Compute the ONT mean-Read Quality score from ONT FastQ data

Stéphane Plaisance [VIB - Nucleomics Core, nucleomics@vib.be]

*TueJan*09, 2024 - version 1.0

Contents

Version 1.0 (initial version)

Aim

Compute the read mean quality score directly from ther basecalled fastQ data (only barcode 1 data used here) and compare with the value stored in the run summary file.

This work was done using a Rapid-kit Lambda library loaded on a Flongle (ARX535) with the ONT lambda material and sequenced using Minknow (23.07.12) on our GridION device (Guppy v7.1.4, SUP mode).

NOTE: For more information about this topic, please read the following **ONT Community forum post** [1]

[1] https://community.nanoporetech.com/posts/what-is-the-base-value-for

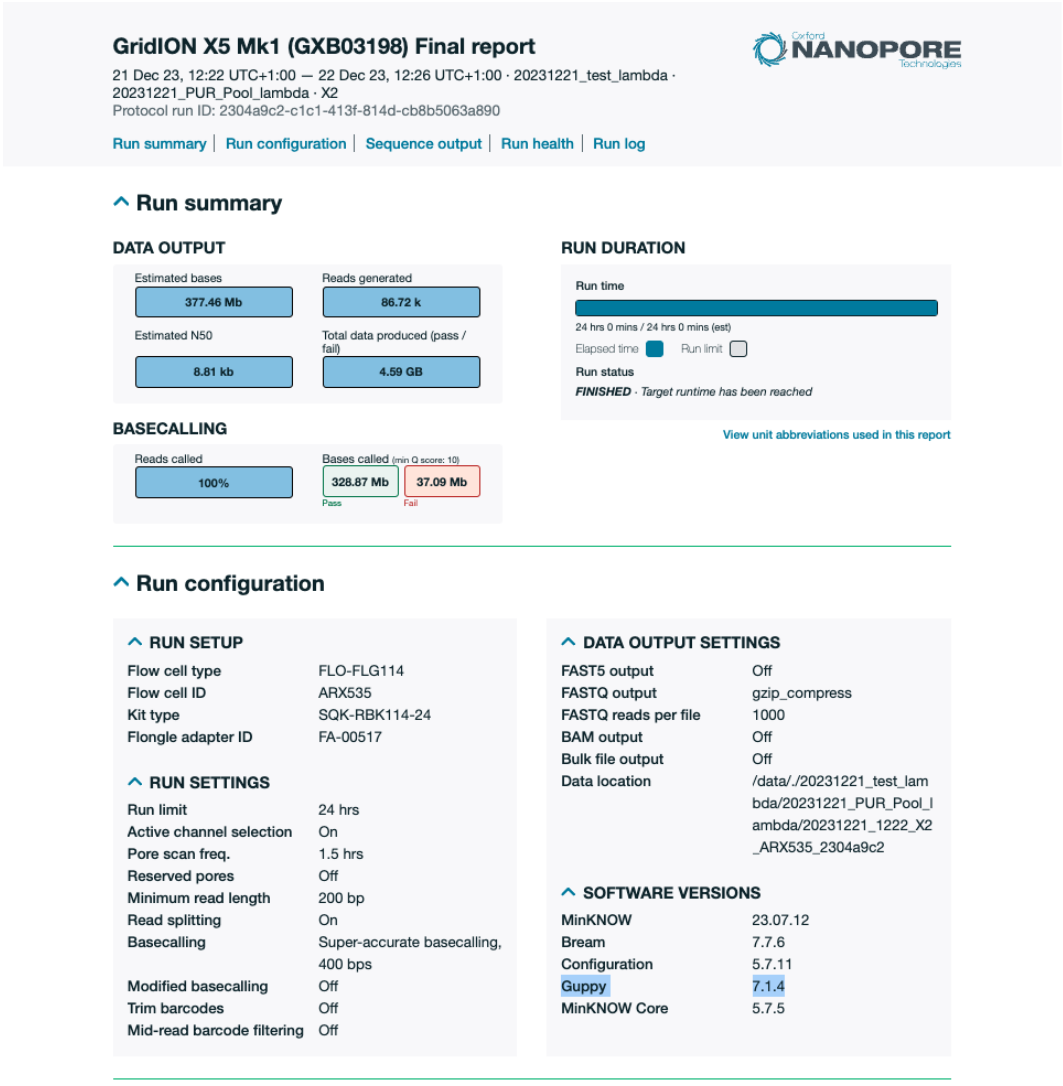
Results

Run results

The run delivered reasonable amounts of data after 24h and we extracted here only the reads for the barcode01 sample to reduce the size of data to visualize. Other barcodes were also prepared with lambda and gave similar results (see table below).

