Real-time biochemical-free targeted sequencing of RNA species with RISER

Alexandra Sneddon[1,2,3], Agin Ravindran[1,2,3], Nadine Hein[4], Nikolay Shirokikh[3], Eduardo Eyras[1,2,3,*]

[1] EMBL Australia Partner Laboratory Network at the Australian National University, Canberra, Australia.

[2] Centre for Computational Biomedical Sciences, The John Curtin School of Medical Research, Australian National University, Canberra, Australia.

[3] The Shine-Dalgarno Centre for RNA Innovation, The John Curtin School of Medical Research, Australian National University, Canberra, Australia.

[4] ACRF Department of Cancer Biology and Therapeutics, The John Curtin School of Medical Research, Australian National University, Canberra, Australia.

* Correspondence to: eduardo.eyras@anu.edu.au

Abstract

We present the first biochemical-free technology for Real-time In Silico Enrichment of RNA species (RISER). RISER classifies RNA species from direct RNA nanopore signals, without the need for basecalling or a reference, and communicates with the sequencing hardware in real-time to enact in silico targeted RNA sequencing. We illustrate RISER for the enrichment and depletion of coding and non-coding RNA, demonstrating a 3.4-3.6x enrichment and 6.2-6.7x depletion of non-coding RNA in live sequencing experiments.

Main

Nanopore technology enables the sequencing of RNA at single-molecule resolution. The nucleotide sequence of the RNA molecule can be inferred through analysis of the current fluctuations caused by the RNA transiting the nanopore[1]. To enable the translocation of RNA through the pore, standard library preparation protocols for nanopore direct RNA sequencing (DRS) ligate an adaptor and attach a "motor" protein to the 3' end of transcripts that are natively or artificially 3'-polyadenylated (poly(A)$^+$) RNA[2]. The resultant libraries therefore typically contain a medley of RNA species including messenger RNA (mRNA) and long non-coding RNAs (lncRNAs), amongst others[2].

Such a mixture can be detrimental to studies where only a specific RNA species is of interest, since unwanted RNAs consume the available working time of the nanopores and thus limit sequencing capacity. This is especially problematic for lncRNAs, which generally have a low yet highly tissue-specific expression pattern that can be obscured by the highly abundant mRNA or mistaken for noise in RNA expression measurements[3]. Compounding this issue are the practical limitations of the DRS system, such as slow RNA translocation speed (~70 bases per second (bps) compared to 200-450 bps for DNA), pore inactivation during the run, and the possible degradation of RNA in the flow cell. Overall, these factors decrease read throughput across the run[1]. It is therefore imperative that for each nanopore the available sequencing time is concentrated on the RNA species of interest for the entire duration of the run to maximize the sequencing depth of relevant transcripts.

Thus far, efforts to target RNA sequencing have been limited to biochemical approaches[3,4], which require time-consuming and expensive specialized experimental protocols. They are also restricted in their applicability to a pre-determined set of specific transcripts or species. For example, CaptureSeq relies on the design of custom probes to hybridize against the transcripts of interest[5], thus requiring prior definition of targets. Similarly, Mt-Clipping is designed to specifically cleave the 3' polyA tail in mitochondrial RNAs[4] (mtRNA). No approach supports enrichment or depletion of an entire class of RNA such as mRNAs or non-coding RNAs. Furthermore, biochemical treatment has been shown to induce RNA degradation and reduce integrity, compromising the quality, length and content of the resultant reads[3,4,6].

A key advantage of Oxford Nanopore Technologies (ONT) sequencing platforms is the opportunity to control sequencing per molecule by reversing the voltage across the pore to eject the resident molecule. This functionality can be effectuated computationally via the ONT ReadUntil application programming interface (API), which allows third-party software to retrieve data from, and send commands to, individual pores in the sequencing hardware in real-time[7]. "Read until" control has thus far only been deployed for DNA sequencing and decisions have been founded on the comparison of partially sequenced molecules against user-defined targets[7]. This requires real-time basecalling and mapping to a reference, which are both compute- and memory-intensive. Although efforts have been made to optimize the mapping step[8] or make decisions from the signal directly[9,10], they are yet to be demonstrated for DRS. Additionally, the need for the user to define a set of targets precludes the opportunity to discover novel transcripts. This is of critical relevance for non-model organisms that lack an annotation reference or generally for organisms and samples without a genome reference of sufficient quality.

To address these limitations, we have developed RISER to enable the real-time in silico enrichment of RNA species during DRS (**Fig. 1a**). We have applied this technology to the problem of enriching protein-coding and non-coding RNAs. Since DRS always sequences RNA in the 3' to 5' direction, the start of the nanopore signal for every transcript corresponds to the 3' untranslated region (UTR) for protein coding RNAs, or the 3' end for non-coding RNAs. We hypothesized that due to the differences in molecular composition in the 3' end of protein-coding RNAs[11,12], the DRS signals from the 3' end would be sufficient to directly discriminate between coding and non-coding RNA species.

