine
    if combined_data:
        # Combine all data into one DataFrame
        combined_df = pd.concat(combined_data, ignore_index=True)
```

```
        # Add "No." column with sequential positive integers
        combined_df.insert(0, 'No.', range(1, len(combined_df) + 1))

        # Save the combined CSV
        combined_df.to_csv(output_file, index=False)
        print(f"Combined file saved to: {output_file}")
    else:
        print("No valid data to combine.")


if __name__ == "__main__":
    base_dir =
'/local/muham020/suismetsys/trial_pipeline/result_8_statistical_analysis/8.1_hsdS_allele_distributio
n_table'

    # Directories
    input_dir = base_dir
    output_dir = base_dir

    # Task 1: Remove rows with Sequence Count = 0
    # Remove empty rows task retained for continuity
    remove_empty_rows(input_dir, output_dir)

    # Task 2: Combine specific CSV files
    combined_output_file = os.path.join(output_dir, 'result_8.1_table_combined.csv')
    combine_specific_csv_files(input_dir, combined_output_file)
```

Script S12 Python script to generate two .csv files: one to visualize *hsdS* allele distribution across isolate source, and the other one across primer sets

```
import pandas as pd
import os
import numpy as np

def generate_hsds_distribution_tables(input_file, output_dir):
    # Read the input CSV file
    df = pd.read_csv(input_file)

    # Initialize lists to store rows for the output CSV files
    source_distribution = []
    primers_distribution = []

    # Helper function to calculate metrics
    def calculate_metrics(group, allele):
        allele_group = group[group['Allele'] == allele]
        non_zero_barcodes = allele_group[allele_group['Sequence Count'] > 0]['Barcode'].unique()
        barcode_count = len(non_zero_barcodes)
        total_sequence_count = allele_group['Sequence Count'].sum()
        mean_count = total_sequence_count / barcode_count if barcode_count > 0 else 0
        median_count = allele_group['Sequence Count'].median() if barcode_count > 0 else 0
        stdev_count = allele_group['Sequence Count'].std() if barcode_count > 0 else 0
        variance_count = stdev_count ** 2  # Calculate variance as the square of the standard
deviation

        return {
            'Sequence Count': total_sequence_count,
            'Frequency (%)': round((total_sequence_count / group['Sequence Count'].sum()) * 100, 2)
if group['Sequence Count'].sum() > 0 else 0,
            'Barcode Count': barcode_count,
            'Mean': round(mean_count, 2),
            'Median': round(median_count, 2),
            'St Dev': round(stdev_count, 2),
            'Variance': round(variance_count, 2),
            'Barcode List': f"({','.join(f'BC{int(bc)}' for bc in non_zero_barcodes)})" if
barcode_count > 0 else "()"
        }

    # Group by source and allele for 8.1_table_hsdS_distribution_source.csv
    for source in ['H', 'S']:
        source_group = df[df['Source'] == source]

        for allele in ['A', 'B', 'C', 'D']:
            metrics = calculate_metrics(source_group, allele)

            source_distribution.append({
                'No.': len(source_distribution) + 1,
```

```
                'Primer Set': 'ALL',
                'Source': source,
                'Allele': allele,
                **metrics
            })

    # Group by primer set, source, and allele for 8.1_table_hsdS_distribution_primers.csv
    for primer_set in ['ps1', 'ps2', 'ps3']:
        primer_group = df[df['Primer Set'] == primer_set]

        for source in ['H', 'S']:
            source_group = primer_group[primer_group['Source'] == source]

            for allele in ['A', 'B', 'C', 'D']:
                metrics = calculate_metrics(source_group, allele)

                primers_distribution.append({
                    'No.': len(primers_distribution) + 1,
                    'Primer Set': primer_set,
                    'Source': source,
                    'Allele': allele,
                    **metrics
                })

    # Convert the results to DataFrames
    source_df = pd.DataFrame(source_distribution)
    primers_df = pd.DataFrame(primers_distribution)

    # Adjust column order
    column_order = ['No.', 'Primer Set', 'Source', 'Allele', 'Sequence Count', 'Frequency (%)',
'Barcode Count', 'Mean', 'Median', 'St Dev', 'Variance', 'Barcode List']
    source_df = source_df[column_order]
    primers_df = primers_df[column_order]

