QAA_report

2025-09-07

Part 1 - Read quality score distributions

The following plots consist of output from a FASTQC run on 2 RNA-seq files from the electric fish studies (PRJNA1005245 and PRJNA1005244). The two libraries I am processing throughout this report include SRR25630301 (Crh_rhy52_EO_6cm_1) and SRR25630377 (CcoxCrh_comrhy114_EO_adult_2). Also included in this part of the report are per base sequence quality plots created by python script rather than FastQC report for comparison.

SSRR25630301:

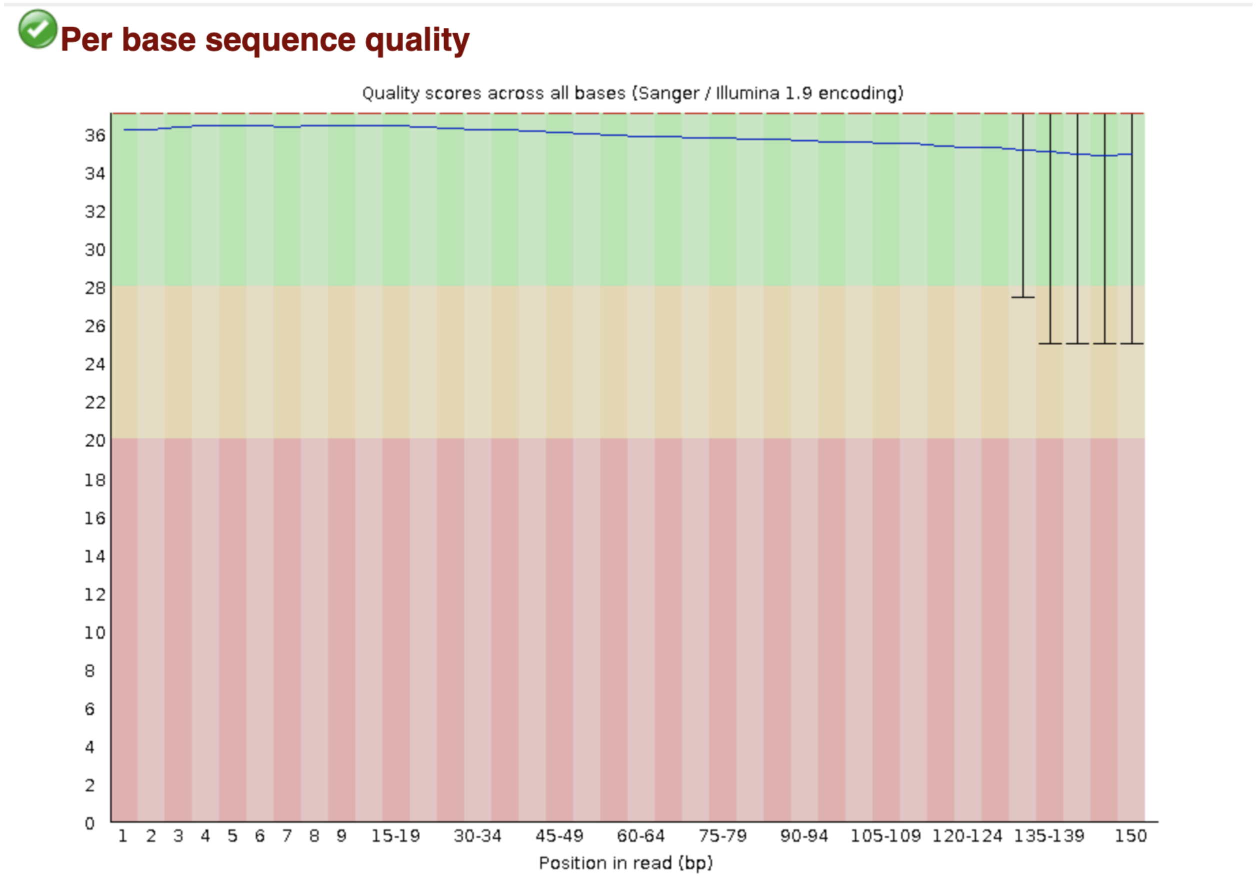
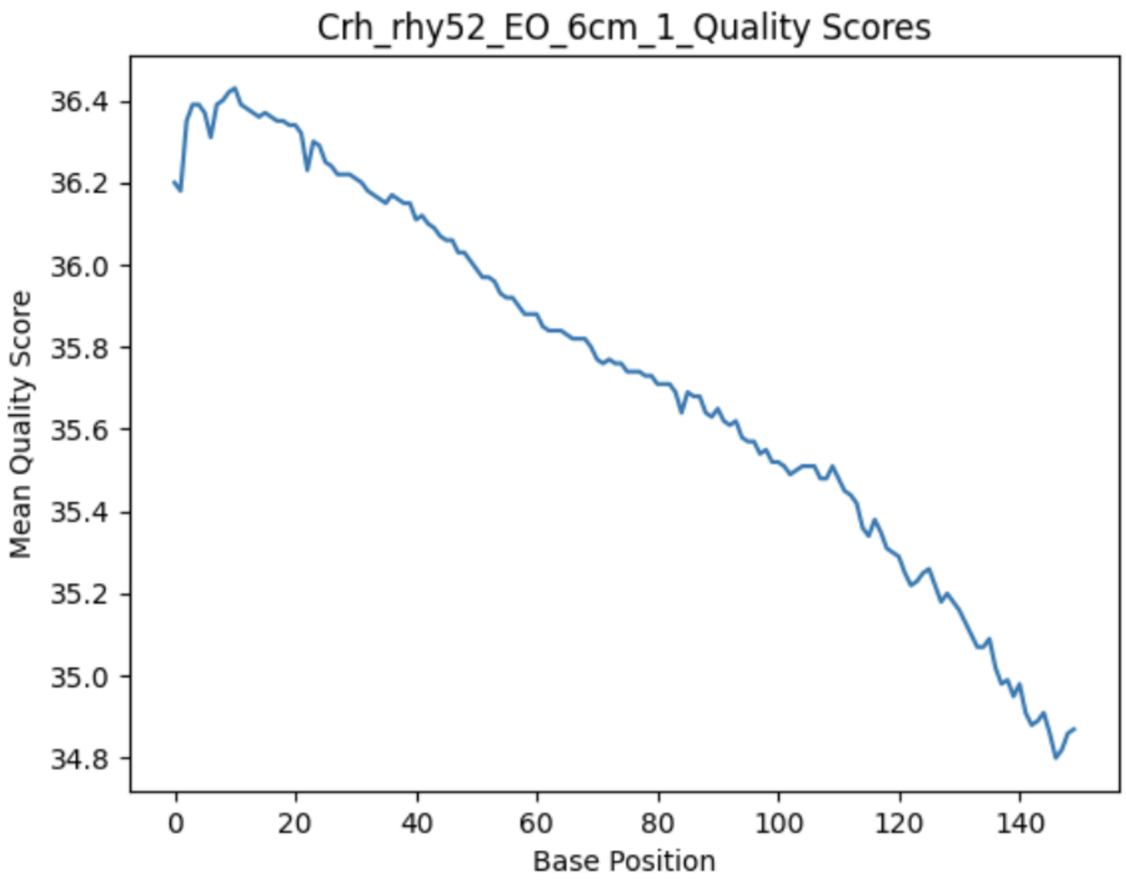
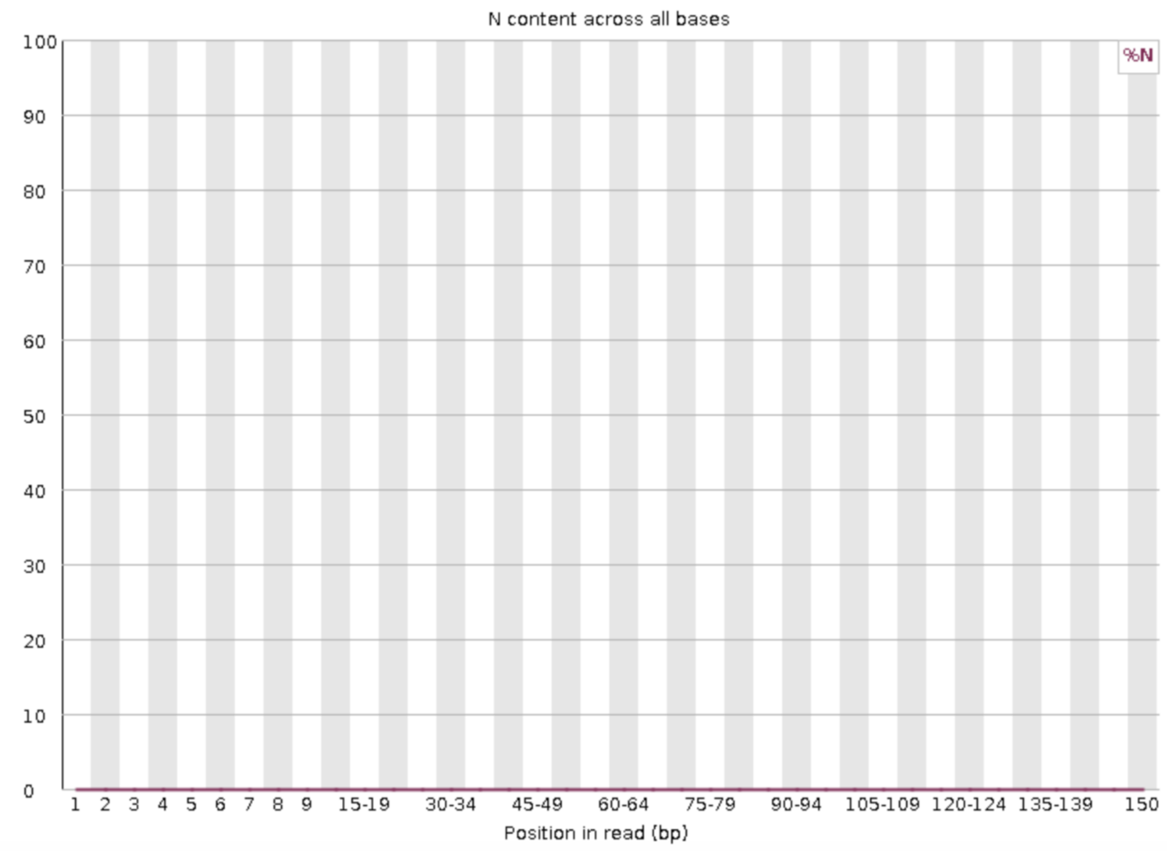