Demultiplexing results in the Run report were:

Barcode	Total bases (Mb)	Passed bases (%)	Total reads (k)	Passed reads (%)
barcode01	54	95,7	11	96,8
barcode02	59	96,2	13	96,6
barcode03	36	95,8	8	96,2
barcode04	53	96,1	11,1	96,7
barcode05	53	94,7	10,64	96,2
barcode06	52	94,3	11,46	95,4



The run final summary was as follows:

The summary file was saved under: final_summary_ARX535_2304a9c2_e3334dcc.txt and contains the following information.

```
instrument=GXB03198
position=X2
```

```
flow_cell_id=ARX535
sample_id=20231221_PUR_Pool_lambda
protocol_group_id=20231221_test_lambda
protocol=sequencing/sequencing_MIN114_DNA_e8_2_400K:FLO-FLG114:SQK-RBK114-24:400
protocol_run_id=2304a9c2-c1c1-413f-814d-cb8b5063a890
acquisition_run_id=e3334dcc5003b3df7b851bdc4d74156348034e1b
started=2023-12-21T12:26:20.482579+01:00
acquisition_stopped=2023-12-22T12:26:23.533448+01:00
processing_stopped=2023-12-22T12:26:32.406056+01:00
basecalling_enabled=1
sequencing_summary_file=sequencing_summary_ARX535_2304a9c2_e3334dcc.txt
fast5_files_in_final_dest=0
fast5_files_in_fallback=0
pod5_files_in_final_dest=105
pod5_files_in_fallback=0
fastq_files_in_final_dest=103
fastq_files_in_fallback=0
```

By contrast, our Minknow software was reported as follows in the corresponding gridION log (/var/log/minknow/X2/analyser_lo 0.txt)

```
2023-11-24 13:05:23.086944    INFO: starting_up (control)
    hostname: GXB03198
    system: GridION X5 Mk1 (GRD-X5B003) GXB03198 running on ubuntu 20.04
            Distribution:           23.07.12 (STABLE)
            MinKNOW Core:           5.7.5
            Bream:                  7.7.6
            Protocol configuration: 5.7.11
            Dorado (build):          7.0.5+b44bfcb66
            Dorado (connected):      7.1.4+d7df870c0
```

Note that while **Guppy** was cited in the run html report as **v7.1.4**, we read here Dorado instead with the two different version numbers for build and connected (same version as Guppy).

A few questions come to mind:

- which basecaller was actually used ?
- is Dorado a version of Guppy or Guppy using Dorado under the hood on the gridION?
- how can we obtain the exact version of the software used in each experiment? should we just believe what is written at the top of the HTML report or are the */var/log/minknow* files telling the truth?

Although these details may seem insignificant, they illustrate the constant changes in ONT tools, making it very difficult to keep track of what was done in the primary data generation and document this properly.

Custom code to compute the read mean quality

A custom script was written to parse fastq data and extract several read metrics. The underlying logic for the computation of the read mean quality score is similar to code found in different web pages, and in particular in the **Dorado** source [2] as kindly communicated by **Chris Wright** in the community post.

```
132 float mean_qscore_from_qstring(std::string_view qstring) {
133     if (qstring.empty()) {
134         return 0.0f;
135     }
136
137     // Lookup table avoids repeated invocation of std::pow, which
138     // otherwise dominates run time of this function.
139     // Unfortunately std::pow is not constexpr, so this can't be.
140     static const auto kCharToScoreTable = [] {
141         std::array<float, 256> a{};
142         for (int q = 33; q <= 127; ++q) {
143             auto shifted = static_cast<float>(q - 33);
144             a[q] = std::pow(10.0f, -shifted / 10.0f);
145         }
146         return a;
147     }();
148     float total_error =
149             std::accumulate(qstring.cbegin(), qstring.cend(), 0.0f,
150                             [](float sum, char qchar) { return sum + kCharToScoreTable[qchar]; });
151     float mean_error = total_error / static_cast<float>(qstring.size());
152     float mean_qscore = -10.0f * std::log10(mean_error);
153     return std::clamp(mean_qscore, 1.0f, 50.0f);
154 }
```

