BOOK CHAPTER

Transcript isoform-specific estimation of poly(A) tail length by Nanopore sequencing of native RNA

Adnan M. Niazi[1†], Maximilian Krause[1,2†] & Eivind Valen[1,2]

† Equally contributing authors

[1]Computational Biology Unit, University of Bergen, 5008 Bergen, Norway

[2]Sars International Centre for Marine Molecular Biology, University of Bergen, 5008 Bergen, Norway

Corresponding author
Eivind Valen
Email: eivind.valen@gmail.com
Phone: +47 55584074

Funding information
The project was supported by the Bergen Research Foundation (E.V.), the Sars International Centre for Marine Molecular Biology core funding (M.K), University of Bergen core funding (A.N.; K.L.) and the Norwegian Research Council (# 250049) (Y. T.-C.).

Keywords
Poly(A) tail, Nanopore sequencing, Native RNA, tailfindr, R, transcriptomics

Online Protocol
dx.doi.org/10.17504/protocols.io.9cjh2un

Code Repository
github.com/adnaniazi/tailfindr

The poly(A) tail is a homopolymeric stretch of adenosine at the 3'-end of mature RNA transcripts and its length plays an important role in nuclear export, stability, and translational regulation of mRNA. Existing techniques for genome-wide estimation of poly(A) tail length are based on short-read sequencing. These methods are limited because they sequence a synthetic DNA copy of mRNA instead of the native transcripts. Furthermore, they can identify only a short segment of the transcript proximal to the poly(A) tail which makes it difficult to assign the measured poly(A) tail length uniquely to a single transcript isoform. With the introduction of native RNA sequencing by Oxford Nanopore Technologies, it is now possible to sequence full-length native RNA. A single long read contains both the transcript and the associated poly(A) tail, thereby making transcriptome-wide isoform-specific poly(A) tail length assessment feasible. We developed *tailfindr* — an R-based package for estimating poly(A) tail length from Oxford Nanopore sequencing data. In this chapter, we describe in detail the pipeline for transcript isoform-specific poly(A) tail profiling based on native RNA Nanopore sequencing — from library preparation to downstream data analysis with *tailfindr*.

1 | INTRODUCTION

A poly(A) tail is formed by the non-templated addition of a stretch of adenosines to the 3'-end of messenger RNA (mRNA) during RNA processing in the nucleus [1]. It mediates the transfer of processed RNA from nucleus into the cytoplasm in eukaryotes [2]. Furthermore, it is known to stabilize or destabilize the mRNA depending on its length: relatively long poly(A) tails inhibit degradation of mRNA by 3'-exonucleases and 5'-cap hydrolysis, whereas short poly(A) tails mark the mRNA for degradation by the exosome [3]. Additionally, the length of the poly(A) tail can, under certain conditions, influence the translational efficiency of the mRNA [4–6]. Measuring isoform-specific poly(A) tail length over the whole transcriptome is therefore important in understanding its role in regulation of mRNA localization, mRNA half-life and translation regulation.

Existing methods for transcriptome-wide estimation of poly(A) tail length — which are primarily based on Illumina short-read sequencing technology [5, 7, 8] — have numerous limitations. First, RNA in its native form cannot be sequenced using Illumina sequencing: the RNA must first be reverse transcribed into cDNA, and subsequently amplified with PCR cycles to form clusters on the flow cell that are sequenced by synthesis. The conversion of RNA into cDNA results in loss of information; for example the occurrence of native RNA modifications might be interesting to study along with the poly(A) tail length. Second, the repeated PCR cycles may introduce artefacts in the homopolymer regions that may cause errors in poly(A) tail length estimation [9–11]. Third, these methods do not query the native poly(A) tail; instead, they estimate poly(A) tail length by inferring cDNA poly(A) or poly(T) segments using additional library preparation steps or custom-designed software for processing raw images of the sequencing clusters. This renders these methods not only time-consuming but also technically challenging. Lastly, as Illumina sequencing is a short-read sequencing technology, a sequenced read from these methods contains only a small segment reflecting parts of the transcript proximal to the poly(A) tail. With such partial transcript fragments, transcript isoform-specific poly(A) tail assignment is hard, and in many instances impossible. This is because a read may align equally well to two or more transcript isoforms, making it impossible to decipher as to which transcript the read — and its associated poly(A) tail measurement — belongs to (see **Figure 1**). Until recently it was therefore impossible to address whether different transcript isoforms have different poly(A) tail lengths.

With the advent of long read sequencing methods it recently became possible to sequence full length transcripts and their associated poly(A) tails [12–14]. In addition to offering long read sequencing only limited by the molecules integrity [15], Oxford Nanopore Technologies (ONT) novel sequencing approach also allows to sequence native RNA molecules without the conversion into cDNA [16]. This new technology has the potential to address isoform-specific poly(A) length measurements and RNA modification detection in a single assay [14, 17].

In this chapter, we will explain how ONT's sequencing approaches allow direct poly(A) measurement of native RNA (Section 2), describe the necessities for efficient Nanopore library preparation (Section 3), and how to process the data generated using *tailfindr* to perform transcriptome-wide isoform-specific poly(A) tail profiling (Section 4).

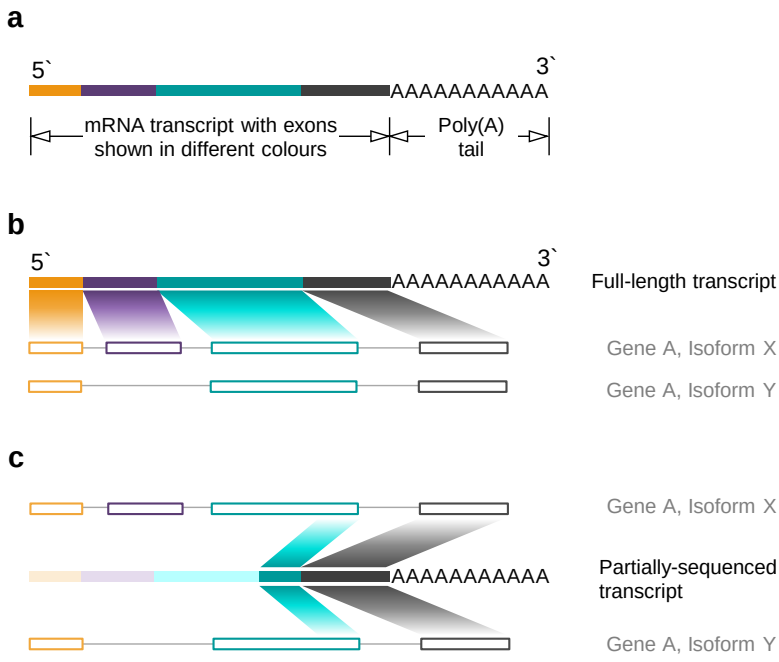

FIGURE 1 | (**a**) A poly(A)-tailed mRNA. (**b**) A full-length transcript uniquely and unambiguously maps to the isoform that it originated from. In the illustrated case, the read perfectly aligns to isoform X of gene A, and the measured poly(A) tail length can be uniquely attributed to isoform X. (**c**) A partially-sequenced transcript can map equally well to multiple transcript isoforms, making it impossible to decipher from which of the many possible isoforms the read originated from. In this case, the partially-sequenced transcript aligns equally well to both isoform X and isoform Y of gene A. Thus transcript-isoform specific poly(A) tail length assignment is not possible.

2 | NANOPORE SEQUENCING

In ONT sequencing approaches, a protein nanopore is suspended in a hydrophobic material (membrane) that separates two buffer-filled wells [18]. A cross-membrane voltage of -180mV is applied such that the *trans* side of the membrane is set at a positive potential compared to the *cis* side (see **Figure 2**). This causes a constant ionic current to flow through the pore. The molecule to be sequenced, which can be either DNA or RNA, is located on the *cis* side of the membrane. Under the influence of the applied voltage, the negatively-charged nucleotide strand threads through the pore. To ensure a homogeneous translocation rate (450bps for DNA and 70bps for RNA, [19] and to minimize the influence of secondary structure or DNA duplex binding energy, the DNA or RNA is fed into the pore by the ratcheting action of a motor protein. The nucleotides located in the constriction of the pore — 5 to 6 nucleobases at any given time — modulate the current passing through the pore, thereby creating sequence-specific modulations in the current. This current is sampled at a rate of 3012 samples/second and saved as an array in a `.fast5` file by MinKNOW — the data acquisition and experiment management software provided by ONT. The resulting signal trace — the so-called squiggle — thus contains the information of the contiguous nucleotide strand and possible RNA modifications and should be stored as 'raw data files'. The raw data files are then used by a basecaller to predict the original sequence.