Figure 1:FASTQC per-base quality score distribution for Crh_rhy52_EO_6cm R1 reads. The majority of bases have high quality (Phred >30), with slight decline near the read ends.

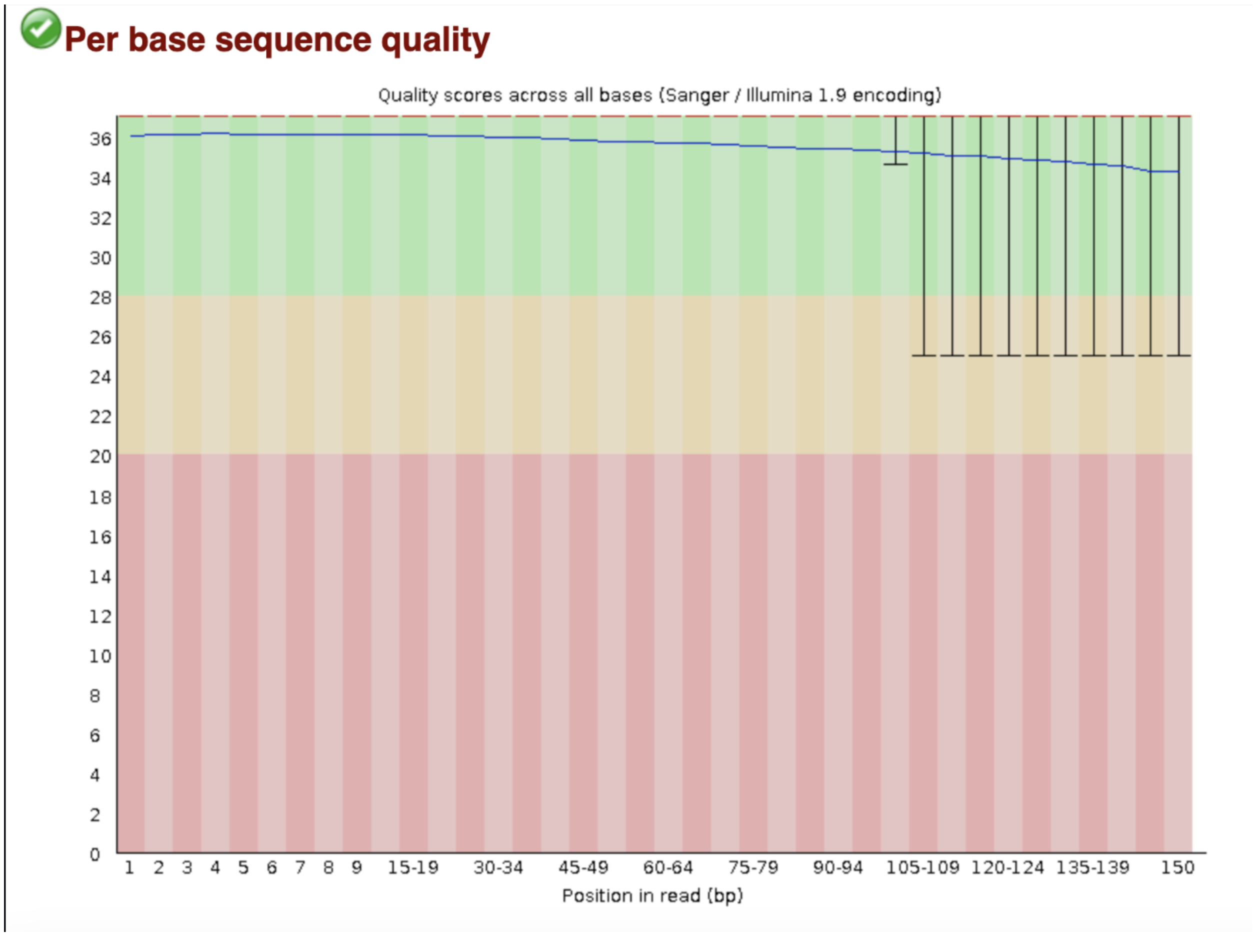


Python Script: Per Base Quality Score Distribution for Crh_rhy52_EO_6cm R1 Reads

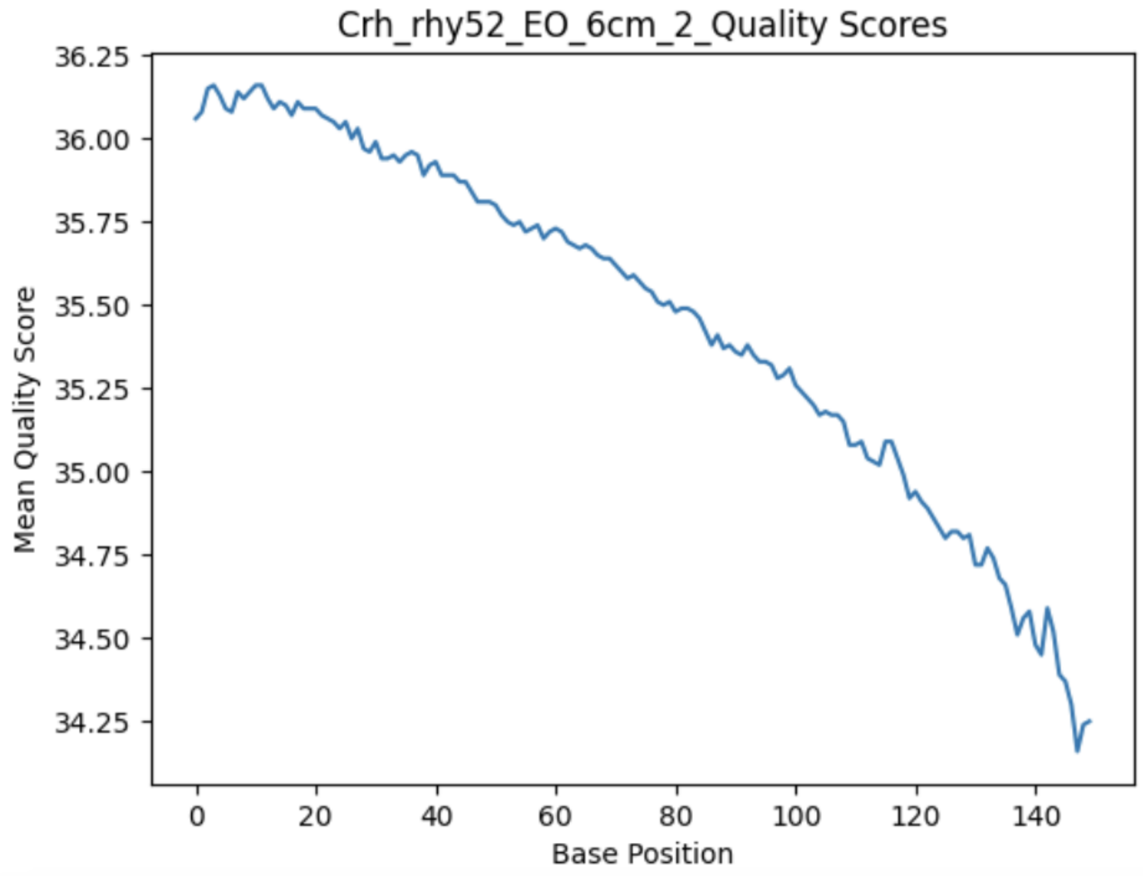