    # Save the output CSV files
    source_output_file = os.path.join(output_dir, '8.1_table_hsdS_distribution_source.csv')
    primers_output_file = os.path.join(output_dir, '8.1_table_hsdS_distribution_primers.csv')

    source_df.to_csv(source_output_file, index=False)
    primers_df.to_csv(primers_output_file, index=False)

    print(f"Source distribution table saved to: {source_output_file}")
    print(f"Primers distribution table saved to: {primers_output_file}")

if __name__ == "__main__":
    # Define input and output paths
```

```
    base_dir =
'/local/muham020/suismetsys/trial_pipeline/result_8_statistical_analysis/8.1_hsdS_allele_distributio
n_table'
    input_file = os.path.join(base_dir, 'result_8.1_table_combined.csv')
    output_dir = base_dir

    # Generate the distribution tables
    generate_hsds_distribution_tables(input_file, output_dir)
```

Script S13 Python script to generate summary .csv files that list the general hsdS phase variant preference by *S. suis*. The script is also designed to generate histogram and pie chart to visualize general hsdS phase variant distribution in *S. suis*

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np  # Importing numpy for mathematical operations

# Load the input CSV file
file_path =
'/local/muham020/suismetsys/trial_pipeline/result_8_statistical_analysis/8.1_hsdS_allele_distributio
n_table/8.1_table_hsdS_distribution_source.csv'
data = pd.read_csv(file_path)

# Define a function to calculate global statistics
def calculate_global_statistics(group):
    total_sequence_count = group['Sequence Count'].sum()
    global_frequency = (group['Sequence Count'] / total_sequence_count * 100)
    global_mean = total_sequence_count / group['Barcode Count'].sum()
    return total_sequence_count, global_frequency, global_mean

# Separate data by alleles and sources
grouped = data.groupby(['Allele', 'Source'])
result_rows = []
summary_rows = []

alleles = data['Allele'].unique()

for allele in alleles:
    h_group = data[(data['Allele'] == allele) & (data['Source'] == 'H')]
    s_group = data[(data['Allele'] == allele) & (data['Source'] == 'S')]

    # Combine values for H and S
    sequence_count_h = h_group['Sequence Count'].sum()
    sequence_count_s = s_group['Sequence Count'].sum()
    sequence_count_total = sequence_count_h + sequence_count_s

    barcode_count_h = h_group['Barcode Count'].sum()
    barcode_count_s = s_group['Barcode Count'].sum()
    barcode_count_total = barcode_count_h + barcode_count_s

    global_frequency = round((sequence_count_total / data['Sequence Count'].sum()) * 100, 2)

    barcode_list_h = ','.join(h_group['Barcode List'].values)
    barcode_list_s = ','.join(s_group['Barcode List'].values)
    barcode_list_all = ','.join([barcode_list_h, barcode_list_s])

    mean_h = round(sequence_count_h / barcode_count_h, 2) if barcode_count_h > 0 else 0
    mean_s = round(sequence_count_s / barcode_count_s, 2) if barcode_count_s > 0 else 0
```

```
    global_mean = round(sequence_count_total / barcode_count_total, 2) if barcode_count_total > 0
else 0

    median_h = round(h_group['Median'].mean(), 2)
    median_s = round(s_group['Median'].mean(), 2)
    median_global = round((median_h + median_s) / 2, 2)

    stdev_h = round(h_group['St Dev'].mean(), 2)
    stdev_s = round(s_group['St Dev'].mean(), 2)
    stdev_global = round((stdev_h + stdev_s) / 2, 2)

    variance_h = round(h_group['St Dev'].std(), 2)
    variance_s = round(s_group['St Dev'].std(), 2)
    variance_global = round((variance_h + variance_s) / 2, 2)

    # Add to the detailed result
    result_rows.append([
        len(result_rows) + 1, "ALL", "ALL", allele,
        sequence_count_h, sequence_count_s, sequence_count_total,
        round(h_group['Frequency (%)'].sum(), 2), round(s_group['Frequency (%)'].sum(), 2),
global_frequency,
        barcode_count_h, barcode_count_s, barcode_count_total,
        mean_h, mean_s, global_mean,
        median_h, median_s, median_global,
        stdev_h, stdev_s, stdev_global,
        variance_h, variance_s, variance_global,
        barcode_list_h, barcode_list_s, barcode_list_all
    ])

    # Add to the summary result
    summary_rows.append([
        len(summary_rows) + 1, "ALL", "ALL", allele,
        sequence_count_total, global_frequency,