We prioritized real-time requirements in the design of RISER given the unique challenges in applying "read until" technology to DRS. DRS generally produces shorter reads than DNA sequencing[2,13] and potential formation of RNA structures on the trans-side of the pore may block ejection if enough of the molecule has transited. These constraints exacerbate the urgency of the reject operation for maximal efficiency. Thus, the input signal used to make the decision has to be short yet contain sufficient information for a correct decision. We found that the first four seconds of the signal (corresponding to approximately the first 280nt of a transcript from the 3' end) provide a good balance between input signal length and percentage of assessable reads (**Fig. 1b**). Longer inputs increased the number of reads that would escape through the pore before a decision can be made.

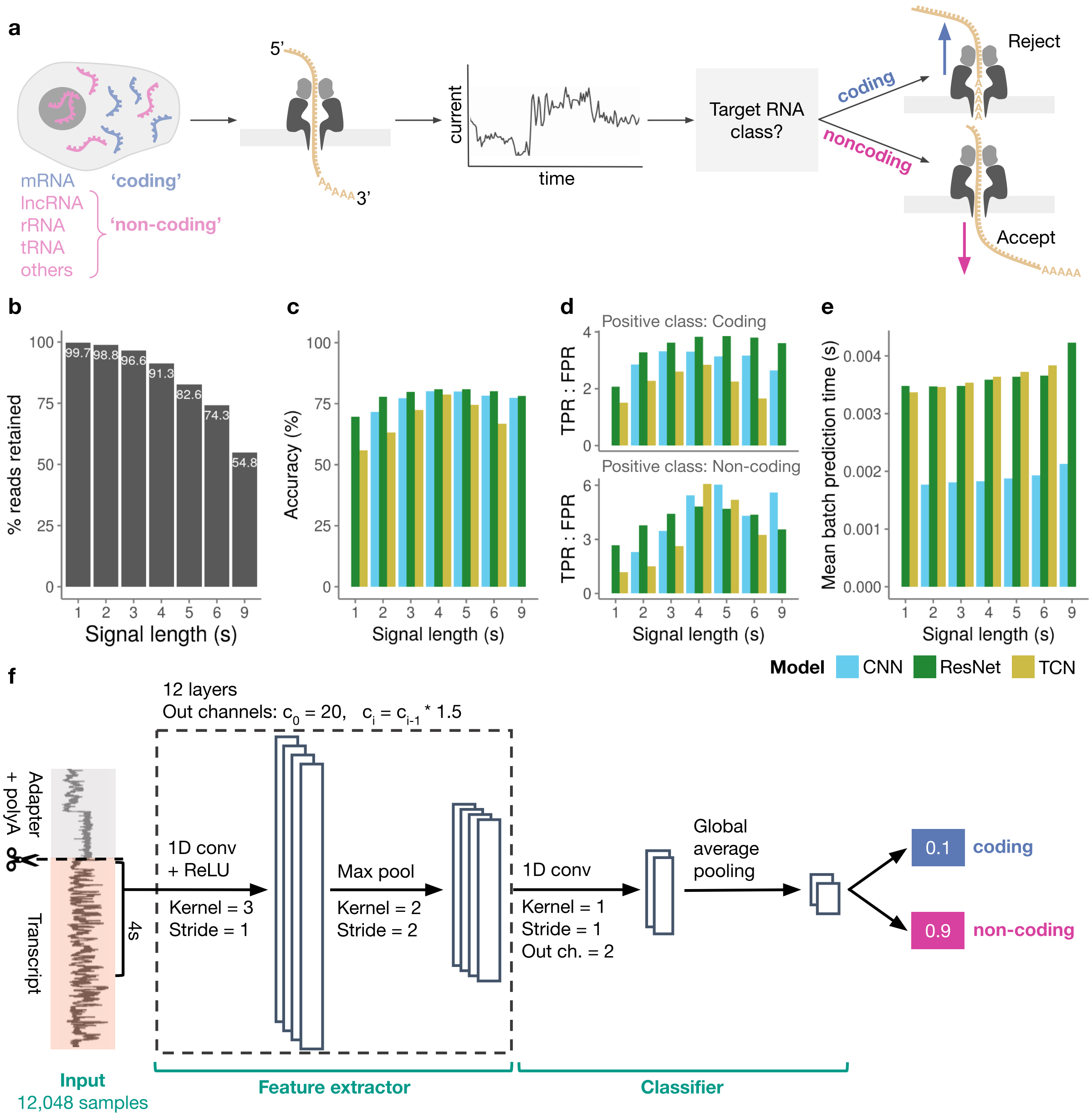


Figure 1: RISER identifies RNA species directly from raw nanopore signals. a, RISER classifies RNA molecules using the start of the nanopore signal, then sends an accept or reject decision to the sequencing hardware depending on the user-defined target RNA species. We show the specific application to the classification of coding and non-coding RNAs. **b,** Percentage of reads in training set with raw signals long enough to be assessed by RISER (y-axis) for each candidate input signal length expressed in seconds (x-axis). **c-e,** Model performance on the test set for each candidate input signal length (x-axes), color-coded by the three convolutional network architectures: "vanilla" convolutional neural network (CNN) (cyan), residual network (ResNet) (dark green), temporal convolutional network (TCN) (dark yellow). We show the accuracy (**c**), ratio of true positive rate (TPR) to false positive rate (FPR) for both coding (upper panel) and non-coding (lower panel) targets (**d**) and mean prediction time per batch (**e**). **f,** Neural network architecture for the CNN model selected to implement RISER. The model takes as input 4 seconds from the start of the raw nanopore signal, after the sequencing adapter and polyA tail have been processed, and outputs the probabilities that the signal corresponds to a coding or non-coding RNA.

To model the local signal patterns in the 3' end, we tested deep neural networks with convolutions, since they are particularly suitable for learning local temporal dependencies in time series[14]. We trained and tuned a "vanilla" convolutional neural network (CNN), a residual network (ResNet) and temporal convolutional network (TCN), then tested each on the binary classification coding/non-coding for a range of input signal lengths. All three models had the highest accuracy (**Fig. 1c**) and area under the receiver operating characteristic curve (AUROC) (**Ext. Data Fig. 1a**) at input signal lengths of 4s and 5s. Moreover, at these signal lengths, they also showed a higher ratio of (TPR) to false positive rate (FPR) for both coding and non-coding targets (**Fig. 1d**). This ratio provides a better estimate of the simultaneous maximization of accepted on-target molecules and rejected off-target molecules than the individual TPR, FPR and precision metrics (**Ext. Data Figs. 1b-d**). This further justified the selection of an input signal length of 4s. Finally, the CNN was approximately twice as fast as both the ResNet and TCN (**Fig. 1e**), while also achieving high accuracy and TPR:FPR for the 4s input. Thus, the CNN was selected as RISER's architecture (**Fig. 1f**).

To enable in silico targeted real-time RNA sequencing during DRS, we integrated RISER with the ReadUntil API. Prior to the RNA transcript signal, DRS signals retrieved from the API start with an adapter and a variable length polyA tail (**Ext. Data Fig. 2a**). We tested whether trimming a fixed amount from the start of each signal would allow accurate classification while also being efficient to execute in real-time. We expected this approach to work since the convolution operation is translation-invariant, thus the relevant components of the input signal will be recognized by the feature maps if they are present anywhere along the signal, regardless of their absolute position. Indeed, we found that trimming a fixed length of 2.2s from the start of the signals achieved a similar accuracy to trimming the exact signal length corresponding to the adapter and polyA tail (**Ext. Data Figs. 2b and 2c**).

To evaluate the RISER model's capacity to generalize to new samples, we tested an independent dataset from HeLa cells that had not been used for training, hyperparameter tuning or architecture selection. RISER achieved an accuracy of 88% as well as high precision (≥ 0.86) and TPR (≥ 0.86) with low FPR (≤ 0.14) for both coding and non-coding targets (**Fig. 2a**) and an AUROC of 0.96 (**Fig. 2b**). We next utilized the "playback" feature of the ONT MinKNOW software, which allows signals recorded from a previous sequencing run to be replayed as though they were being generated in real-time[7]. For testing, we replayed a sequencing run from a cancer cell line that had not been used for model development or evaluation. Since the playback functionality cannot mimic molecule ejection from the pore, it cannot be used for estimating enrichment directly. Instead, when a reject command is issued, the signal being replayed is prematurely terminated and so read length can be used as a proxy measure of enrichment potential. The expectation is that on-target reads will be significantly longer than off-target reads, which is the effect that we observed for RISER (**Ext. Data Fig. 3**).