Per base N content



FASTQC: Per Base N Content for Crh_rhy52_EO_6cm R1 Reads

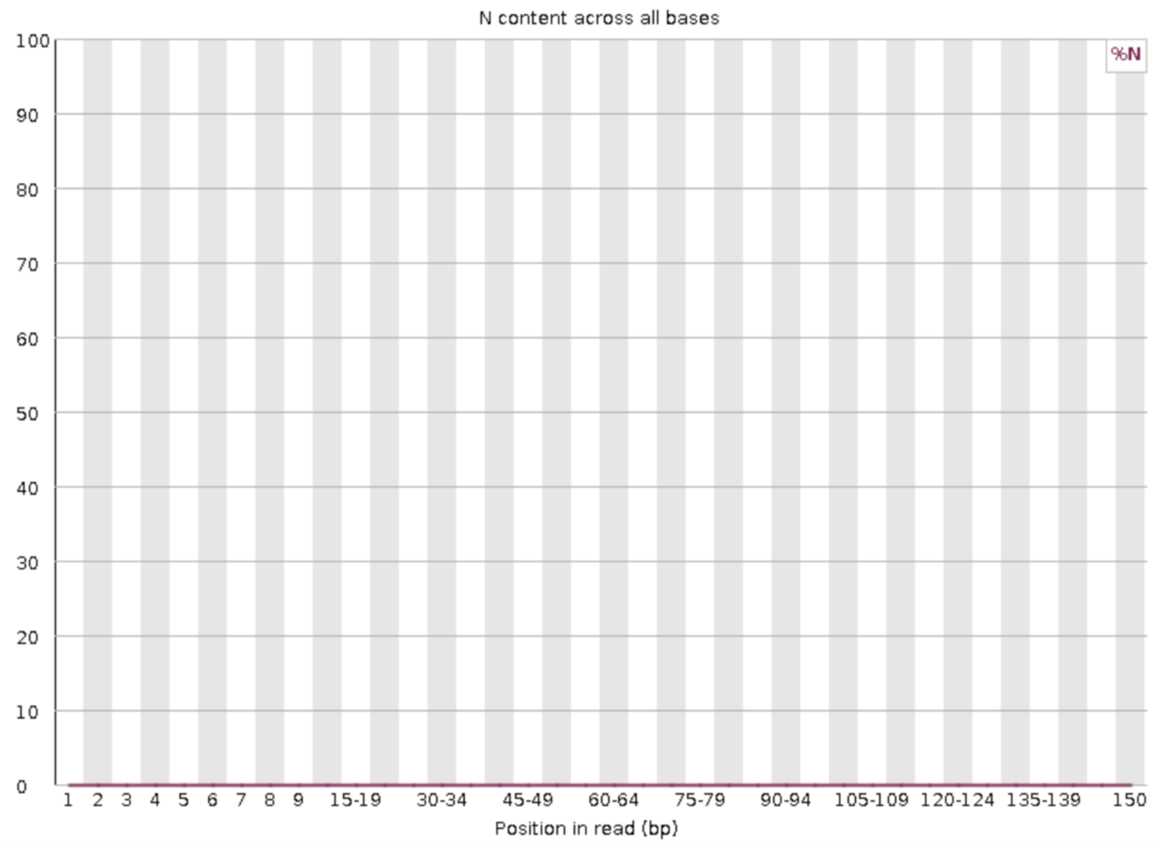


FASTQC: Per Base Quality Score Distribution for Crh_rhy52_EO_6cm R2 Reads



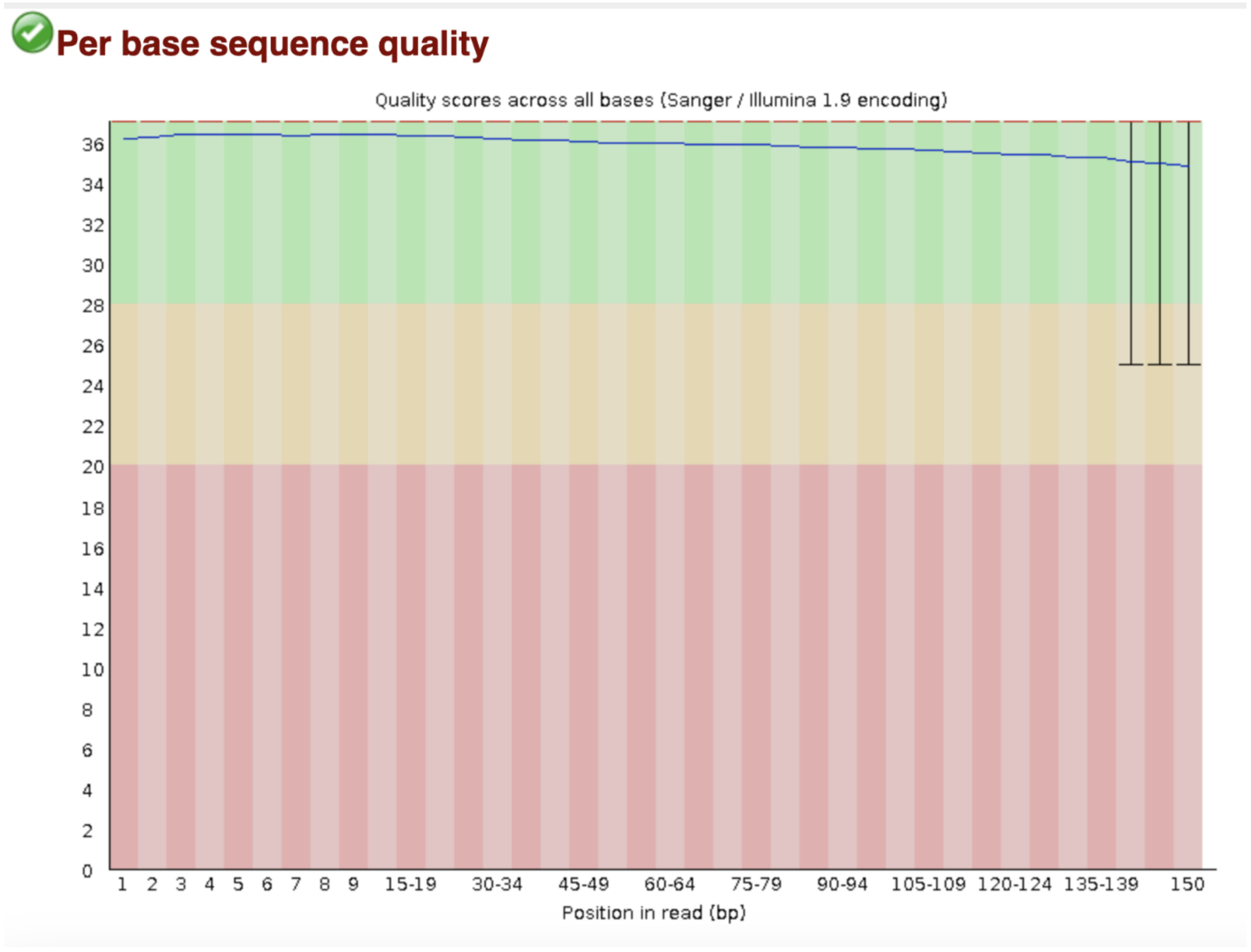
Python Script: Per Base Quality Score Distribution for Crh_rhy52_EO_6cm R2 Reads