        barcode_count_total, global_mean, median_global,
        stdev_global, variance_global, barcode_list_all
    ])

# Define column headers for both files
result_columns = [
    "No.", "Primer Set", "Source", "Allele",
    "Sequence Count H", "Sequence Count S", "Sequence Count Total",
    "Frequency H (%)", "Frequency S (%)", "Global Frequency (%)",
    "Barcode Count H", "Barcode Count S", "Barcode Count Total",
    "Mean H", "Mean S", "Global Mean",
    "Median H", "Median S", "Median Global",
    "St Dev H", "St Dev S", "St Dev Global",
    "Variance H", "Variance S", "Variance Global",
    "Barcode List H", "Barcode List S", "Barcode List All"
```

```
]

summary_columns = [
    "No.", "Primer Set", "Source", "Allele",
    "Sequence Count Total", "Global Frequency (%)",
    "Barcode Count Total", "Global Mean", "Global Median",
    "Global StDev", "Global Variance", "Barcode List All"
]

# Create DataFrames for results
result_df = pd.DataFrame(result_rows, columns=result_columns)
summary_df = pd.DataFrame(summary_rows, columns=summary_columns)

# Save to CSV
output_dir =
'/local/muham020/suismetsys/trial_pipeline/result_8_statistical_analysis/8.1_hsdS_allele_distributio
n_table/'
result_df.to_csv(output_dir + '8.3_comprehensive_allele_analysis.csv', index=False)
summary_df.to_csv(output_dir + '8.3_summary_allele_analysis.csv', index=False)

# Load the summary data for visualization
summary_data = pd.read_csv(output_dir + '8.3_summary_allele_analysis.csv')

# Prepare data for histogram and pie chart
allele_counts = summary_data['Sequence Count Total']
allele_percentages = summary_data['Global Frequency (%)']
alleles = summary_data['Allele']

# Define pastel colors
colors = sns.color_palette("pastel", len(alleles))

# Pie chart with custom label layout
fig, ax = plt.subplots(figsize=(8, 8))
wedgeprops = {"edgecolor": "black"}

def custom_label_position(wedges, counts, percentages):
    for i, wedge in enumerate(wedges):
        angle = (wedge.theta2 - wedge.theta1) / 2. + wedge.theta1
        x = np.cos(np.radians(angle))
        y = np.sin(np.radians(angle))

        if i == 1:  # Special case for allele B
            ax.text(x * 0.5, y * 0.5, f"{counts[i]} ({percentages[i]:.2f}%)",
                    ha='center', fontsize=10)
        elif i == 3:  # Special case for allele D with arrow
            connection_line_x = 1.05 * x
            connection_line_y = 1.05 * y
            label_x = 1.4 * x
```

```python
            label_y = 1.3 * y - 0.15  # Slightly lower the label and shorten the arrow
            ax.annotate(
                f"{counts[i]} ({percentages[i]:.2f}%)",
                xy=(connection_line_x, connection_line_y),
                xytext=(label_x, label_y),
                arrowprops=dict(arrowstyle="-", color='black'),
                ha='center', fontsize=10
            )
        elif i == 0:  # Adjust label for allele A slightly to the left
            ax.text(1.05 * x - 0.1, 1.05 * y, f"{counts[i]} ({percentages[i]:.2f}%)",
                    ha='center', fontsize=10)
        elif i == 2:  # Adjust label for allele C slightly to the right
            ax.text(1.05 * x + 0.1, 1.05 * y, f"{counts[i]} ({percentages[i]:.2f}%)",
                    ha='center', fontsize=10)

wedges, texts = ax.pie(
    allele_counts,
    colors=colors,
    startangle=90,
    wedgeprops=wedgeprops,
    labels=None
)

# Custom label positions
custom_label_position(wedges, allele_counts, allele_percentages)

# Add legend
plt.legend(wedges, alleles, title="Alleles", loc="center left", bbox_to_anchor=(1, 0, 0.5, 1),
fontsize=10)

plt.title("Allele Distribution (Percentage)", fontsize=14, pad=20)  # Adjusted title position
plt.savefig(output_dir + "allele_distribution_pie_chart.png")
plt.show()

# Histogram
plt.figure(figsize=(10, 6))
plt.bar(alleles, allele_counts, color=colors, edgecolor='black')
for i, (value, percentage) in enumerate(zip(allele_counts, allele_percentages)):
    plt.text(i, value + 5, f'{value} ({percentage:.2f}%)', ha='center', fontsize=10)
plt.title("Allele Distribution (Sequence Count)", fontsize=14)
plt.xlabel("Allele", fontsize=12)
plt.ylabel("Sequence Count", fontsize=12)
plt.tight_layout()
plt.savefig(output_dir + "allele_distribution_histogram.png")
plt.show()
```