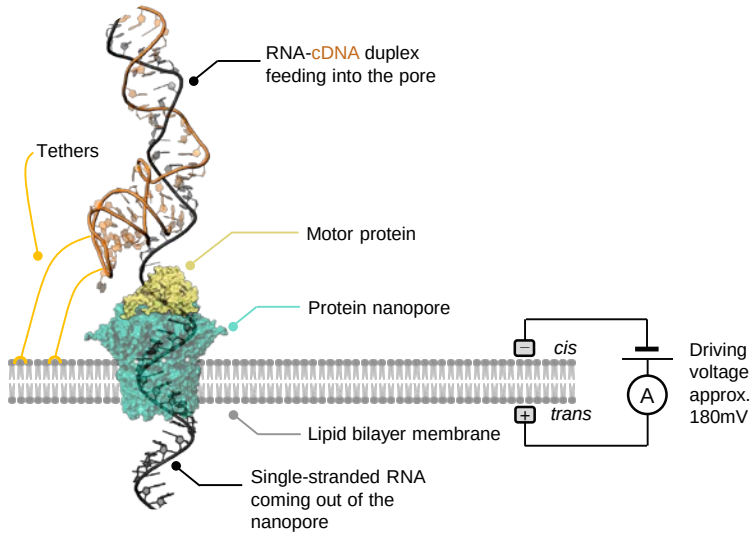

FIGURE 2 | RNA sequencing using ONT Direct-RNA sequencing. The RNA to be sequenced is first reverse-transcribed to make an RNA-cDNA duplex; this step removes RNA secondary structure that may otherwise cause pore blockage. The RNA-cDNA duplex, along with the ligated adaptor that contains the motor protein, is initially located on the *cis* side of the membrane. The tethers attached to the DNA adaptor have an affinity for the lipid membrane and help anchor the RNA-cDNA duplex to it. Under the influence of the applied voltage, the duplex shifts towards the pore, and eventually the RNA part of the duplex threads through the pore. The motor protein unwinds the RNA-cDNA duplex, and ratchets the RNA through the nanopore one base at a time. The fluctuations in the pore current as the RNA strand translocates through the pore are recorded.

In the special case of ONT native RNA sequencing, the motor protein is added at the 3'-end of the molecule by poly(A)-guided ligation (see **Figure 4**). Reverse transcription is optional, as the synthesized cDNA strand will not be sequenced at any time. Nevertheless it is recommended to perform reverse transcription, as the resulting RNA-cDNA heteroduplex is devoid of secondary structure that potentially interferes with pore translocation. Furthermore, the RNA-cDNA heteroduplex is more stable than single-stranded RNA towards degradation by RNases (see Note 1). The added motor protein threads the RNA through the pore from its 3'-end to the 5'-end. The resulting current signal thus contains in this order: signal for the adaptor sequence that initially carried the motor protein, the poly(A) tail and the full-length transcript.

Although in theory it should be possible to infer the length of the poly(A) tail from the basecalled sequence alone, in practice this is not the case. When the raw signal is basecalled, the number of adenosines (reflected as A in sequence) called by the current basecallers in the poly(A) tail region is far lower than the actual number of adenosines in the poly(A) tail of the original RNA sequence (see **Figure 3**). This is because the raw signal corresponding to a homopolymeric stretch of adenosine is a monotonous current devoid of any detectable transition from one adenosine to the next [20, 21]. The basecaller cannot decide where the signal of one adenosine ends and the next one starts; the entire poly(A) tail signal is therefore treated as a single adenosine base that got stalled in the nanopore for a long time. Thus, the poly(A) tail length currently cannot be faithfully estimated from basecalled sequences directly as it will often underestimate the actual poly(A) tail length. To accurately estimate the poly(A) tail length from Nanopore sequencing

data, we developed an R package — *tailfindr* [12]. The software uses basecalled `.fast5` files and annotates the reads with poly(A) tail estimates (for more details refer to Section 4).

In the following sections, we will describe how to successfully perform library preparation for native RNA sequencing using Nanopore, and how to use *tailfindr* to obtain isoform-specific poly(A) tail measurements from the obtained data.

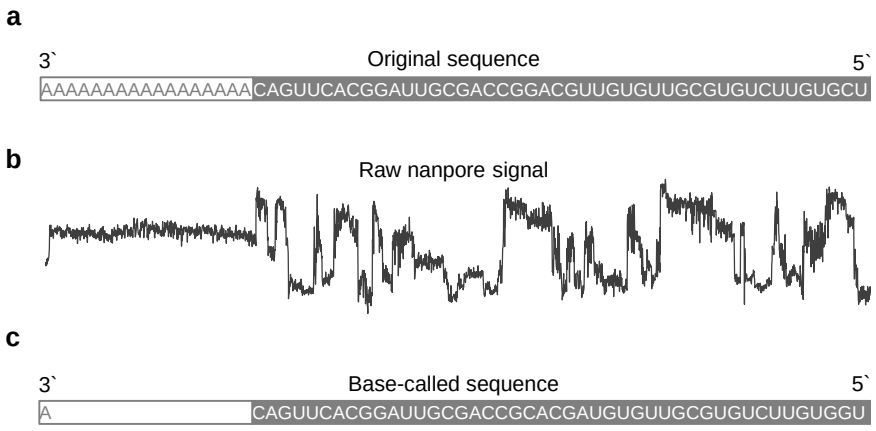

FIGURE 3 | Current basecalling algorithms underestimate poly(A) tail length. (**a**) A full-length mRNA with a 17-nt long poly(A) tail. (**b**) Raw signal generated by ONT sequencing when the sequence shown in (a) passes through the nanopore. Notice that the signal corresponding to the poly(A) tail is low-variance and monotonous. (**c**) Sequence predicted by the basecaller. Notice that the basecaller predicts only one adenosine in the poly(A) tail whereas the original sequence has 17 adenosines in the poly(A) region. This shows that although the raw signal for poly(A) tail is captured using Nanopore sequencing, it is not basecalled properly, preventing poly(A) tail length estimation directly from basecalling. N.B.: The sequences shown in this figure represent exemplified data.

3 | LIBRARY PREPARATION

ONT sequencing provides single-molecule long-read sequencing applications for RNA for the first time. However, the quality of the produced data and — most importantly — the quantity of data output directly depends on the quantity and quality of the provided RNA. It is therefore essential to make sure that enough RNA of good quality can be achieved prior to planning the experiment. Any RNA degradation not only affects the read length of the data obtained, but also makes library preparation inefficient, as it is based on poly(A)-dependent ligation of DNA adapters (**Figure 4**). Therefore, all experimental procedures upstream of sequencing should be reviewed for forces that could degrade molecules, such as vigorous shaking or pipetting. Furthermore, RNA should be extracted as fresh as possible, or alternatively stored at -80C in RNA storage medium (TRI reagent or RNALater). Extraction should be chosen to avoid any contaminants, as these could be detrimental to the sequencing chemistry. In our experience, silica-column based purification strategies not only degrade RNA by physical force, but also retain Guanidine-hydrochloride contamination. We thus recommend the use of phenol-chloroform extraction methods, such as the use of TRI reagent. These are more time-consuming, but in our hands yield higher quality RNA with minimal contaminant carry-over. An

example workflow for the use of TRI reagent for purification, as well as poly(A) enrichment based on the Poly(A)Purist MAG Kit, is described in an exemplary protocol at the end of this section.