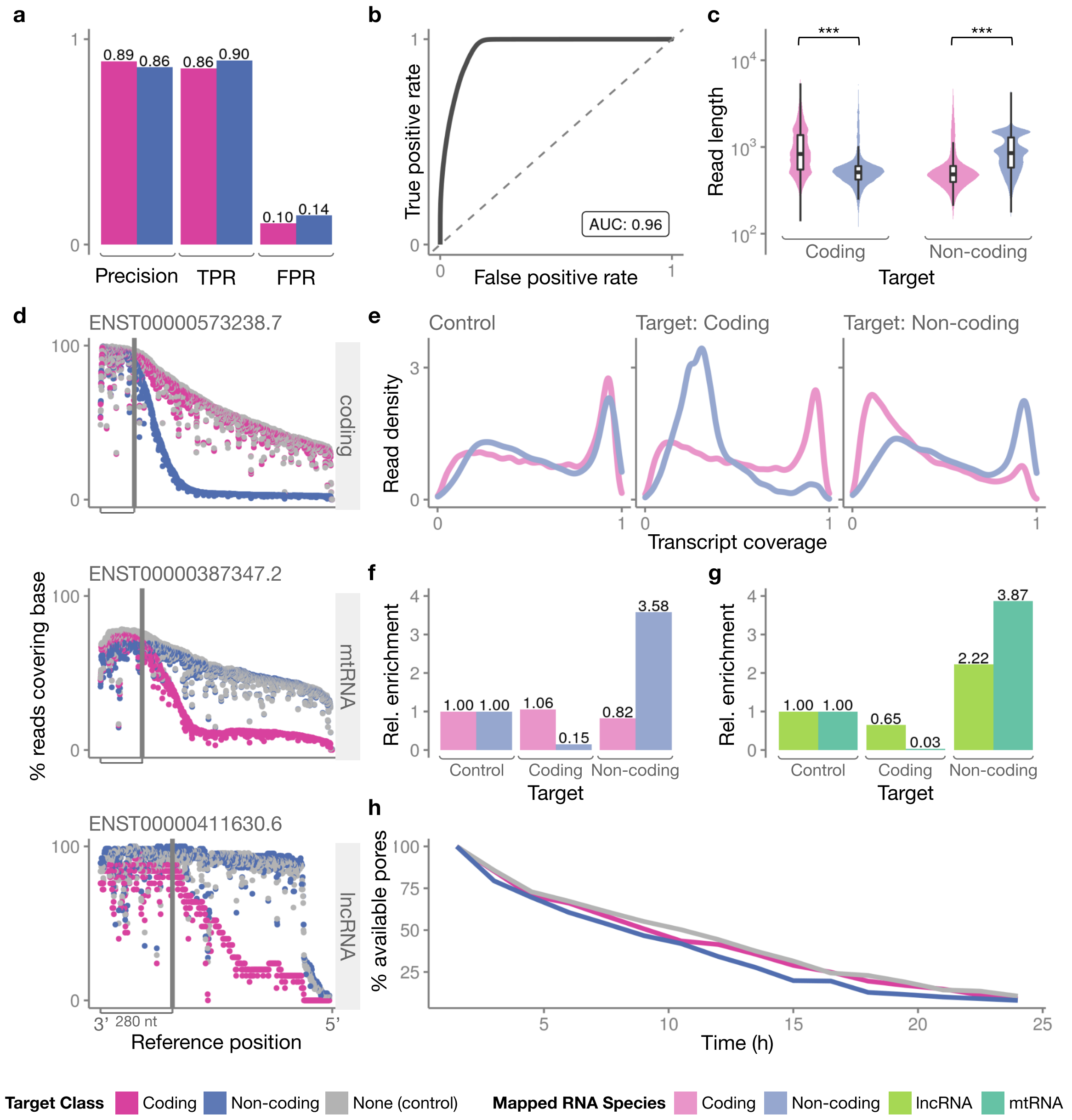


Figure 2: RISER enables real-time enrichment of RNA species. a-b, RISER model performance on a non-live independent experiment of poly(A)$^+$ RNA from HeLa cells. We show precision, TPR and FPR color-coded by target class (dark pink: coding, dark blue: non-coding) (**a**) and area under the receiver operating characteristic (AUROC) curve (**b**). **c-h,** RISER model performance during live sequencing of poly(A)$^+$ RNA from HeLa cells, using a MinION flow cell split into three RISER target classes: coding, non-coding, and none (i.e., control). The RISER target class is color-coded using a dark color scheme (dark pink: coding, dark blue: non-coding, grey: control), while the mapped RNA species is color-coded using a light color scheme (light pink: coding, light blue: non-coding). **c,** The distribution of read lengths for the mapped RNA species when the RISER target class was coding (left panel) and non-coding (right panel). Outliers were not included. For both targets, the read lengths of the mapped RNA species were compared using a Wilcoxon rank sum test with continuity correction (H_1: on-target > off-target) (p-value < 2.22E-16 for both target classes). **d,** Sequencing coverage per base for an example protein-coding RNA (upper panel), mtRNA (middle panel) and lncRNA (lower panel). The reference positions (x-axes) are ordered 3' to 5'. We indicate the position 280nt upstream of the 3' end, which approximately corresponds to the RISER input length. **e,** The distribution of the proportion of transcript covered by each sequenced read. **f,** The

enrichment of each mapped RNA species, calculated as the proportion of reads that RISER accepted through the pore for each mapped RNA species, relative to the proportions in the control where RISER was not used. **g,** The enrichment of lncRNA and mtRNA; calculation performed as in (**f**). **h**, Available pores as a percentage of the initial number of available pores, as recorded during the 1.5-hourly pore scans across the 24h duration of the sequencing run.