Script S14 Python script to perform Shapiro-Wilk normality test for sequence count data in healthy and sick pig groups.

```
import pandas as pd
from scipy.stats import shapiro
import os

def normality_test(input_file, output_dir):
    # Read the input CSV file
    df = pd.read_csv(input_file)

    # Ensure output directory exists
    os.makedirs(output_dir, exist_ok=True)

    # Initialize a list to store results
    report_lines = []
    report_lines.append("### Shapiro-Wilk Normality Test for H and S ###\n")
    report_lines.append("Objective: To determine if the data is normally distributed for both
healthy (H) and sick (S) sources.\n")

    # Perform the test for each Source (H and S)
    normality_results = {}
    for source in ['H', 'S']:
        source_data = df[df['Source'] == source]['Frequency (%)']
        report_lines.append(f"Source {source} Analysis:")
        if len(source_data) > 2:  # Shapiro-Wilk requires at least 3 data points
            stat, p = shapiro(source_data)
            report_lines.append(f"  - Shapiro-Wilk Test Statistic (W): {stat:.4f}")
            report_lines.append(f"  - p-value: {p:.4f}")
            if p > 0.05:
                report_lines.append(f"  - Interpretation: Data is likely normally distributed for
Source {source}.")
                normality_results[source] = "normal"
            else:
                report_lines.append(f"  - Interpretation: Data is not normally distributed for
Source {source}.")
                normality_results[source] = "not normal"
        else:
            report_lines.append(f"  - Warning: Not enough data points for Shapiro-Wilk test for
Source {source}.")
            normality_results[source] = "insufficient data"
        report_lines.append("")

    # Conclusion
    report_lines.append("### Conclusion ###")
    if normality_results.get('H') == "normal" and normality_results.get('S') == "normal":
        report_lines.append("Both healthy (H) and sick (S) data are normally distributed. You may
proceed with **parametric tests**.")
    elif normality_results.get('H') == "not normal" or normality_results.get('S') == "not normal":
```

```
        report_lines.append("One or both of the groups (H or S) are not normally distributed. You 
should proceed with **non-parametric tests**.")
    else:
        report_lines.append("Insufficient data to determine normality for one or both groups. 
Proceed with caution and consider non-parametric tests as safer.")

    # Save the results to a report file
    report_file = os.path.join(output_dir, 'report_normality_test_H_S_detailed.txt')
    with open(report_file, 'w') as f:
        f.write("\n".join(report_lines))

    print(f"Normality test completed. Report saved to: {report_file}")

if __name__ == "__main__":
    base_dir = '/local/muham020/suismetsys/trial_pipeline/result_8_statistical_analysis'
    input_file = os.path.join(base_dir, 
'8.1_hsdS_allele_distribution_table/8.1_table_hsdS_distribution_source.csv')
    output_dir = os.path.join(base_dir, '8.2_normality_nonparametric_test')

    normality_test(input_file, output_dir)
```