Enriching for poly(A)-containing RNA is necessary in current ONT protocols, as the adapters are added specifically to the poly(A) tail. The addition of adapters happens through RNA ligation. However, the presence of high amounts of non-polyadenylated RNA (such as rRNA) can significantly impact the efficiency of adapter ligation by titrating the enzyme or adapters by non-productive binding events. Poly(A) enrichment based on magnets is the gold-standard experimental approach, but any other strategies that do not involve physical forces — such as vortexing, vigorous pipetting or column-based purification — would work as well.

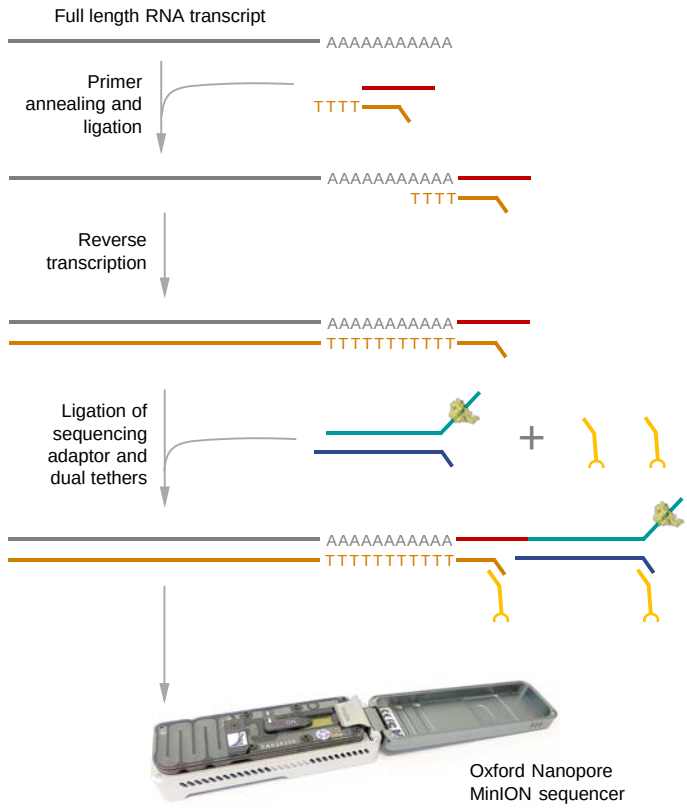

FIGURE 4 | Representation of ONT Direct-RNA library preparation protocol. The reverse transcription adaptor containing a T-overhang is ligated to the full-length RNA (shown in grey). This adaptor can only bind at the 3'-end of the transcript and initiates reverse transcription. The reverse transcription creates a DNA strand (shown in orange). In this way, an RNA-cDNA duplex is formed. Next, a sequencing adaptor containing the motor protein is ligated to the RNA-cDNA duplex along with dual tethers. During sequencing on a ONT MinION sequencer, these tethers anchor the DNA strand to the lipid bilayer membrane, which helps to efficiently feed the RNA strand through the pore.

The efficiency of library preparation solely depends on the efficiency of DNA-RNA ligation procedures. A schematic workflow of Nanopore Library preparation is provided in **Figure 4**. Any contaminant that reduces ligation efficiency will impact the sequencing performance of the library. It is thus important to follow the recommendations given in the Nanopore protocols (nanoporetech.com) for RNA quality and quantity measures. The only exception are ligation and bead purification incubation times, which we routinely double. A longer incubation time at room temperature might increase the risk of RNA degradation, yet also increases the chance of successful ligation or DNA binding or elution events to beads, which leads to a more efficient library preparation. Finally, it is crucial to proceed quickly from the final ligation to actual sequencing and avoid harsh chemicals and temperatures with the final library, as an active protein has been added whose function is essential for sequencing. An example protocol for library preparation including total RNA extraction and poly(A) enrichment together with notes arising from our library preparation experience can be found at `dx.doi.org/10.17504/protocols.io.9cjh2un`.

4 | BIOINFORMATICS ANALYSIS

To accurately estimate the poly(A) tail length from Nanopore native RNA sequencing data, we developed an R package — *tailfindr* [12]. Briefly, *tailfindr* estimates poly(A) tail length by first locating the monotonous stretch of current signal corresponding to the poly(A) tail within the raw signal, and then calculating its duration in samples (see **Figure 5**). Next, a read-specific translocation rate is computed; it specifies the average of samples per nucleotide translocation. After estimating this translocation rate, it is used to normalize the tail length in samples found earlier to yield tail length in nucleotides. During all these steps, *tailfindr* only needs basecalled FAST5 files to estimate the poly(A) tail length, making it independent of downstream data processing and thus implementable in real-time data analysis pipelines. The following paragraphs will give you detailed instructions on how to use tailfindr towards obtaining isoform-specific poly(A) measurements from Nanopore native RNA sequencing.

4.1 | Requirements

4.1.1 | Test dataset

We extracted RNA from Zebrafish (Danio rerio) using the protocol described above, and sequenced it on a MinION sequencer. A subset of reads from this experiment can be downloaded from `tiny.cc/polya_rna_data`. We will now demonstrate the various steps involved in transcript-isoform specific poly(A) tail length assessment using this example dataset, but you can use your own dataset as well.

4.1.2 | Hardware requirements

The example dataset can be processed on any laptop or desktop computer running a UNIX-based operating system with at least 3GB of free disk space. For a large real-world dataset, it is recommended that the pipeline is run on a Linux cluster, or a powerful workstation. For accelerating the basecalling speed, GPUs can be used. For more details on

which GPUs are compatible with the current basecaller, please refer to this link: `https://community.nanoporetech.com/posts/guppy-3-0-gpu-recommendati` (requires community login).

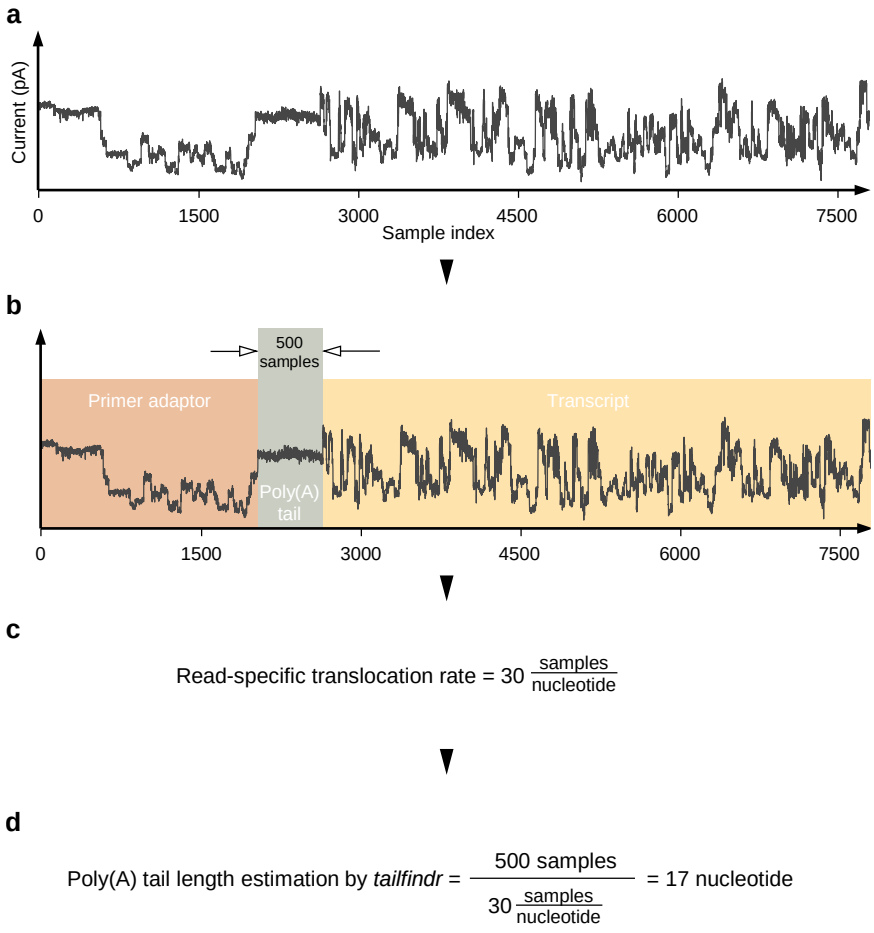

FIGURE 5 | A simplified view of how *tailfindr* estimates poly(A) tail length. (**a**) Complete raw signal corresponding to an RNA transcript translocating through the pore. The signal consists of a series of current samples measured in picoAmperes (pA). (**b**) *tailfindr* first locates the monotonous signal corresponding to the poly(A) tail (highlighted in brown). In this example, the segment is 500 samples long. (**c**) Next, *tailfindr* estimates the read-specific translocation rate, i.e., the average number of samples generated per nucleotide in a given read. (**d**) Poly(A) length is calculated by dividing the tail length in samples by the read-specific translocation rate.