RISER was then run live during MinION sequencing of poly(A)$^+$ RNA from HeLa cells. The flow cell channels were split into three groups to simultaneously test the following conditions: (1) coding as the RISER target class, (2) non-coding as the RISER target class, and (3) no RISER as a control. We found that RISER significantly reduced the length of off-target reads (**Fig. 2c**). Notably, the coverage per base for off-target transcripts drops off approximately 280nt upstream of the 3' end of the transcripts, corresponding to the point of the RISER reject decision (**Fig. 2d**). This is consistent with our observation that RISER reduces the coverage of transcripts not belonging to the target class (**Fig. 2e**). Furthermore, among the reads that RISER accepted through the pore, the target RNA species was enriched while the off-target species was depleted. The largest impact was seen for non-coding RNA reads, which were depleted by a factor of 6.7x relative to the control run when the target class was coding and enriched by over 3.5x when the target class was non-coding (**Fig. 2f**). Importantly, we also found RISER enriched lncRNA reads by 2.2x and depleted mtRNA reads by 33x (**Fig. 2g**). Finally, we observed that RISER did not impact the percentage of available pores after 24h of sequencing (**Fig. 2h**). The same split flow cell experiments performed on poly(A)$^+$ RNA from HEK293 cells recapitulated the same effects of RISER on read length, transcript coverage, enrichment in terms of reads and pore availability (**Ext. Data Fig. 4**). We also observed an enrichment of on-target and depletion of off-target nucleotides for both HeLa and HEK293 experiments (**Ext. Data Fig. 5**).

While the flow cell splitting approach negated the effect of inter-flow cell variability, we also tested RISER at a broader scale using a whole flow cell per condition. Three technical replicates for each cell line were sequenced for 24h each. One replicate was run without RISER as a control, while the remaining two replicates were sequenced in parallel with RISER to target coding and non-coding RNAs, respectively. For both target classes, the effect of RISER on read length, transcript coverage, and enrichment in terms of reads (**Ext. Data Fig. 6**) and nucleotides (**Ext. Data Fig. 7**) were consistent with those observed with the flow cell splitting approach.

It is important to sequence as much on-target RNA as quickly as possible, since the theoretical maximum number of on-target reads that can be sequenced is fixed by the sample composition. This is a key motivation for using RISER rather than bioinformatically discarding off-target reads post-sequencing. As the standard DRS protocol selects for poly(A)$^+$ RNA, which is already enriched for mRNA (coding), RISER has a greater opportunity to affect the relative proportion of non-coding RNAs. Improvements in the sequencing hardware or library preparation to increase throughput, RNA stability or abundance of target RNA within the library are likely to magnify the benefits of targeted DRS using RISER.

RISER could be further improved through additional knowledge of the interactions between the pore and the RNA molecule and how this is influenced by the voltage polarity and amplitude. Knowing how pore efficiency decreases with more voltage switches could be used to optimize the frequency of reject commands per pore. Moreover, further understanding of the formation of RNA secondary structures during translocation would help to predict the success of ejection per transcript. At present, the risk of pore blockage is mitigated in RISER by limiting the number of reject commands to one per molecule.

In summary, RISER is the first technology for real-time in silico enrichment of RNA species. RISER alleviates the limitations of biochemical enrichment while also exploiting the capabilities of ReadUntil technology without the need for any form of target reference, to provide a level of sequencing control that is not possible with any existing biochemical or computational method. By operating directly on the raw signal, RISER obviates the need for basecalling or mapping and their associated computational expense. Importantly, RISER enables the enrichment of lowly expressed non-coding RNAs that are difficult to detect in an unbiased sequencing experiment. We have shown during multiple live sequencing experiments that RISER successfully rejects reads that do not belong to the target RNA class, freeing the pore to spend more time sequencing the target. Moreover, RISER's modular design (**Ext. Data Figs. 8 & 9**) facilitates easy adaptations of each software component, opening up a broad range of possible applications, such as the enrichment or depletion of other RNA classes, or the identification of RNA from different organisms. Finally, RISER is freely available to use through a simple and intuitive command-line tool (**Supp. Notes 1 and 2**) to empower RNA researchers with biochemical-free, real-time targeted RNA sequencing.