Script S15 Python script to perform χ2 test to identify the significance of data distribution in healthy pig group and sick pig group. Visualization in the form of stacked and grouped histograms are also indicated in the script

```
import pandas as pd
import os
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import chi2_contingency

def add_bar_labels(ax, stacked=False):
    """Add bar labels to the histogram bars."""
    for p in ax.patches:
        height = p.get_height()
        width = p.get_width()
        x = p.get_x()
        y = p.get_y()

        if height > 0:
            label_color = "black"  # Set to black for all scenarios
            if stacked:
                # Place the label in the center of the stacked bar
                ax.text(
                    x + width / 2, y + height / 2,
                    f'{int(height)}',
                    ha='center', va='center', fontsize=9, color=label_color
                )
            else:
                # Place the label inside the grouped bar
                ax.text(
                    x + width / 2, y + height / 2,
                    f'{int(height)}',
                    ha='center', va='center', fontsize=9, color=label_color
                )

def run_chi_square(data, value_col):
    """Run Chi-Square Test for a specific value column."""
    contingency_table = data.pivot_table(
        index='Allele',
        columns='Source',
        values=value_col,
        aggfunc='sum',
        fill_value=0
    )
    chi2, p, dof, expected = chi2_contingency(contingency_table)
    return chi2, p, dof, expected, contingency_table

def generate_analysis(input_combined, input_distribution, output_dir):
    # Read the input CSV files
    combined_df = pd.read_csv(input_combined)
```

```
    distribution_df = pd.read_csv(input_distribution)

    # Ensure output directory exists
    os.makedirs(output_dir, exist_ok=True)

    # Run Chi-Square Tests
    chi_square_results = {}
    for value_col in ['Sequence Count', 'Barcode Count', 'Frequency (%)']:
        chi2, p, dof, expected, contingency_table = run_chi_square(distribution_df, value_col)
        chi_square_results[value_col] = {
            'chi2': chi2,
            'p': p,
            'dof': dof,
            'expected': expected,
            'contingency_table': contingency_table
        }

        # Save the results to a text file
        report_path = os.path.join(output_dir, f'chi_square_{value_col.replace(" ",
"_").lower()}_report.txt')
        with open(report_path, 'w') as f:
            f.write(f"Chi-Square Test for {value_col}:\n")
            f.write(f"Chi-Square Statistic: {chi2:.3f}\n")
            f.write(f"Degrees of Freedom: {dof}\n")
            f.write(f"P-value: {p:.3e}\n\n")
            f.write("Observed Frequencies:\n")
            f.write(contingency_table.to_string())
            f.write("\n\nExpected Frequencies:\n")
            f.write(pd.DataFrame(expected, index=contingency_table.index,
columns=contingency_table.columns).to_string())

    # Generate histograms
    for value_col, ylabel in [('Barcode Count', 'Barcode Count'),
                              ('Sequence Count', 'Sequence Count'),
                              ('Frequency (%)', 'Frequency (%)')]:
        p_value_text = f"P-value: {chi_square_results[value_col]['p']:.3e}"

        # Stacked Histogram: X = Population Group
        plt.figure(figsize=(10, 6))
        ax = sns.histplot(
            data=distribution_df,
            x='Source',
            hue='Allele',
            weights=value_col,
            multiple='stack',
            kde=False,
            shrink=0.8
        )
```

```
        add_bar_labels(ax, stacked=True)
        plt.title(f"hsdS Alleles Distribution by Population Group ({ylabel})\n{p_value_text}")
        plt.xlabel("Population Group")
        plt.ylabel(ylabel)
        stacked_path = os.path.join(output_dir, f"stacked_histogram_{value_col.replace(' ',
'_').lower()}.png")
        plt.savefig(stacked_path)
        plt.close()

        # Grouped Histogram: X = Population Group
        plt.figure(figsize=(10, 6))
        ax = sns.histplot(
            data=distribution_df,
            x='Source',
            hue='Allele',
            weights=value_col,
            multiple='dodge',
            kde=False,
            shrink=0.8
        )
        add_bar_labels(ax)
        plt.title(f"hsdS Alleles Distribution by Population Group ({ylabel})\n{p_value_text}")
        plt.xlabel("Population Group")
        plt.ylabel(ylabel)
        grouped_path = os.path.join(output_dir, f"grouped_histogram_{value_col.replace(' ',
'_').lower()}.png")
        plt.savefig(grouped_path)
        plt.close()

        # Stacked Histogram: X = Alleles
        plt.figure(figsize=(10, 6))
        ax = sns.histplot(
            data=distribution_df,
            x='Allele',
            hue='Source',
            weights=value_col,
            multiple='stack',
            kde=False,
            shrink=0.8
        )
        add_bar_labels(ax, stacked=True)
        plt.title(f"hsdS Alleles Distribution by Alleles ({ylabel})\n{p_value_text}")
        plt.xlabel("Alleles")
        plt.ylabel(ylabel)
        stacked_alleles_path = os.path.join(output_dir,
f"stacked_alleles_histogram_{value_col.replace(' ', '_').lower()}.png")
        plt.savefig(stacked_alleles_path)
        plt.close()
```

```
        # Grouped Histogram: X = Alleles
        plt.figure(figsize=(10, 6))
        ax = sns.histplot(
            data=distribution_df,
            x='Allele',
            hue='Source',
            weights=value_col,
            multiple='dodge',
            kde=False,
            shrink=0.8
        )
        add_bar_labels(ax)
        plt.title(f"hsdS Alleles Distribution by Alleles ({ylabel})\n{p_value_text}")
        plt.xlabel("Alleles")
        plt.ylabel(ylabel)
        grouped_alleles_path = os.path.join(output_dir,
f"grouped_alleles_histogram_{value_col.replace(' ', '_').lower()}.png")
        plt.savefig(grouped_alleles_path)
        plt.close()

if __name__ == "__main__":
    # Define paths
    base_dir = '/local/muham020/suismetsys/trial_pipeline/result_8_statistical_analysis'
    input_combined = os.path.join(base_dir, '8.1_hsdS_allele_distribution_table',
'result_8.1_table_combined.csv')
    input_distribution = os.path.join(base_dir, '8.1_hsdS_allele_distribution_table',
'8.1_table_hsdS_distribution_source.csv')
    output_dir = os.path.join(base_dir, '8.3_hsdS_analysis')

    # Run the analysis
    generate_analysis(input_combined, input_distribution, output_dir)
```