4.1.3 | *Software requirements*

Software requirements The following software should be installed on the analysis computer:

1. Python 3 environment
2. R (version 3.5.3 or greater)
3. Git

4.2 | Data analysis pipeline

There are various steps involved in going from raw reads produced by ONT sequencing to transcript isoform-specific poly(A) tail length assignment, as shown in **Figure 6**. We will now describe each of these steps in detail.

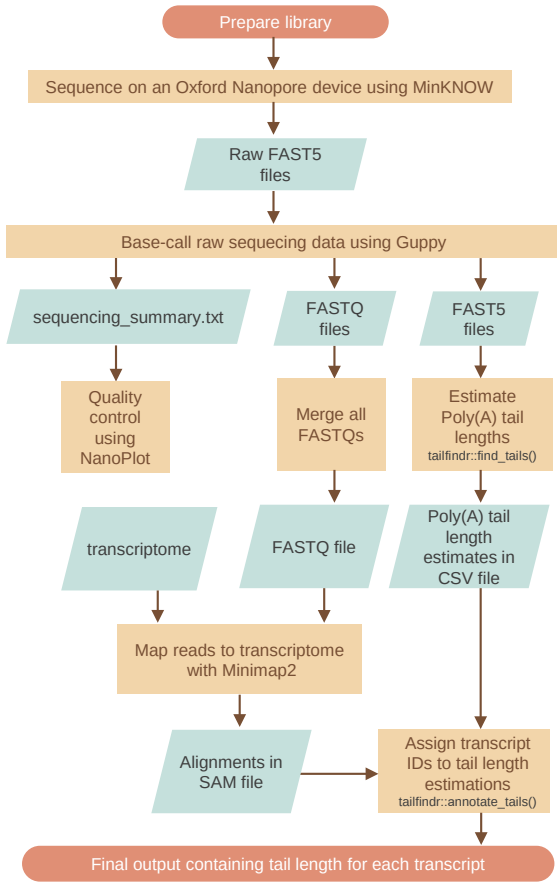

FIGURE 6 | Flowchart for poly(A) tail length estimation using Nanopore sequencing and *tailfindr*.

4.2.1 | *Basecalling*

Nanopore sequencing produces raw FAST5 files that record the current signal through the pore as an RNA molecule translocates through it (see **Figure 7a**). The first step is to basecall this raw signal to find the nucleotide sequence corresponding to the recorded current. There are many basecallers that can do this; please refer to [22] for a review on this topic. Some of the basecallers have been developed by ONT, while others are developed by Nanopore users. Albacore is a widely-used basecaller developed by ONT. Guppy — a recently released basecaller, also developed by ONT — has now replaced Albacore because it has better basecalling performance, and is faster than its predecessor. We recommend using the latest version of Guppy which can be downloaded from the ONT Community website `https://community.nanoporetech.com/downloads`.

Basecalling a raw FAST5 file using Guppy will add a Basecall_1D_000 group to the FAST5 file hierarchy (see **Figure 7b**). This basecall group contains a Move table (Events table in case of Albacore) which is used by *tailfindr* to compute the read-specific translocation rate. The structure of a FAST5 file — raw or basecalled — can be easily explored by opening it in HDFView (`https://www.hdfgroup.org/downloads/hdfview/`).

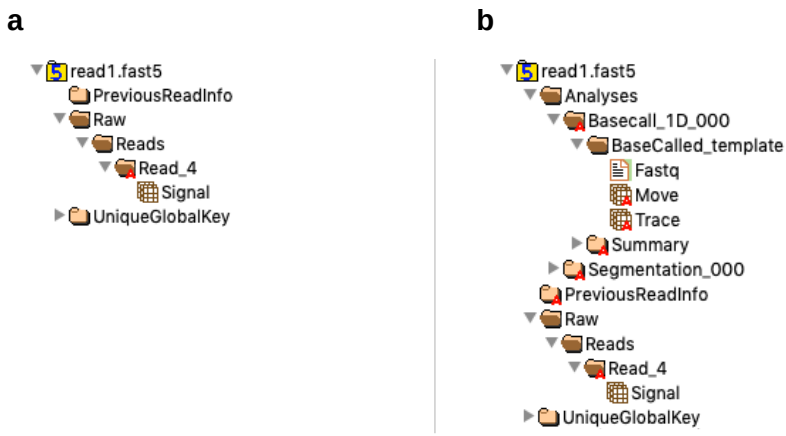

FIGURE 7 | Structure of a FAST5 file as displayed by HDFView software. (**a**) File structure of a raw FAST5 file generated by MinKNOW. (**b**) The same file as in (a) after basecalling by Guppy. During basecalling a new FAST5 is generated that contains not only raw signal data, but also additional basecalling information. Notice how additional levels of information (mintinlinebashAnalyses, `Basecall_1D_000` etc.) have now been added in this new file.

Guppy has both CPU and GPU versions. If you have access to an Nvidia GPU, then install and use the GPU version of Guppy, as it is faster to basecall on GPUs compared to CPUs. Here, we will demonstrate basecalling using the CPU version of Guppy (see Note 2 on where to get the latest version of Guppy). Assuming that you have a Quad Core processor (with 2 threads per processor; 8 threads in total) and 16GB of RAM, basecalling can be done by executing the following on the command line:

```
guppy_basecaller \
        --config rna_r9.4.1_70bps_hac.cfg \
        --input_path \path\to\raw\reads\folder \
        --recursive \
        --save_path \path\to\save\basecalled\data\to \
        --fast5_out \
        --trim_strategy none \
        --num_callers 1 \
        --cpu_threads_per_caller 8 \
        2>&1 | tee logfile.txt
```

Parameter description

`--config` specify the model configuration to be used during basecalling. In this case, we have chosen the "high accuracy" (*hac*) RNA model for pore version 9.4.1. The *hac* models yield more accurate basecalls at the cost of basecalling speed.

Refer to:

- *Note 3 for choosing a faster basecalling model.*
- *Note 4 for selecting an appropriate config file for your experiment in case you are not sure.*
- *Note 5 if your data is from a legacy RNA kit.*

`--input_path` specify the path of the folder containing raw FAST5 files produced by the ONT sequencing platform. When using the example dataset, extract it first, and then specify the path of the extracted directory here.

`--recursive` specifies that the `input_path` directory should be recursively searched to discover all raw FAST5 files within any subfolders.

`--save_path` specify the path of the directory where basecalled file should be stored.

`--fast5_out` specifies that in addition to the FASTQ files, the basecaller should also output FAST5 files. Basecalled files containing FAST5 output is essential for *tailfindr* to calculate the read-specific translocation rate for normalizing the poly(A) tail length.

`--trim_strategy` should be set to none so that the basecaller does not trim off the adaptor sequence that was added to the 3' end of the poly(A)+ RNA.

`--num_callers` specifies how many basecallers to use in parallel. `--cpu_threads_per_caller` specifies how many threads should be used per basecaller. In general, `num_callers * cpu_threads_per_caller` should not exceed the total number of threads available on the machine. Furthermore, there must be at least `4GB + 1GB * num_callers` RAM available. In our case, both these criteria are satisfied for the machine that we are using. For more exhaustive information on these settings, please refer to the document "Guppy basecaller and Guppy basecaller server" (`https://`

`community.nanoporetech.com/protocols/Guppy-protocol/v/gpb_2003_v1_revm_14dec2018/guppy-basecaller-and-guppy-basecaller-server`) on Nanopore Community.