✓ Per base N content

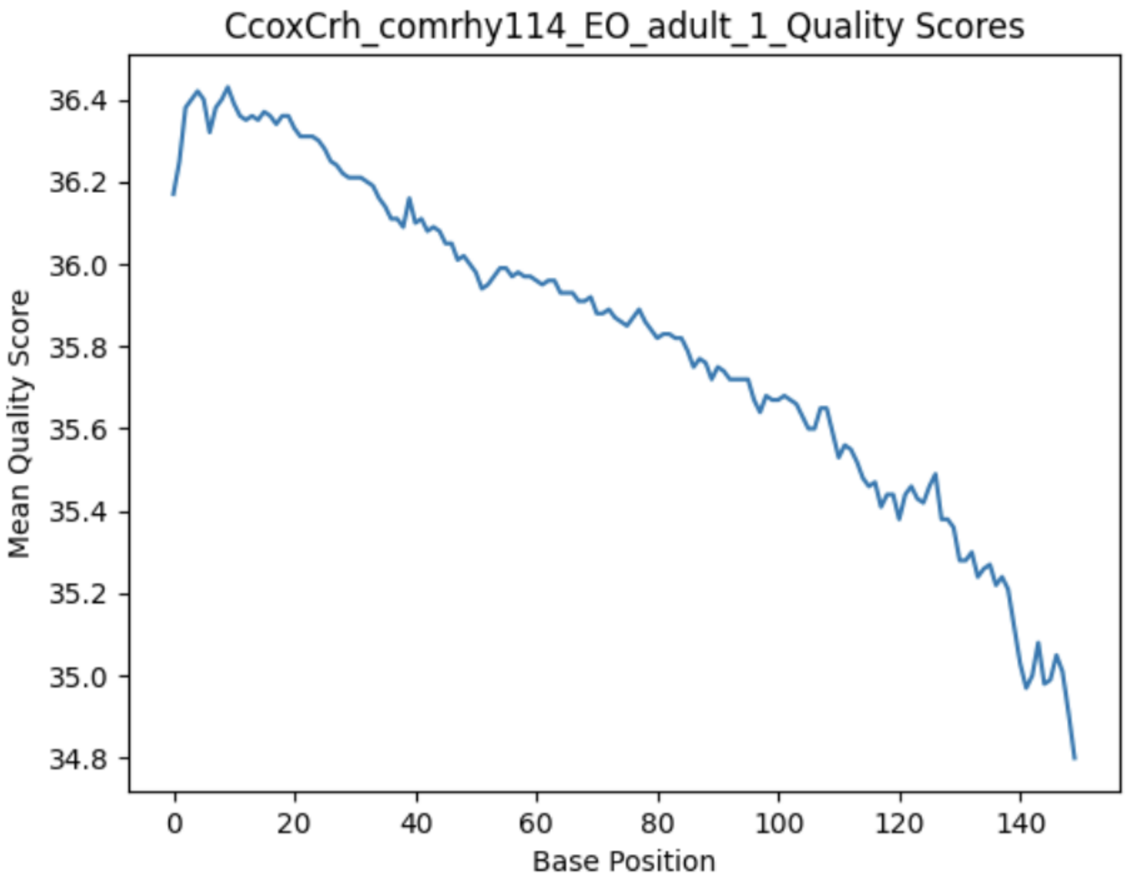


FASTQC: Per Base N Content for Crh_rhy52_EO_6cm R2 Reads

SSRR25630377:

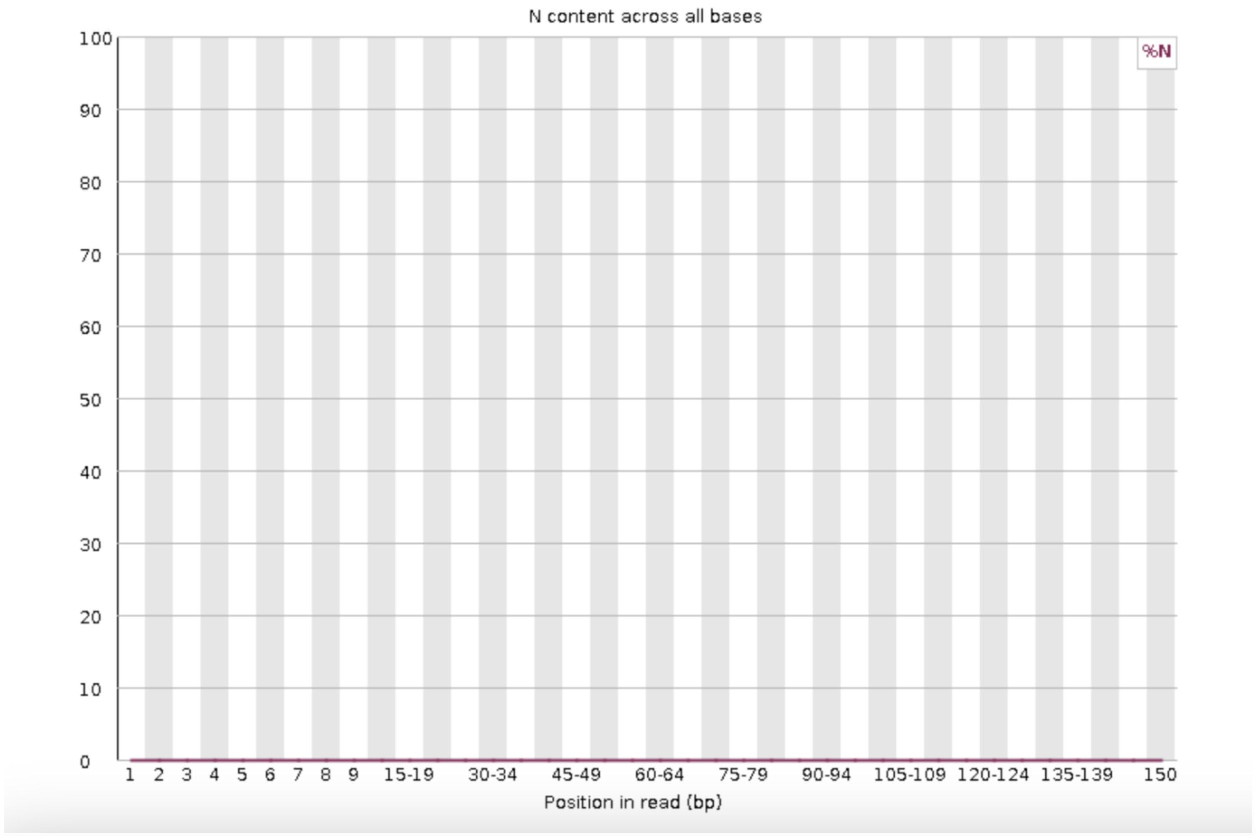


FASTQC: Per Base Quality Score Distribution for CcoxCrh_comrhy114_EO_adult R1 Reads



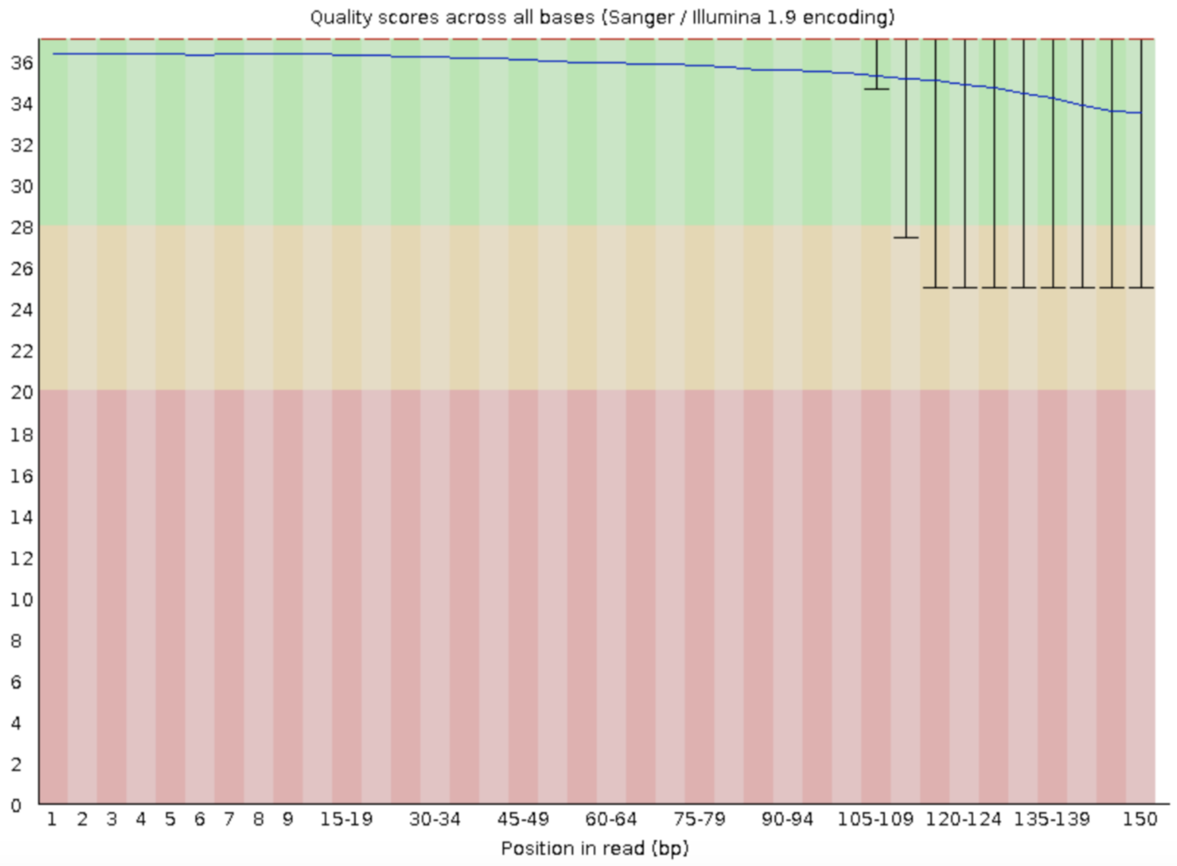
Python Script: Per Base Quality Score Distribution for CcoxCrh_comrhy114_EO_adult R1 Reads