Script S16 Python script to perform the Mann-Whitney U test to test the significance of each phase variant distribution between healthy and sick pig groups. The script also generates a box plot and violin plot to visualize the *hsdS* phase variant distribution.

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import mannwhitneyu
import os

# Function to perform Mann-Whitney U test for each allele
def mann_whitney_test(data, value_column, output_dir):
    alleles = data['Allele'].unique()
    results = {}

    for allele in alleles:
        allele_data = data[data['Allele'] == allele]
        h_values = allele_data[allele_data['Source'] == 'H'][value_column]
        s_values = allele_data[allele_data['Source'] == 'S'][value_column]

        # Perform the test
        u_stat, p_value = mannwhitneyu(h_values, s_values, alternative='two-sided')

        # Store results
        results[allele] = {'U Statistic': u_stat, 'P-value': p_value}

        # Save detailed results to a text file
        with open(os.path.join(output_dir, f"mann_whitney_{allele}.txt"), 'w') as f:
            f.write(f"Mann-Whitney U Test Results for Allele {allele}\n")
            f.write(f"U Statistic: {u_stat:.3f}\n")
            f.write(f"P-value: {p_value:.3e}\n")
            f.write(f"H Population Median: {h_values.median()}\n")
            f.write(f"S Population Median: {s_values.median()}\n")
            f.write("\nExplanation of Analysis:\n")
            f.write("The Mann-Whitney U test is used to compare the distributions of two independent
samples (H and S populations) for the selected allele.\n")
            f.write("The null hypothesis assumes no difference between the groups, and a p-value
below 0.05 indicates a statistically significant difference.\n")

    return results

# Function to normalize sequence counts
def normalize_sequence_counts(data):
    data['Normalized Sequence Count (%)'] = (
        data.groupby('Source')['Sequence Count']
        .transform(lambda x: (x / x.sum()) * 100)
    )
    return data

# Function to generate box and violin plots
```

```python
def generate_plots(data, value_column, results, output_dir, normalized=False):
    plot_suffix = "normalized" if normalized else "raw"

    # Box plot
    plt.figure(figsize=(12, 8))
    sns.boxplot(data=data, x='Allele', y=value_column, hue='Source', palette='Set2')