`2>&1 | tee logfile.txt` specifies that the output, and any errors produced by the command, should be saved in a text file in addition to being displayed in the terminal. It is a good practice to do this for troubleshooting in case of a computer crash, power failure etc.

After successfully running the above, the basecalled FASTQ and FAST5 file can be found in the directory as specified in `save_path`. The structure of this directory is depicted in **Figure 8**. Please refer to Note 6 to find how the structure of this directory changes when multi-fast5 files are basecalled.

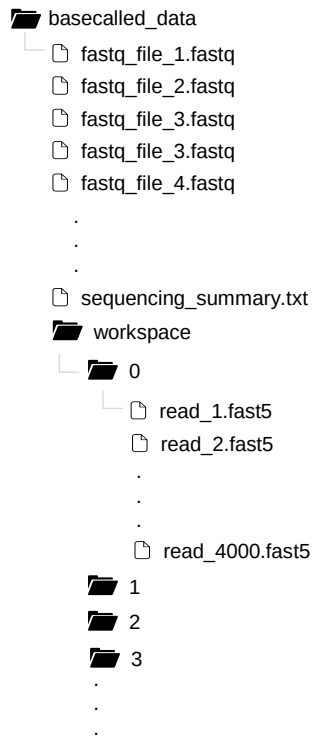

FIGURE 8 | Structure of the output directory produced by the Guppy basecaller. Each FASTQ file in the output of the basecaller contains sequence and quality scores for 4000 (default) reads. The `sequencing_summary.txt` file contains a summary of useful basecalling information, which is used by tools such as NanoPlot. The workspace folder contains numbered subfolders, each of which contain 4000 basecalled FAST5 files, which are used by tools such as *tailfindr*.

4.2.2 | *Quality control after basecalling*

Running quality control checks after basecalling is an optional but recommended step as it can reveal important information about the sequencing run such as the length of the reads (**Figure 9a**), sequencing performance over time (**Figure 9b**), and the quality of the reads (**Figure 9c**).

There are many tools to perform QC on Nanopore data, but the ones that produce the most informative plots are NanoPlot (`https://github.com/wdecoster/NanoPlot`) and PycoQC (`https://a-slide.github.io/pycoQC/`) [23, 24].

Here, we will use NanoPlot to perform quality control checks on the basecalled data. NanoPlot requires only the `sequencing_summary.txt` file produced by Guppy. To run NanoPlot, first activate a Python 3 environment, and then run the following in the command line:

```
NanoPlot \
        --summary \path\to\sequencing_summary.txt \
        --outdir \output\path \
        --loglength
```

Parameter description

`--summary path` of the summary file generated by Guppy.
`--outdir` path of the directory where NanoPlot output should be saved.
`--loglength` specifies that the read lengths should scaled logarithmically in the plots.

The output of NanoPlot is an HTML file that can be viewed in any browser of your choice.

4.2.3 | *Installing and running tailfindr*

We are now ready to estimate poly(A) tail lengths in the basecalled data using *tailfindr*. Please refer to its documentation (`https://github.com/adnaniazi/tailfindr`) to learn how to install it. After installing *tailfindr*, poly(A) tail lengths can be estimated by using the following commands in R:

```
library(tailfindr)
df <- find_tails(fast5_dir = '/path/to/basecalled_data',
                 save_dir = '/path/to/save/directory/',
                 csv_filename = 'rna_tails.csv',
                 num_cores = 2)
```

tailfindr discovers all FAST5 files recursively within the `fast5_dir`. The resulting CSV file — as specified in the `csv_filename` parameter — is saved in the `save_dir`. `num_cores` specifies the number of physical cores on the machine to be used when running *tailfindr*.

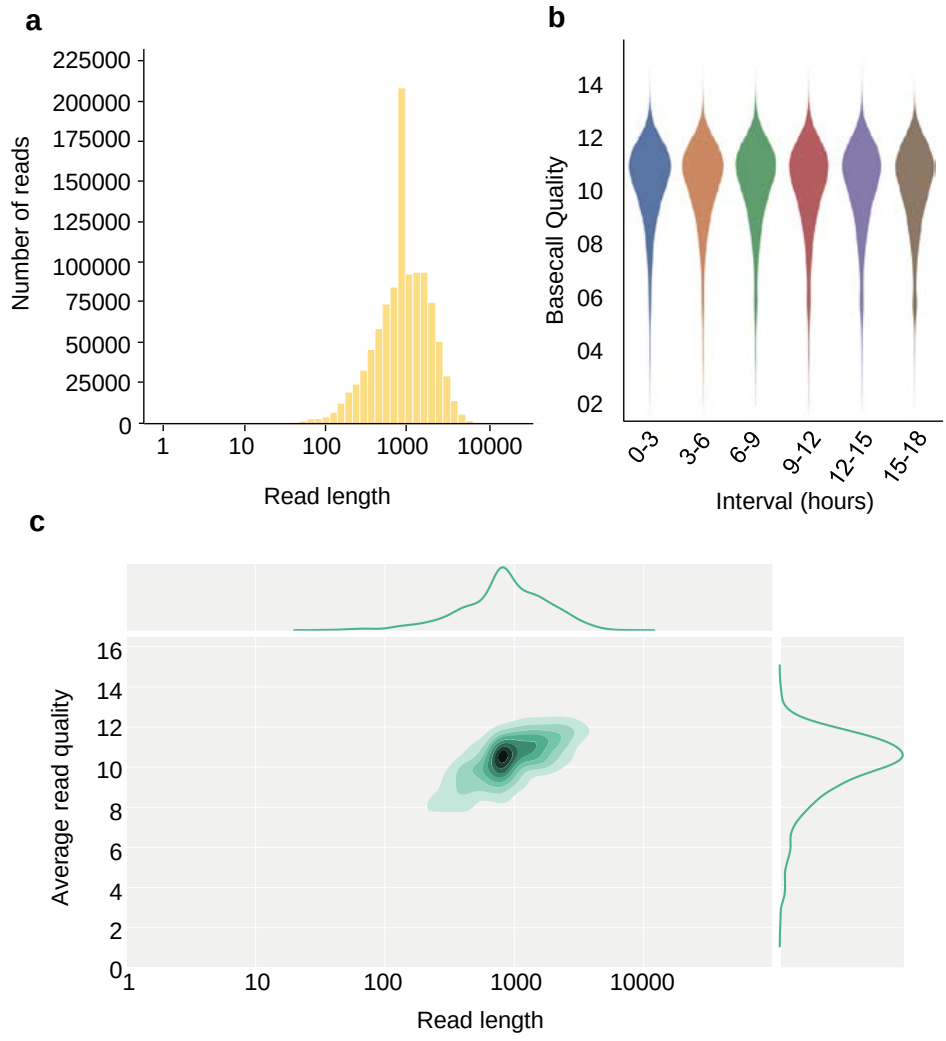

FIGURE 9 | A subset of figures generated by NanoPlot. (**a**) Read length histogram. This plot can be useful in understanding if RNA degradation significantly affected the sample. This particular histogram was generated for a sequencing run in which Zebrafish RNA was spiked with a synthetic GFP RNA construct of approx. 800bp in length. The spike in the histogram around 800 represents these GFP reads, and the background represents the read length distribution for the Zebrafish transcriptome. (**b**) Basecall quality vs. time of sequencing. This plot is useful in assessing if the sequencing chemistry — which might degrade over time — is having an adverse effect on the quality of the reads. Ideally, the basecalling quality should not drop dramatically during the sequencing run. (**c**) Read length vs. average read quality plot. It is useful in understanding how the read quality varies over read length. In a good sequencing run, the read quality for the majority of the reads should be around 8-14 for RNA (9-20 for DNA). Higher reads quality are good, and lower read qualities for majority of reads might warrant revisiting the library preparation steps and figuring out what might have gone wrong.

Please refer to:

- *Note 7 if you are running tailfindr on MinKNOW Live-basecalled data.*
- *Note 8 if you want to generate plots highlighting the poly(A) tail region in the raw current data*
- *Note 9 on how to use tailfindr for estimating poly(A)/(T) length in cDNA data*

The output of *tailfindr* is CSV file contain six columns as described in **Table 1**.