Methods

Model development

Raw signal data

MinION DRS reads from human heart as well as HEK293, GM24385 and HeLa cell lines were used for training and evaluating the model (**Supp. Table 1**). The HeLa dataset was reserved as an independent test set, while the remaining datasets (heart, HEK293-A, HEK293-B and GM24385) were used for developing the model and are hereafter referred to as the "model development datasets".

GM24385 sequencing

The lymphoblastoid cell line (LCL) GM24385 (received from Corielle Institute) was grown in RPMI1640 media (Gibco) supplemented with 15% Hi-FCS and 2 mM L-Glutamine in 6-well plates (Coning) under 5% CO_2. Cells were harvested at a density of 10^6 cells/ml. Cell pellet collection was performed by transferring GM24385 cell suspension into 15 ml conical centrifuge tubes (Falcon) and centrifuging at 500×g for 10 minutes at room temperature.

To isolate RNA from the cytoplasmic and nuclei fractions, 10^7 cells were lysed in 200 μl of the non-denaturing lysis buffer containing 25 mM HEPES-KOH (pH 7.6 at 25°C), 50 mM KCl, 5.1 mM $MgCl_2$, 2 mM DTT, 0.1 mM EDTA, 5% v/v glycerol, 2× Complete EDTA-free protease inhibitor and 0.5% v/v Igepal CA-630. Cells were resuspended in the lysis buffer through pipetting and RNasin Plus (Promega) was immediately added to the final concentration of 1 U/μl. Cell lysis was completed by passing the lysate 4× through a 20-gauge needle followed by passing it 4× through a 27-gauge needle. The cell lysate was then centrifuged at 1,000× g for 5 minutes at 4°C. The supernatant was transferred into a new 1.5 ml tube and mixed with 350 μl of the RA1 lysis buffer, followed by RNA isolation using columns (Macherey Nagel) to obtain the cytoplasmic RNA fraction. The pelleted nuclei were resuspended in 1 ml of ice-cold sterile PBS and 50 µl counted using cell counter (Beckman-Coulter). The nuclei suspension was spun again at 1,000× g for 5 minutes at 4°C, the supernatant aspirated and 6.6×10^6 nuclei were lysed in 700 µl of the RA1 RNA lysis buffer and isolated using columns (Macherey Nagel). RNA isolated from the cytoplasmic fraction was eluted from the columns in 80 µl, and from nuclei in 120 µl of RNase-free water and stored at -80°C.

For in vitro polyadenylation, ~9 µg of the RNA in 94 µl of deionised water or 25 mM HEPES-KOH (pH 7.6 at 25°C), 0.1 mM EDTA (HE) buffer were first denatured by incubating at 65°C for 3 minutes and immediately chilling in ice. The solution was then supplemented with 12 µl of 10× E. coli Poly(A) Polymerase buffer (New England Biolabs), 8 µl of 1 mM ATP and mixed. To the resultant solution, 3 µl of 40 U/µl RNasin Plus (Promega) and 3 µl of 5 U/µl E. coli Poly(A) Polymerase (New England Biolabs) were added and mixed, and the resultant mixture incubated at 37°C for 30 minutes. The eluate from in vitro polyadenylated RNA was further purified following the protocol previously described[15] for RNA cleanup with SPRI beads.

The DRS flow cell priming and library sequencing protocol was followed as previously described[16]. Two DRS runs for the nuclei RNA libraries and one DRS run for the cytoplasmic RNA library were conducted on a MinION Mk1B with R9.4.1 flow cells, for 44h, 29h and 72h, respectively, following the procedure previously described[15]. Version 20.10.3 of the MinKNOW software was used.

Data preparation

Fast5 files were basecalled, mapped and the mappings filtered as previously described[15]. Reads were then split by biotype into protein-coding and non-coding (all other biotypes) classes, with pseudogenes removed to ensure there were no common sequences between the two classes. For the model training and tuning datasets, each class was further split randomly into 80% training and 20% testing groups. To resolve class imbalance, the majority class (protein-coding) was undersampled to achieve a 50/50 class balance in each of the train and test sets so that the model was trained in an unbiased way.

The start of the raw nanopore signals were then trimmed using BoostNano[17] to remove the portions of signal that correspond to the sequencing adapter and polyA tail. The first n seconds (s) of the remaining transcript segment of signal was then extracted and normalized using median absolute deviation with outlier smoothing. Reads with transcript signal lengths less than n were discarded. For the model development datasets, 20% of the training data was reserved for hyperparameter tuning. For $n = 4$, the training and validation sets contained 1,073,720 and 268,428 signals, respectively.