    # Overlay mean points with distinct colors for each group (H and S)
    means = data.groupby(['Allele', 'Source'])[value_column].mean().reset_index()
    colors = {'H': 'blue', 'S': 'green'}  # Distinct colors for H and S (changed orange to green for
better contrast)
    allele_order = {allele: i for i, allele in enumerate(data['Allele'].unique())}  # Get numeric
positions of alleles

    for _, row in means.iterrows():
        x_position = allele_order[row['Allele']] + (-0.2 if row['Source'] == 'H' else 0.2)
        plt.scatter(
            x=x_position,  # Use numeric positions for plotting
            y=row[value_column],
            color=colors[row['Source']], zorder=5
        )
        plt.text(
            x=x_position,
            y=row[value_column] + data[value_column].max() * 0.02,  # Slightly above the mean
            s=f"{row[value_column]:.2f}",
            ha='center', va='bottom', color=colors[row['Source']], fontsize=9
        )

    title = f"Box Plot of {'Normalized ' if normalized else ''}Sequence Count (%) by Allele and
Source"
    plt.title(title)
    plt.xlabel('Allele')
    plt.ylabel(value_column)
    plt.grid(axis='y', linestyle='--', alpha=0.7)

    # Annotate U and p-values
    for i, allele in enumerate(data['Allele'].unique()):
        u_stat = results[allele]['U Statistic']
        p_value = results[allele]['P-value']
        plt.text(x=i, y=data[value_column].max() * 1.02,  # Lower the position of the annotation
                 s=f"U={u_stat:.1f}, p={p_value:.3e}",
                 ha='center', fontsize=9, color='black')

    plt.legend(title='Source', loc='upper center', bbox_to_anchor=(0.5, -0.15), ncol=2)
    boxplot_path = os.path.join(output_dir, f'box_plot_{plot_suffix}.png')
    plt.tight_layout()
    plt.savefig(boxplot_path)
```

```
    plt.close()

    # Violin plot
    plt.figure(figsize=(12, 8))
    sns.violinplot(data=data, x='Allele', y=value_column, hue='Source', palette='Set2', split=True,
scale='width')

    # Overlay mean points with distinct colors for each group (H and S)
    for _, row in means.iterrows():
        x_position = allele_order[row['Allele']] + (-0.2 if row['Source'] == 'H' else 0.2)
        plt.scatter(
            x=x_position,  # Use numeric positions for plotting
            y=row[value_column],
            color=colors[row['Source']], zorder=5
        )
        plt.text(
            x=x_position,
            y=row[value_column] + data[value_column].max() * 0.02,  # Slightly above the mean
            s=f"{row[value_column]:.2f}",
            ha='center', va='bottom', color=colors[row['Source']], fontsize=9
        )

    title = f"Violin Plot of {'Normalized ' if normalized else ''}Sequence Count (%) by Allele and
Source"
    plt.title(title)
    plt.xlabel('Allele')
    plt.ylabel(value_column)
    plt.grid(axis='y', linestyle='--', alpha=0.7)

    # Annotate U and p-values
    for i, allele in enumerate(data['Allele'].unique()):
        u_stat = results[allele]['U Statistic']
        p_value = results[allele]['P-value']
        plt.text(x=i, y=data[value_column].max() * 1.02,  # Lower the position of the annotation
                 s=f"U={u_stat:.1f}, p={p_value:.3e}",
                 ha='center', fontsize=9, color='black')

    plt.legend(title='Source', loc='upper center', bbox_to_anchor=(0.5, -0.15), ncol=2)
    violinplot_path = os.path.join(output_dir, f'violin_plot_{plot_suffix}.png')
    plt.tight_layout()
    plt.savefig(violinplot_path)
    plt.close()

# Main function
def main(input_file, value_column, output_dir):
    # Load the dataset
    data = pd.read_csv(input_file)
```

```
    # Create output directory if it doesn't exist
    os.makedirs(output_dir, exist_ok=True)

    # Raw data analysis
    results_raw = mann_whitney_test(data, value_column, output_dir)
    generate_plots(data, value_column, results_raw, output_dir, normalized=False)

    # Normalize the data
    data = normalize_sequence_counts(data)

    # Normalized data analysis
    normalized_column = 'Normalized Sequence Count (%)'
    results_normalized = mann_whitney_test(data, normalized_column, output_dir)
    generate_plots(data, normalized_column, results_normalized, output_dir, normalized=True)

    print("Analysis complete. Results and plots (both raw and normalized) have been saved.")

if __name__ == "__main__":
    input_file =
'/local/muham020/suismetsys/trial_pipeline/result_8_statistical_analysis/8.1_hsdS_allele_distributio
n_table/result_8.1_table_combined.csv'
    value_column = 'Sequence Count'          # Column to analyze
    output_dir =
'/local/muham020/suismetsys/trial_pipeline/result_8_statistical_analysis/8.4_mann_whitney_u_test'

    main(input_file, value_column, output_dir)
```

Script S17 Python script to perform Kruskal Wallis test to identify the significance of phase variant distribution across healthy and sick groups and across primer sets