✓ Per base N content

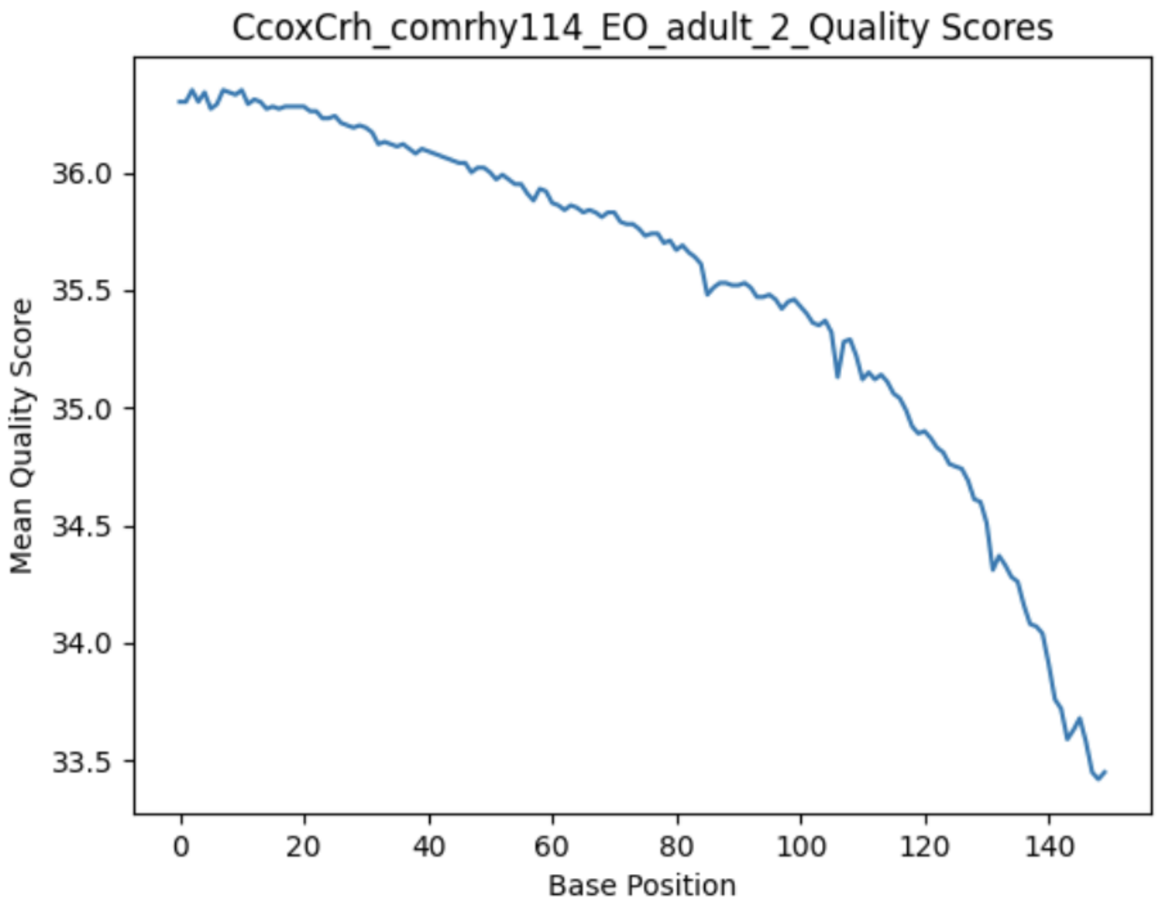


FASTQC: Per Base N Content for CcoxCrh_comrhy114_EO_adult R1 Reads

Per base sequence quality

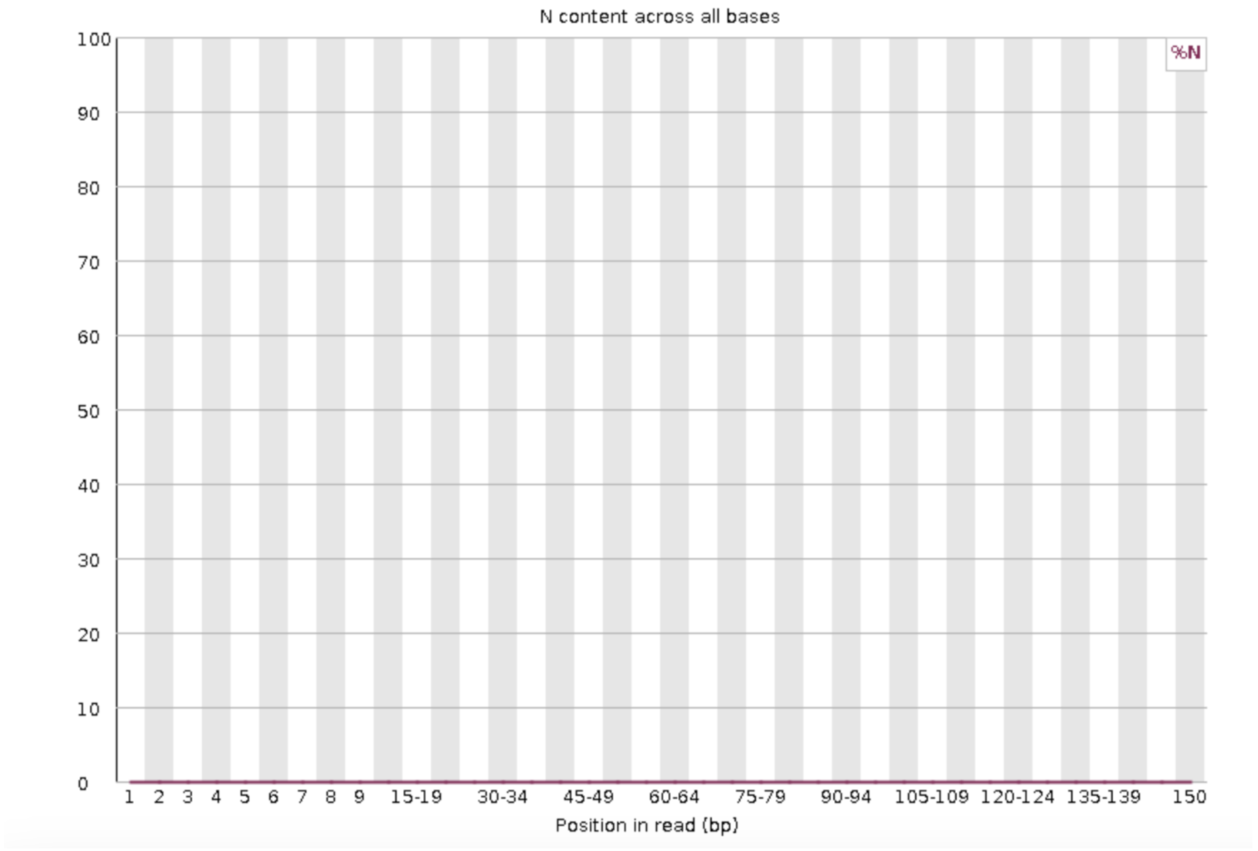


Python Script: Per Base Quality Score Distribution for CcoxCrh_comrhy114_EO_adult R2 Reads



Python Script: Per Base Quality Score Distribution for CcoxCrh_comrhy114_EO_adult R2 Reads

✓ Per base N content



FASTQC: Per Base N Content for CcoxCrh_comrhy114_EO_adult R2 Reads

FastQC vs. Python script plots Both approaches show consistently high per-base Phred quality (>30) across most positions. FastQC shows slightly larger error bars toward read ends, while my Python script reports a smoother decline. The difference is due to how each tool calculates and visualizes base quality variance: FastQC bins quality values and adds error bars, while my script averages directly.

Runtime & resources: FastQC is optimized C++/Java and runs quickly with low memory, whereas my Python script was much slower and used more memory.

Overall quality: The two RNA-seq libraries analyzed in this study showed clear differences in overall quality.

For SRR25630301 (Crh_rhy52_EO_6cm), FastQC revealed consistently high per-base quality scores across both R1 and R2, with only a minor decline at the 3' ends. Per-base N content remained close to zero, and adapter contamination was low, with only ~5–6% of reads trimmed by Cutadapt. Other FastQC modules (e.g., duplication levels, overrepresented sequences) passed without major concerns. Together, these results indicate that this library is of high quality and well-suited for downstream alignment and differential expression analysis.

In contrast, SRR25630377 (CcoxCrh_comrhy114_EO_adult) displayed lower quality metrics. Per-base quality scores declined more steeply toward the 3' ends, and adapter contamination was substantially higher, with ~20.6% of reads trimmed. N content was slightly elevated at later cycles, and FastQC flagged additional warnings, including sequence duplication and overrepresented sequences, which are expected for smaller or lower-complexity RNA-seq datasets. These quality issues were reflected later in downstream analyses: only ~27,000 reads aligned to the reference genome out of ~7 million processed, and fewer than 100 reads were assigned to annotated features.

Taken together, SRR25630301 provides a high-quality dataset for downstream analysis, while SRR25630377 is of much lower quality and should be interpreted with caution. Including both libraries remains useful for completeness, but conclusions should acknowledge the stark difference in sequencing depth, alignment rates, and overall reliability.

Part 2 – Adaptor trimming comparison

Trimmed reads with cutadapt and trimmomatic

Cutadapt

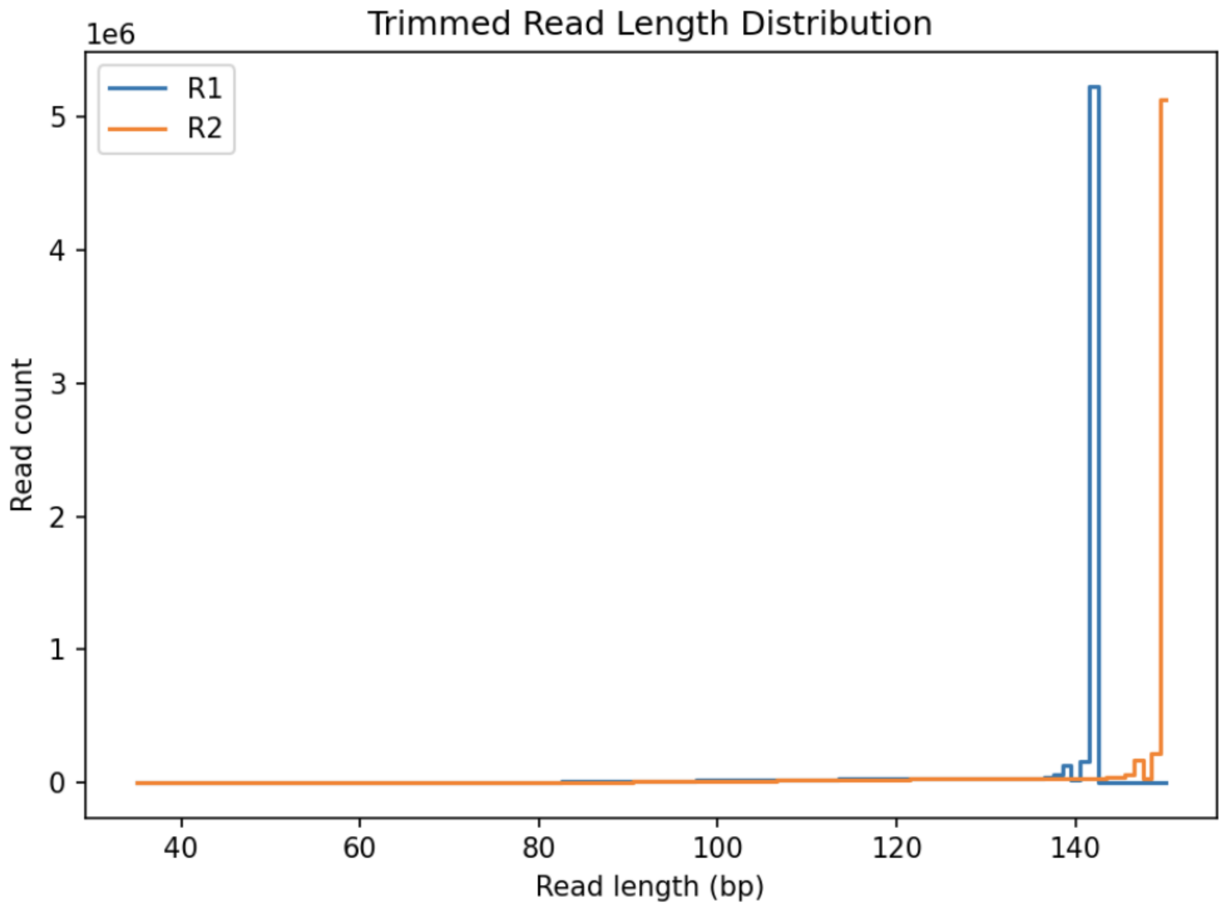
Proportion trimmed: Crh_rhy52_EO_6cm: 5.4% (R1) and 6.1% (R2)