TABLE 1 | Description of columns in the CSV output of *tailfindr*

Column name	Column type	Description
`read_id`	character	Read ID as given in the FAST5 file
`tail_start`	numeric	Sample index of start site of the tail in raw data
`tail_end`	numeric	Sample index of end site of the tail in raw data
`samples_per_nt`	numeric	Read-specific translocation rate in terms of samples per nucleotide
`tail_length`	numeric	Tail length in nucleotides. It is obtained by dividing the difference between `tail_end` and `tail_start` divided by `samples_per_nt`
`file_path`	character	Absolute path of the FAST5 file

Now that we have the poly(A) tail length for each read in the CSV file, it is possible to perform quality control checks of this data. For example, a distribution of poly(A) tail lengths can be plotted to see if it aligns with the expected distribution of poly(A) tail lengths. Furthermore, a distribution of the translocation rate `samples_per_nt` can also be plotted. Ideally, this distribution should be unimodal with no skew (see **Figure 12**).

4.2.4 | Concatenate FASTQ files

During the basecalling step, Guppy produced both FASTQ and FAST5 files. By default, each FASTQ file contains sequences of 4000 reads (see **Figure 8**). Downstream processing software, such as the mapper Minimap2 [25], require only a single FASTQ file as input. Therefore, all FASTQ files produced by Guppy should be concatenated. Execute the following script in command line to combine all FASTQ file into one:

```
BASECALLED_DATA_PATH=/directory/containing/basecalled/data
OUTPUT_PATH=/directory/where/concatenated/fastq/is/to/be/saved
# Do not edit the code below this line
cd $BASECALLED_DATA_PATH
find ${BASECALLED_DATA_PATH} -name '*.fastq' | cat > ${OUTPUT_PATH}/filenames.txt
{ xargs cat < ${OUTPUT_PATH}/filenames.txt ; } > ${OUTPUT_PATH}/all_reads.fq
```

The above shell script searches `BASECALLED_DATA_PATH` directory for all files with `.fastq` extension and produces the following two files in the `OUTPUT_PATH` directory:

1. `filenames.txt` file that contains the names of all FASTQ files that were found in `BASECALLED_DATA_PATH` directory, and will be concatenated.

2. `all_reads.fq` file that contains the concatenated FASTQ sequences from all the FASTQ files recorded in the `filenames.txt` file.

4.2.5 | Alignment of data to transcriptome

Although we have estimated poly(A) tail lengths for all reads, we still do not know which transcript each of these reads originated from. To find the transcript identities, the reads must be mapped to the transcriptome of the organism from which the RNA was extracted (please refer to Note 10 if no reliable transcriptome is present and data should be mapped to a reference genome). The alignment information can then be merged with *tailfindr* output to associate the poly(A) tail length estimations to their respective transcript IDs.

To map the data to the transcriptome, we will use Minimap2 (`https://github.com/lh3/minimap2`) [25]. Minimap needs a single FASTQ file containing all the reads to be aligned. Run the following command in command line to invoke Minimap2:

```
minimap2 \
        -ax map-ont \
        /path/to/reference.fa \
        /path/to/all_reads.fq > /path/to/alignments.sam
        2>&1 | tee logfile.txt
```

Parameter description
Here is a description of the parameters used in the above command:
`-a` Specifies that CIGAR string and output alignments should be produced in the SAM format.
`-x` Use predefined settings for mapping. As each of these sequencing technologies differ in their insertion, deletion and error rates, there are a number of presets available in Minimap2 to choose from; `map-ont` is one of them. It specifies that Minimap2 should use alignment parameters fine-tuned for ONT sequencing data. This is because Minimap2 can align reads from Illumina, PacBio, and ONT sequencing.

4.2.6 | Annotating tailfindr output with transcript IDs

Now that we have the poly(A) tail length estimates from *tailfindr* in a CSV file, and the alignment information in a SAM file, we are ready to merge them together. This will annotate each read with its corresponding transcript ID. To do this, invoke *tailfindr's* builtin convenience function `annotate_tail()` in R:

```
df_annotate <-
annotate_tails(
sam_file = "/path/to/sam/file.sam",
           tails_csv_file = "/path/to/tails.csv",
           output_file = "/path/to/annotated_tails.csv"
)
```

This command will add three more columns to the input CSV file as described in **Table 2**.

TABLE 2 | A comparison of raw read accuracies between fast and high-accuracy basecalling models

Column name	Column type	Description
`transcript_id`	character	Transcript ID from from the transcriptome
`mapping_quality`	numeric	Mapping quality of the transcript
`sam_flag`	numeric	SAM flag

We now have the tail length and the corresponding transcript IDs in the `annotated_tails.csv` file. Thus we have successfully annotated each read with a transcript-isoform ID and a corresponding poly(A) tail length.

4.2.7 | What next?

Now that we have transcript-specific poly(A) tail lengths, we can do a number of things. For example, we can plot the distribution of poly(A) tail length of our dataset. We can also annotate the poly(A) tail length of a transcript with additional features such as gene name, gene length and its function. These steps are beyond the scope of this chapter, however, the reader should note that they can be easily done within R using the biomaRt Bioconductor package [26]. With gene name annotations, we can for instance generate a scatter plot of poly(A) tail length vs. gene length to see if there is any interesting relationship between the two. Additionally it is possible to plot poly(A) tail distributions from transcript isoforms of the same genes. Many further possibilities for data analysis exist, and implementation depends on the particular research question. The here described *tailfindr*-based pipeline provides the first step towards exploring these possibilities enabling the study of isoform-specific poly-A tail-dependent regulation.

5 | CONCLUSION

We have here demonstrated how long read ONT native RNA sequencing in combination with *tailfindr* can be used for transcriptome-wide isoform-specific poly(A) tail profiling. This method simplifies isoform-specific poly(A) tail measurements and avoids common caveats from short-read based sequencing approaches, namely (I) the possible introduction of amplification artefacts, (II) transcript isoform quantification based on statistical analysis of short reads spanning exon borders, and (III) elaborate and time-consuming sequencing sample preparation.

The portability and low investments for ONT sequencers, coupled with its ability to basecall and analyze sequencing data in real-time, enables anyone to sequence anything, anywhere. Additionally, the ability of direct RNA sequencing to detect any epigenetic modification in native RNA alleviates the need for separate assays for detecting each RNA modification. This enables future studies — both in the field and in a laboratory settings — to assay poly(A) tail length and RNA modifications in a single experiment. Such a transcriptome-wide holistic approach would provide a valuable insight in understanding RNA biology — one long molecule at a time.

6 | NOTES

1. Reverse-transcribing RNA into an RNA-cDNA duplex is an optional but recommended step. Without performing this step, the throughput will be about 30% lower and basecalling quality scores will also be slightly lower. Most likely this is caused by secondary RNA structure affecting pore translocation, making current signal more variable. Additionally, RNA degradation causes the average read length to be shorter. We recommend that you perform this step unless you have a very good reason not to.

2. We demonstrated how to basecall reads using the latest basecaller at the time of this writing provided by ONT — Guppy v3.2.4. However, it should be noted that the basecalling technology is constantly evolving. Always check ONT's Software Download section (`https://community.nanoporetech.com/downloads`) to read about the latest version of the basecaller and how to use it, as these might significantly increase basecalling accuracy and thus transcript isoform assignment.

3. If basecalling speed is more important than basecalling accuracy, then you can use the fast model configuration file `rna_r9.4.1_70bps_fast.cfg`. At the time of this writing, the fast models are approximately 5-8 times faster than the high accuracy model. **Table 3** shows a comparison between raw read accuracy of fast and high accuracy models.