```
import pandas as pd
from scipy.stats import kruskal
import os

# File path
input_file =
"/local/muham020/suismetsys/trial_pipeline/result_8_statistical_analysis/8.1_hsdS_allele_distributio
n_table/result_8.1_table_combined.csv"
output_dir =
"/local/muham020/suismetsys/trial_pipeline/result_8_statistical_analysis/8.5_kruskal_wallis_across_p
rimers"
output_file = os.path.join(output_dir, "kruskal_wallis_results.csv")
report_file = os.path.join(output_dir, "kruskal_wallis_report.txt")

# Ensure output directory exists
os.makedirs(output_dir, exist_ok=True)

# Load the data
data = pd.read_csv(input_file)

# Normalize sequence counts as percentages within each group
data['Normalized Sequence Count (%)'] = (
    data.groupby('Source')['Sequence Count']
    .transform(lambda x: (x / x.sum()) * 100)
)

# Filter necessary columns
results = []
report_lines = []

# Perform Kruskal-Wallis test for each allele within group H and group S
for allele in data['Allele'].unique():
    allele_data = data[data['Allele'] == allele]
    for group in ['H', 'S']:
        group_data = allele_data[allele_data['Source'] == group]
        primer_sets = group_data['Primer Set'].unique()
        if len(primer_sets) > 1:  # Ensure there are multiple groups to compare
            grouped_data = [group_data[group_data['Primer Set'] == primer]['Sequence Count'].values
for primer in primer_sets]
            # Check if all numbers are identical across all groups
            if all(len(set(data)) == 1 for data in grouped_data):
                report_lines.append(f"Allele: {allele} (Group: {group})\n")
                report_lines.append("  All numbers are identical across groups. Test not
performed.\n\n")
                continue
            stat, p_value = kruskal(*grouped_data)
```

```python
            significant = "Yes" if p_value < 0.05 else "No"
            results.append({
                'Allele': allele,
                'Group': group,
                'Comparison': 'Across Primers',
                'Kruskal-Wallis Statistic': stat,
                'p-value': p_value,
                'Significant': significant
            })
            # Add to report
            report_lines.append(f"Allele: {allele} (Group: {group})\n")
            report_lines.append(f"  Comparison: Across Primers\n")
            report_lines.append(f"  Kruskal-Wallis Statistic (H): {stat:.4f}\n")
            report_lines.append(f"  p-value: {p_value:.4f}\n")
            report_lines.append(f"  Significant (p < 0.05): {significant}\n")
            report_lines.append("\n")

# Perform Kruskal-Wallis test across groups (H vs. S) for each allele
for allele in data['Allele'].unique():
    allele_data = data[data['Allele'] == allele]
    h_values = allele_data[allele_data['Source'] == 'H']['Normalized Sequence Count (%)']
    s_values = allele_data[allele_data['Source'] == 'S']['Normalized Sequence Count (%)']

    if len(h_values) > 0 and len(s_values) > 0:  # Ensure both groups have data
        stat, p_value = kruskal(h_values, s_values)
        significant = "Yes" if p_value < 0.05 else "No"
        results.append({
            'Allele': allele,
            'Group': 'H vs S',
            'Comparison': 'Across Groups',
            'Kruskal-Wallis Statistic': stat,
            'p-value': p_value,
            'Significant': significant
        })
        report_lines.append(f"Allele: {allele} (H vs S)\n")
        report_lines.append(f"  Comparison: Across Groups\n")
        report_lines.append(f"  Kruskal-Wallis Statistic (H): {stat:.4f}\n")
        report_lines.append(f"  p-value: {p_value:.4f}\n")
        report_lines.append(f"  Significant (p < 0.05): {significant}\n")
        report_lines.append("\n")

# Save the results to a CSV file
results_df = pd.DataFrame(results)
results_df.to_csv(output_file, index=False)

# Save the comprehensive report to a text file
with open(report_file, 'w') as f:
    f.writelines(report_lines)
```

```
print(f"Kruskal-Wallis test results saved to {output_file}")
print(f"Comprehensive report saved to {report_file}")
```