CcoxCrh_comrhy114_EO_adult: 20.6% (both R1 and R2)

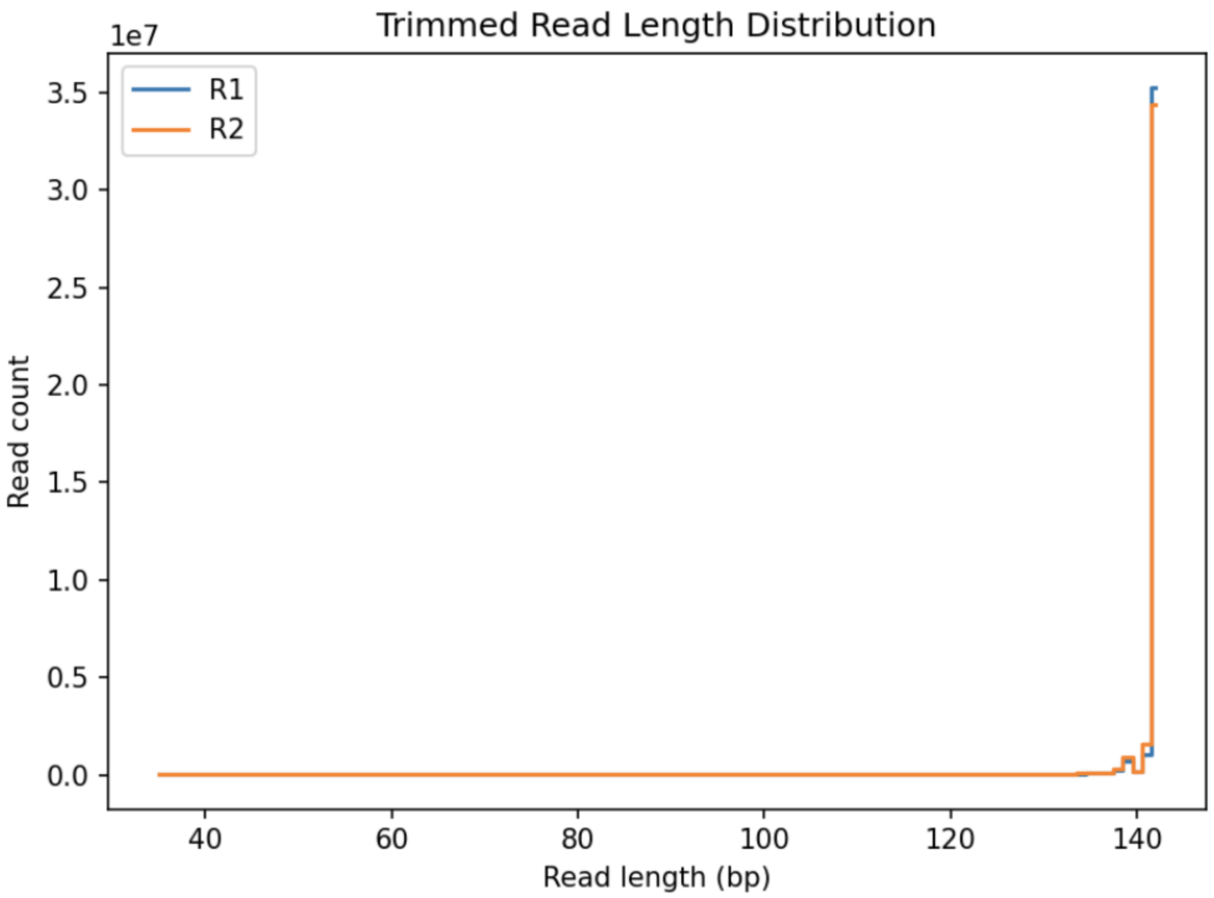
#R1 vs. R2 trimming rates:

It is expected that R2 reads are more frequently adapter-trimmed because Illumina insert sizes can be shorter than read length, causing R2 to run into adapter sequence sooner.

Trimmed Reads Plots



Trimmed read length distribution for CcoxCrh_comrhy114_EO_adult paired-end reads (R1 and R2). R1 and R2 exhibit more extensive trimming than Crh_rhy52, consistent with the higher proportion of adapters detected (~20.6% vs. ~5–6%).



Trimmed read length distribution for Crh_rhy52_EO_6cm paired-end reads (R1 and R2). The majority of reads retain near full length after trimming, with modest shortening at the 3' ends.

Proportion trimmed:

Crh_rhy52_EO_6cm: 5.4% (R1) and 6.1% (R2)

CcoxCrh_comrhy114_EO_adult: 20.6% (both R1 and R2)

Sanity check (adapter confirmation): You could confirm with zcat sample.fastq.gz | grep -c AGATCGGAA-GAGC for raw detection of Illumina adapter motifs, or grep -B1 -A1 AGATCGGAAGAGC file.fastq.gz | head to visualize flanking bases. Matches confirm the correct adapter orientation.

R1 vs. R2 trimming rates: It is expected that R2 reads are more frequently adapter-trimmed because Illumina insert sizes can be shorter than read length, causing R2 to run into adapter sequence sooner.

Part 3 – Alignment and strand-specificity

Table 1. Mapped and unmapped reads after deduplication

Sample ID	Species	Mapped Reads	Unmapped Reads	Total Reads
SRR25630301	*Campylomormyrus rhynchophorus*	16,705,591	1,813	16,707,404
SRR25630377	*C. compressirostris* × *C. rhynchophorus*	27,726	6,991,629	7,019,355

Strandedness (htseq-count): SRR25630301: stranded=yes → 579,836 counts; stranded=reverse → 10,681,434 counts.

SRR25630377: stranded=yes → 77 counts; stranded=reverse → 66 counts.

Counts were found with: grep -v '^' **counts_stranded_yes.txt | awk '{sum += $2} END {print sum}' grep -v '^'** counts_stranded_reverse.txt | awk '{sum += $2} END {print sum}'

For SRR25630301, the reverse-stranded run has ~18× more assigned reads, strongly indicating the library is reverse-stranded. For SRR25630377, counts are too low to be meaningful, but following the kit protocol (NEXTflex Rapid Directional RNA-seq, which produces reverse-stranded libraries), you should use –stranded=reverse for downstream analyses.