The following code was compiled and used with the Lambda barcode01 fastq read files to produce a metrics.txt output file used for plotting in the next part.

```cpp
#include <iostream>
#include <sstream>
#include <cmath>

// produce metrics for R-plotting
// usage: (zcat barcode01/* | ./fastq2metrics > fastq2metrics.txt
// SP@NC; 2024_01_05 (+GPT)
//
// script: fastq2metrics.cpp
// compile with: g++ -o fastq2metrics fastq2metrics.cpp

// Function to directly subtract 33 from ASCII value
int ord(char c) {
    return static_cast<int>(c) - 33;
}

// Function to calculate average quality
double aveQual(const std::string& quals) {
    double sum = 0;
    int len = quals.length();
    for (int i = 0; i < len; i++) {
        int q = ord(quals[i]);
        sum += pow(10, q / -10.0);
    }
    double avg_prob = sum / len;
    double avg_phred = -10 * log10(avg_prob);
    return avg_phred;
}
```

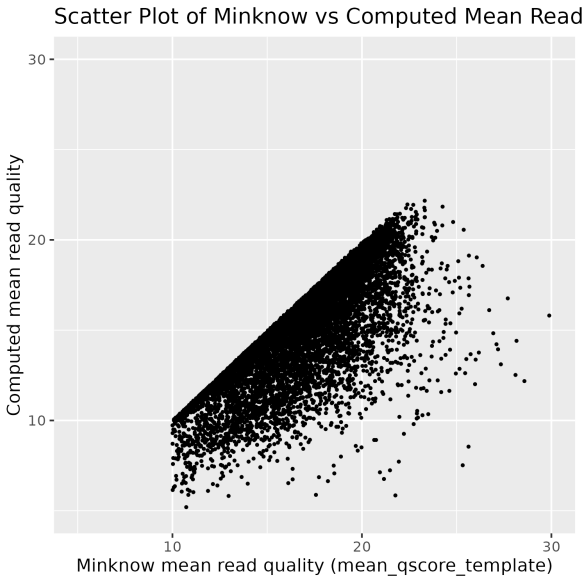
[2] https://github.com/nanoporetech/dorado/blob/a7fb3e3d4afa7a11cb52422e7eecb1a2cdb7860f/dorado/utils/sequence_utils.cpp#L132

```cpp
// Function to calculate GC content
double calculateGC(const std::string& sequence) {
    int gc_count = 0;
    for (char base : sequence) {
        if (base == 'G' || base == 'C') {
            gc_count++;
        }
    }
    return static_cast<double>(gc_count) / sequence.length();
}

// Function to round decimal numbers
std::string round_decimal(double number, int decimals) {
    std::ostringstream stream;
    stream.precision(decimals);
    stream << std::fixed << number;
    return stream.str();
}

int main() {
    std::cout << "readid\tmeanq\tlength\tgc" << std::endl;

    std::string line;
    while (std::getline(std::cin, line)) {
        if (line[0] == '@') {
            std::string header = line;
            std::string seq, sep, qual;
            std::getline(std::cin, seq);
            std::getline(std::cin, sep);
            std::getline(std::cin, qual);

            // Extract the first field from the space-separated header and remove leading '@'
            std::istringstream headerStream(header);
            std::string read_id;
            headerStream >> read_id;
            read_id.erase(0, 1);

            // Call the aveQual function and print the result
            double avg_phred = aveQual(qual);

            // Call the calculateGC function and print the result
            double gc_content = calculateGC(seq);

            // Output tab-separated values with rounded GC content and quality
            std::cout << read_id << "\t" << round_decimal(avg_phred, 2) << "\t"
                      << seq.length() << "\t" << round_decimal(gc_content, 2) << std::endl;
        }
    }

    return 0;
}
```

Import summary and compare to local read mean quality values

The mean read quality values contained in the Minknow summary report as **'mean_qscore_template'** were extracted from the summary file and compared by plotting to our locally computed values.

```
# import computed metrics (zcat barcode01/* | ./fastq2metrics > fastq2metrics.txt)
reads_metrics <- read.delim("fastq2metrics.txt")

# import ONT Minknow summary
summary <- "sequencing_summary_ARX535_2304a9c2_e3334dcc.txt"
sequencing_summary <- read.delim(summary)
# subset barcode1
sequencing_summary_bc01 <- subset(sequencing_summary, alias=="barcode01")
# keep only needed columns
summ <- sequencing_summary_bc01[,c('parent_read_id','sequence_length_template','mean_qscore_template')]

# merge both data.frames by readID
merged <- merge(reads_metrics,summ,by.x = 'readid',by.y='parent_read_id' )

# look for length conflicts between the two datasets
conflicts <- nrow(subset(merged, sequence_length_template != length))

# reports 0

# plot values against each other for each readID
scatter_plot <- ggplot(merged, aes(x = mean_qscore_template, y = meanq)) +
  geom_point(size=0.5) +
  labs(title = "Scatter Plot of Minknow vs Computed Mean Read Quality",
       x = "Minknow mean read quality (mean_qscore_template)",
       y = "Computed mean read quality") +
  xlim(5,30) +
  ylim(5,30)

ggsave("pictures/local_vs_minknow.png", scatter_plot)
```