TABLE 3 | A comparison of raw read accuracies between fast and high-accuracy basecalling models

Sample Type	Model Name	Raw Read Accuracy
DNA	Fast basecalling	92.1%
	High-accuracy basecalling	95.0%
RNA	Fast basecalling	88.6%
	High-accuracy basecalling	93.9%

4. If your experiment uses a pore version other than 9.4.1, then ensure that you specify a configuration file that matches the version of the pore used. You can find a list of all available configuration files for every flow cell and sequencing kit by executing the following command:

```
guppy_basecaller --print_workflows
```

If you are still unsure as to which configuration file to use, then, instead of specifying the configuration file, you can also let Guppy choose the appropriate configuration file for you. In this case, however, you have to specify the flowcell and kit arguments. Assuming if the flow cell and kit used in the experiment are FLO-MIN106 and SQK-RNA001,

respectively, then use the following command in command line to invoke Guppy:

```
guppy_basecaller \
        --flowcell FLO-MIN106 \
        --kit SQK-RNA001 \
        --input_path \path\to\raw\reads\folder \
        --recursive \
        --save_path \path\to\save\basecalled\data\to \
        --fast5_out \
        --trim_strategy none \
        --num_callers 1 \
        --cpu_threads_per_caller 8 \
        2>&1 | tee logfile.txt
```

5. The use of *tailfindr* is compatible with any RNA kit — including legacy kits — as all of these kits sequence both the transcript and the poly(A) tail. Thus, you can use *tailfindr* to find poly(A) tail lengths on any older RNA dataset where the initial aim of the study was something entirely different. For *tailfindr* to work, the only requirement is the availability of FAST5 files — either raw or basecalled; *tailfindr* cannot be used if the only file remaining from past experiments are FASTQ files. We recommend that you always re-basecall the old previously-basecalled FAST5 files before using *tailfindr* on it, and specify an appropriate value for `basecall_group` parameter when invoking *tailfindr*.

6. If the Raw FAST5 files produced by MinKNOW have only one read per FAST5 file, then reads within the workspace folder are arranged in numbered subfolders such that each folder contains 4000 FAST5 reads, as depicted in **Figure 8**. However, if the raw FAST5 files, produced by the sequencer, have multiple reads per FAST5 file, then there are no subfolders within workspace folder, and each basecalled FAST5 file in workspace folder will contain multiple reads (default is 4000) inside them.

7. MinKNOW — the data acquisition software used during sequencing on ONT sequencers — can basecall while the raw data is being acquired. This feature is called "MinKNOW Live Basecalling". Currently, *tailfindr* does not support MinKNOW live basecalled data because these FAST5 files do not contain Event/Move table (see **Figure 10a**). The Event/Move table is required by *tailfindr* to compute a read-specific translocation rate in order to normalize the poly(A) tail length in samples to yield poly(A) tail length in nucleotides.
To circumvent this problem, please basecall MinKNOW live-basecalled data again using standalone Guppy or Albacore. This will add an additional Basecall group (`Basecall_1D_001`) in the file structure of the FAST5 file (see **Figure 10b**). When using *tailfindr* on these re-basecalled reads, you must correctly specify the Basecall group containing the Event/Move table. For example, the read shown in **Figure 10**, the Event/Move table in the re-basecalled file is in the `Basecall_1D_001` in the FAST5 file structure hierarchy. *tailfindr*, in the case, should be invoked in R as shown below:

```
df <- find_tails(fast5_dir = '/path/to/basecalled_data',
                 save_dir = '/path/to/save/folder/',
                 basecall_group = 'Basecall_1D_001',
                 csv_filename = 'rna_tails.csv',
                 num_cores = 2)
```

The default value of `basecall_group` is `Basecall_1D_000`, which in the command above, has been changed to `Basecall_1D_001`.

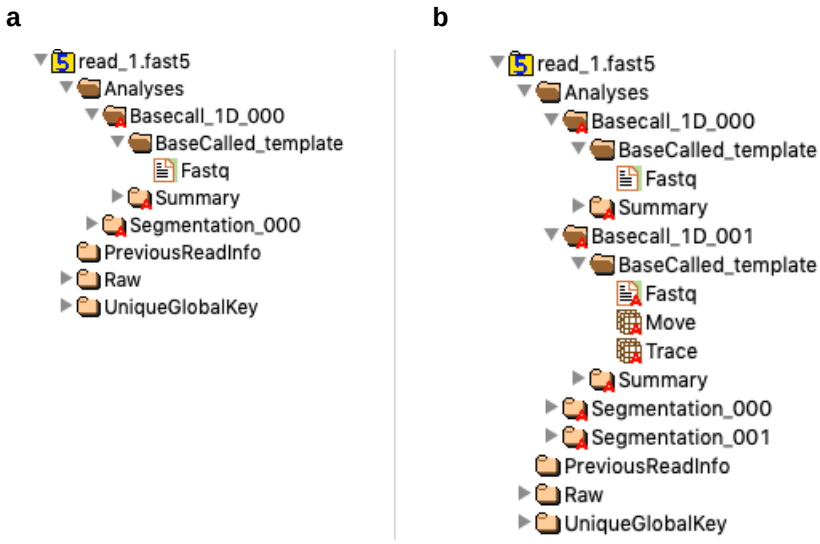

FIGURE 10 | Hierarchy of contents within basecalled FAST5 files as viewed through the HDFView software. (**a**) Contents of a MinKNOW live Basecalled read. Notice that under the `Basecall_1D_000` group, there is no Move table, which is required by *tailfindr* to find the read-specific translocation rate (**b**) Contents of the read shown in (a) after it has been basecalled again using standalone Guppy. Notice the addition of `Basecall_1D_001` group in the FAST5 file hierarchy, which now contains Move table. *tailfindr* should now be invoked with `basecall_group` parameter set to `'Basecall_1D_001'` to ensure that it can find the Move table.

8. *tailfindr* allows you to generate plots that show the tail location in the raw squiggle (see **Figure 11**). You can save these plots as interactive `.html` files by using 'rbokeh' as the `plotting_library`. You can then interactively zoom in on the tail region in the raw squiggle and see the exact location of the tail. To generate these plots, execute the following command in R:

```
df <- find_tails(fast5_dir = '/path/to/basecalled_data',
                 save_dir = '/path/to/save/folder/',
                 csv_filename = 'rna_tails.csv',
                 save_plots = TRUE,
                 plotting_library = 'rbokeh',
                 num_cores = 2)
```

Generating plots can slow down the performance of tailfindr. We recommend that you generate these plots only for a small subset of your reads.

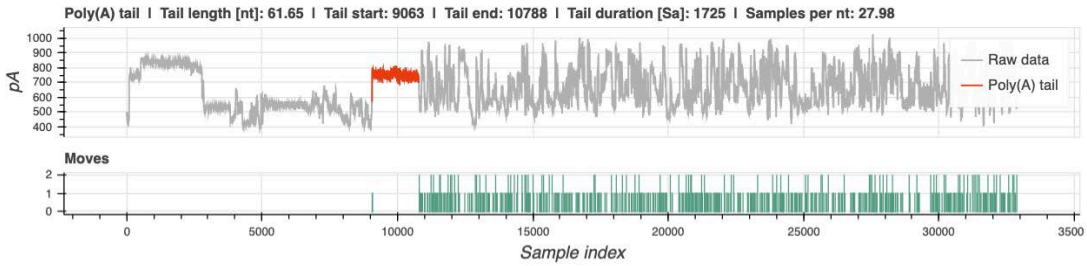

FIGURE 11 | A plot generated by *tailfindr*. The poly(A) tail is highlighted in red in the current trace. Each spike in the bottom panel shows the locations in the current trace where the basecaller has detected a nucleotide transition. Notice how the poly(A) tail region is devoid of any base transition. This is because the basecaller cannot distinguish when one adenosine base in the poly(A) tail ended and the next one started. It can detect a nucleotide transition only if a more diverse sequence is encountered.

9. Although we have demonstrated how to perform poly(A) tail profiling using Nanopore sequencing of native RNA, it is also possible to perform poly(A)/(T) profiling using complementary DNA (cDNA) sequencing data produced by Nanopore sequencing. Sequencing cDNA instead of RNA has many advantages:

a. cDNA is more stable compared to RNA which can degrade quickly if not handled very carefully at every step of library preparation protocol

b. cDNA sequencing requires less starting material compared to RNA sequencing

c. cDNA sequencing on Nanopore devices produces 10 times more data per flowcell compared to RNA sequencing because of the faster motor protein, and

d. poly(A) tail length estimates in DNA are more robust compared to RNA because the motor protein used in DNA sequencing ratchets the DNA at a more controlled speed compared to the motor protein used in RNA sequencing (see **Figure 12**).