Input signal length evaluation

The optimal input signal length n was determined by assessing the tradeoff between signal length and percentage of assessable reads. For this comparison we computed the percentage of reads in the training dataset that had a transcript signal length greater than or equal to the candidate input signal lengths of {1-6,9}s.

Candidate neural network architectures

Convolutional neural networks capture spatial structure in the input by using convolutions, which are computed as the dot product between a filter (also known as a "kernel", which is a matrix of learnable weights) and a portion of the input the same size as the filter (a "local receptive field"). The filter acts as a feature detector and by "sliding" the same filter across the input to produce a feature map, feature detection becomes translation invariant, i.e., the same feature can be found anywhere along the input length. The use of multiple filters ("channels") in each layer allows different features to be detected, while the use of multiple layers in the network allows hierarchies of features to be learned[18].

We explored variants of two convolutional architectures that have shown strong performance for 1D sequence modelling tasks; the Residual Neural Network (ResNet)[19] and Temporal Convolutional Network (TCN)[20] also comparing these against a "vanilla" convolutional network (CNN). For each architecture, we systematically tuned the hyperparameter configuration. All models were trained using binary cross-entropy loss and Adam optimization for up to 100 epochs (within a 48-hour time limit), after which their accuracy was evaluated on the validation set (**Supp. Table 2**). All models were built, trained and tested using PyTorch (v1.9.0)[21] with a single NVIDIA Tesla V100 graphics processing unit (GPU)

Residual network hyperparameter optimization

The ResNet architecture was designed[19] to overcome convergence issues when training deep networks. This was achieved by adding shortcut connections to the network, which directly propagate unmodified inputs to subsequent layers. The effect is a reduced backpropagation distance to mitigate gradient update instability, enabling the training of much deeper networks and the extraction of richer feature hierarchies than was previously possible[19].

We explored 33 variants of the following general ResNet architecture; the input vector was fed into a feature extractor layer composed of a 1D convolution with kernel size *k* and stride of 3 followed by batch normalization, rectified linear unit (ReLU) activation ($f(x) = \max(0, x)$) and max pooling (which computes the maximum value in each local receptive field to downsample the feature maps) with a kernel size and stride of 2. Following were *l* residual layers, with each layer *i* *(i=0,...,l-1)* containing $\boldsymbol{b} = \{b_i\}$ residual blocks using $\boldsymbol{c} = \{c_i\}$ channels. Residual blocks were either "bottleneck" or "basic" types, implemented as described in He et al. (2016)[19].

To determine the optimal values of *k*, *l*, ***b***, ***c*** and block type we experimented with ResNet-34 and ResNet-50 architectures[19] along with variants of these with fewer channels per layer to reduce overfitting. We also tested the SquiggleNet architecture[22] in its original form, before systematically varying each hyperparameter to find the optimal configuration for this new domain. We found that the basic block outperformed the bottleneck block and networks with a more gradual increase to a larger number of channels converged to a better loss minimum. To test the boundaries of this observation, we reduced b to 1 for every layer, set $c_0 = 20$, $c_i = \lfloor c_{i-1} * 1.5 \rfloor$ and $l = 10$, which was the maximum number of layers possible before the feature vectors became smaller than the receptive field. We also set $k = 19$ as in SquiggleNet. We found that this configuration achieved the highest accuracy on the validation set, trained using a batch size of 32 and initial learning rate of 0.001.

Temporal convolutional network hyperparameter optimization

Designed specifically for sequence modelling, TCNs[20] operate on input sequences using dilated causal convolutions; causality is to ensure predictions are based only on past information, while dilation allows the receptive field (RF) size to increase exponentially with network depth. When the network is sized appropriately, the last timestep in the final layer has the entire input sequence as its RF. Thus, classification predictions can be made using the last value in each channel. Residual connections are also employed to increase the depth and hence "memory" of the network. Bai *et al.* (2018) showed the TCN is more efficient and has greater memory than equivalent-capacity recurrent networks[20].

We tested 23 TCN models following the architecture described by Bai et al (2018)[20] to identify the optimal hyperparameter configuration, under the constraint that the last layer's RF covered the entire input length. As such, the number of layers *l*, kernel size *k* and dilation base *d* were varied such that:

$$RF = 1 + 2\sum_{i=1}^{l} 2^{d}(k-1) \geq 12048$$

The number of channels per layer *c* was also varied, with the observation that more channels significantly increased training time and network size and so for practical reasons was set at or below 256. Dropout was used to regularize the network and was another hyperparameter *r* that was optimized. The best model had parameters $l = 10$, $k = 11$, $d = 2$, c = 32, $r = 0.05$, trained using a batch size of 32 and initial learning rate of 0.0001.