After matching the values based on the read unique identifier, a large majority of the reads shows different quality scores between both sources while we would rather expect a perfect diagonal. By contrast, the length of the reads in both data-sets are identical for all reads, excluding a processing issue with our code.

We could not identify the code used in the Minknow process on our own device as we do not know where to look for this but the results shown below strongly suggest that the Dorado logic was not used for our GridION run.

Compare our local values to values obtained by a third-party tool

A perl script was kindly shared by **David Eccles** through the community post [3]

In order to validate our script, we applied this code to the same lambda data and plotted the results of David's perl script against our own C++ results.

```
# load fastx-length_results.txt
fastx_length_results <- read_table("fastx-length_results.txt", col_names = FALSE)
fastx_length_results$meanRQ <- as.numeric(gsub("q","",fastx_length_results$X2))
colnames(fastx_length_results) <- c('length','qstring','atstring','readID','meanRQ')
perlmetrics <- fastx_length_results[,c('readID','length', 'meanRQ')]

# merge with previous merge
merged2 <- merge(merged, perlmetrics, by.x='readid', by.y='readID')

# look for length conflicts between the two datasets
conflicts <- nrow(subset(merged2, length.x != length.y))

# reports 0

# plot values against each other for each readID
scatter_plot2 <- ggplot(merged2, aes(x = meanRQ, y = meanq)) +
  geom_point(size=0.5) +
  labs(title = "Scatter Plot of fastx_length.pl vs Computed Mean Read Quality",
       x = "fastx_length.pl mean read quality (after removing 'q')",
       y = "Computed mean read quality") +
  xlim(5,30) +
  ylim(5,30)

ggsave("pictures/local_vs_perl.png", scatter_plot2)
```

The plot shows a total identity between both sets (David's values are rounded to a single decimal but this does not change anything).

This confirms that the values extracted from the Minknow summary file differ from the consensus mean read quality scores obtained locally using two scripts and the logic present in the Github Dorado code.

[3] https://gitlab.com/gringer/bioinfscripts/-/blob/master/fastx-length.pl

Discussion

We show here that the **read mean quality** scores reported in the MinKnow summary file in the column **mean_qscore_template** do not match the values computed from the fastQ data itself for a majority of the reads and that the difference can be quite significant for a number of reads (cloud of points far away from the diagonal).

By contrast the column **sequence_length_template** reports perfectly matching length for the fastQ read, excluding that the suffix **template** used in the summary file could relate to something different than the final read itself.

The ONT representatives and documentation communicate constantly using the **Nanopore quality score** as reference to report progress in their platform and to compare to other platforms.

If our observation is correct, we consider that is is a major concern and reveals that ONT is using a undocumented methods to express key reference data which we need to blindly trust and cannot confirm locally with our own code.

The fact that Dorado uses the same logic to compute mean read quality scores as we do in our code is weird since our Minknow is expected to use Dorado for basecalling and therefore should have reported scores identical to ours.

A good clarification by the ONT developer team would be very welcome at this point to set things right.

A better transparency concerning the computation and meaning of the **mean read quality score** seems a reasonable request as this is very poorly documented now (and well hidden) as mentionned already two years ago on page 5 in a Plos One publication [4] (thanks to **Calleigh Herren** for sharing this reference).

Adding such information to a well exposed part of the online documentation as well as detailing the exact algorithm used to compute it (for each ONT piece of software that performs that operation) is kindly requested to the developer team as it will be useful not only to analysts but also to the general ONT user community.

As a general comment, clarification about key features, more stability of the pipelines, and a better centralized documentation would serve us a lot better than the now dispersed and sometimes unclear and overabundant information.

Note: the full html report, fastq data, as well as the final summary file are available upon request for validation.

[4] https://doi.org/10.1371/ journal.pone.0257521