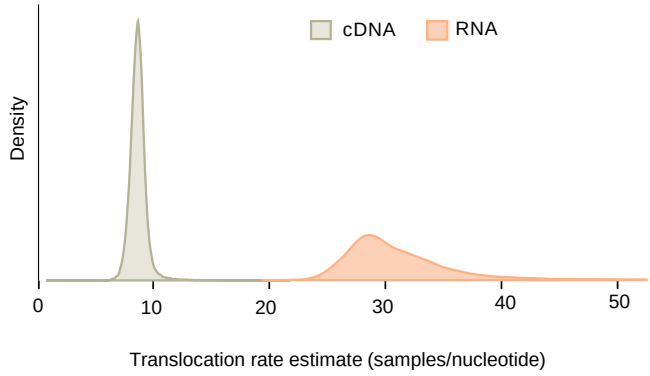

FIGURE 12 | Comparison of cDNA and RNA translocation rate estimates. RNA translocates at a slower speed compared to DNA. Furthermore, the spread in RNA translocation rate is greater than that of cDNA. This in turn translates to more spread in RNA poly(A) tail lengths compared to cDNA poly(A)/(T) tail lengths.

For more information on poly(A)/(T) profiling in cDNA, please refer to the *tailfindr* paper [12] and documentation on GitHub.

10. Reads from RNA sequencing can be mapped either to a reference transcriptome, or to a genome. The latter is more cumbersome, but could yield the identification of new transcript isoforms. This is especially useful if the reference transcriptome is known to be erroneous and is being assessed for the first time by long-read sequencing. For aligning reads to a genome with Minimap2, use the following command:

```
minimap2 \
        -ax splice -uf -k14 \
        /path/to/reference_genome.fa \
        /path/to/all_reads.fq > /path/to/alignments.sam
        2>&1 | tee logfile.txt
```

Parameter description

Here is a description of additional parameters in the above command:

`-splice` Specifies that spliced alignment should be done.

`-uf` By default, spliced alignment assumes the read orientation relative to the transcript strand is unknown and therefore it tries two rounds of alignment to infer the read orientation. This flag forces Minimap2 to consider only the forward transcript strand during mapping.

`-k14` For noisy Nanopore Direct RNA-seq reads, it is recommended to use a smaller k-mer size for increased sensitivity to the first or the last exons. Default value of k-mer size is 15.

References

[1] Bardwell VJ, Zarkower D, Edmonds M, Wickens M. The enzyme that adds poly(A) to mRNAs is a classical poly(A) polymerase. Mol Cell Biol 1990 Feb;10(2):846–849.

[2] Huang Y, Carmichael GG. Role of polyadenylation in nucleocytoplasmic transport of mRNA. Mol Cell Biol 1996 Apr;16(4):1534–1542.

[3] Meyer S, Temme C, Wahle E. Messenger RNA turnover in eukaryotes: pathways and enzymes. Crit Rev Biochem Mol Biol 2004 Jul;39(4):197–216.

[4] Beilharz TH, Preiss T. Widespread use of poly(A) tail length control to accentuate expression of the yeast transcriptome. RNA 2007 Jul;13(7):982–997.

[5] Subtelny AO, Eichhorn SW, Chen GR, Sive H, Bartel DP. Poly(A)-tail profiling reveals an embryonic switch in translational control. Nature 2014 Apr;508(7494):66–71.

[6] Lima SA, Chipman LB, Nicholson AL, Chen YH, Yee BA, Yeo GW, et al. Short poly(A) tails are a conserved feature of highly expressed genes. Nat Struct Mol Biol 2017 Dec;24(12):1057–1063.

[7] Chang H, Lim J, Ha M, Kim VN. TAIL-seq: genome-wide determination of poly(A) tail length and 3' end modifications. Mol Cell 2014 Mar;53(6):1044–1052.

[8] Woo YM, Kwak Y, Namkoong S, Kristjánsdóttir K, Lee SH, Lee JH, et al. TED-Seq Identifies the Dynamics of Poly(A) Length during ER Stress. Cell Rep 2018 Sep;24(13):3630–3641.e7.

[9] Hite JM, Eckert KA, Cheng KC. Factors Affecting Fidelity of DNA Synthesis During PCR Amplification of d(C-A)n•d(G-T)n Microsatellite Repeats. Nucleic Acids Res 1996 Jun;24(12):2429–2434.

[10] Murray EL, Schoenberg DR. Assays for determining poly(A) tail length and the polarity of mRNA decay in mammalian cells. Methods Enzymol 2008;448:483–504.

[11] Hommelsheim CM, Frantzeskakis L, Huang M, Ülker B. PCR amplification of repetitive DNA: a limitation to genome editing technologies and many other applications. Sci Rep 2014 May;4:5052.

[12] Krause M, Niazi AM, Labun K, Torres Cleuren YN, Müller FS, Valen E. tailfindr: Alignment-free poly(A) length measurement for Oxford Nanopore RNA and DNA sequencing. RNA 2019 Jul;.

[13] Legnini I, Alles J, Karaiskos N, Ayoub S, Rajewsky N. FLAM-seq: full-length mRNA sequencing reveals principles of poly(A) tail length control. Nat Methods 2019 Sep;16(9):879–886.

[14] Workman RE, Tang AD, Tang PS, Jain M, Tyson JR, Razaghi R, et al. Nanopore native RNA sequencing of a human poly(A) transcriptome. Nat Methods 2019 Nov;.

[15] Byrne A, Cole C, Volden R, Vollmers C. Realizing the potential of full-length transcriptome sequencing. Philos Trans R Soc Lond B Biol Sci 2019 Nov;374(1786):20190097.

[16] Garalde DR, Snell EA, Jachimowicz D, Sipos B, Lloyd JH, Bruce M, et al. Highly parallel direct RNA sequencing on an array of nanopores. Nat Methods 2018 Mar;15(3):201–206.

[17] Liu H, Begik O, Lucas MC, Mason CE, Schwartz S, Mattick JS, et al., Accurate detection of m6A RNA modifications in native RNA sequences;.

[18] Butler TZ, Pavlenok M, Derrington IM, Niederweis M, Gundlach JH. Single-molecule DNA detection with an engineered MspA protein nanopore. Proc Natl Acad Sci U S A 2008 Dec;105(52):20647–20652.

[19] Cherf GM, Lieberman KR, Rashid H, Lam CE, Karplus K, Akeson M. Automated forward and reverse ratcheting of DNA in a nanopore at 5-Å precision. Nat Biotechnol 2012 Feb;30(4):344–348.

[20] Rang FJ, Kloosterman WP, de Ridder J. From squiggle to basepair: computational approaches for improving nanopore sequencing read accuracy. Genome Biol 2018 Jul;19(1):90.

[21] Jain M, Fiddes IT, Miga KH, Olsen HE, Paten B, Akeson M. Improved data analysis for the MinION nanopore sequencer. Nat Methods 2015 Apr;12(4):351–356.

[22] Wick RR, Judd LM, Holt KE. Performance of neural network basecalling tools for Oxford Nanopore sequencing. Genome Biol 2019 Jun;20(1):129.

[23] De Coster W, D'Hert S, Schultz DT, Cruts M, Van Broeckhoven C. NanoPack: visualizing and processing long-read sequencing data. Bioinformatics 2018 Aug;34(15):2666–2669.

[24] Leger A, Leonardi T. pycoQC, interactive quality control for Oxford Nanopore Sequencing. JOSS 2019 Feb;4(34):1236.

[25] Li H. Minimap2: pairwise alignment for nucleotide sequences. Bioinformatics 2018 Sep;34(18):3094–3100.

[26] Durinck S, Spellman PT, Birney E, Huber W. Mapping identifiers for the integration of genomic datasets with the R/Bioconductor package biomaRt. Nat Protoc 2009 Jul;4(8):1184–1191.