"Vanilla" convolutional neural network hyperparameter optimization

We also tested 26 "vanilla" CNNs, hypothesizing that a simpler architecture may be more efficient yet still accurate. Each model was a variation of the following architecture: the input vector was fed into *l* convolutional layers, each of which was composed of *b* blocks of a 1D convolution with a stride of 1 and kernel size *k* followed by ReLU activation. Each layer ended with a max pooling layer with a kernel size and stride of 2. The number of channels c_i in layer *i* $(i=0,...,l-1)$ was also configured, increasing with network depth to capture higher-level, more complex features. The extracted features were then passed to a classifier *f*, which was either a simple 2-layer fully connected network with ReLU activation, a global average pooling (GAP) layer or global average pooling followed by a fully connected layer (GAP_FC). The model with highest accuracy on the validation set had the parameters $l = 12$, $b = 1$, $k = 3$, *f = GAP_FC* and $c_0 = 20$, $c_i = \lfloor c_{i-1} * 1.5 \rfloor$ and was trained using a batch size of 32 and initial learning rate of 0.0001.

Evaluation of candidate models

The ResNet, TCN and CNN models with the highest accuracy on the validation set were then evaluated on the reserved testing set, which comprised all test reads from the model development datasets. The performance metrics used were accuracy (percentage of correct predictions), true positive rate (TPR) (fraction of the positive class predicted correctly), false positive rate (FPR) (fraction of the negative class predicted incorrectly), precision (fraction of correct positive predictions) and area under the receiver operating characteristic (AUROC) (a holistic measure that considers TPR and FPR across all classification thresholds). TPR, FPR and precision were each calculated considering protein-coding as the target class, as well as non-coding as the target class, to assess classifier performance in both usage scenarios. We also computed the mean inference time per batch of test data (b = 32). Each model was evaluated for each of the candidate input signal lengths of {1-6,9}s, except for the CNN which could not handle an input signal length of 1s, while the receptive field of the TCN was insufficient to test an input signal length of 9s. Testing was conducted on a single-usage computer with 12 CPUs (Intel® Xeon® Platinum 8268) and one GPU (NVIDIA® Tesla® V100).

RISER software design

Integration with the ReadUntil API

The ONT ReadUntil API provides an interface to each pore in the sequencing hardware, allowing the user to request raw current data or reverse the pore voltage during sequencing to reject a molecule. Data is streamed in chunks of 1s by default, which get accumulated per read until RISER has received a long enough signal. RISER then discards the first portion of the signal corresponding to the sequencing adapter and 3' polyA tail, before passing the remaining 4s' worth of signal (equivalent to ~280nt) to the neural network model, which predicts the RNA class. If the prediction matches the target class specified by the user, then RISER allows the RNA to complete sequencing, otherwise it submits a reject ("unblock") request to the API.

If a reject request fails (e.g., due to secondary structure formation on the trans-side of the pore that prevents reversal through the pore), it is preferable to allow the molecule to complete translocation in the forward direction. This circumvents repeated futile ejection attempts that may potentially damage the pore. Therefore, RISER is made to only request rejection a maximum of once per molecule. To balance the frequency of reject requests with the frequency of data streaming, the request interval was set to 1s, the same as the chunk size. The RISER code is comprised of independent software components responsible for data preprocessing, ReadUntil API access, model inference and enrichment logic to facilitate ease of maintaining, modifying, or extending the code for different applications (**Ext. Data Figs. 8 & 9**). Further, we encapsulated access to the ReadUntil API with a wrapper class so that if ONT were to update or replace the API, any affected code is isolated.

Strategy for trimming sequencing adapter and polyA tail

We selected the sequencing adapter and polyA tail "trim length" based on the distribution of trim lengths computed by BoostNano[17] for the training dataset. We trimmed the start of every signal in our independent test set by each of the trim length distribution quartiles (**Ext. Data Fig. 2b**) and compared RISER's performance with using the exact trim length (computed by BoostNano) per signal. Performance metrics were accuracy, AUROC, precision, TPR and FPR for both coding and non-coding targets (**Ext. Data Fig. 2c**). Importantly, the training dataset included reads from both natively and synthetically polyadenylated RNAs, so the selected trim length is useful regardless of sample preparation method.

Command-line tool

We developed a simple command-line tool to run RISER (**Supp. Note 1**). This should be executed on the same computer as MinKNOW, during a sequencing run. The user must specify the RNA species to enrich for (currently either "coding" or "noncoding") as well as the duration to run RISER for, which will be generally less or equal to the MinKNOW run. If RISER stops earlier, the MinKNOW run will continue without enrichment. RISER run can be set to stop later. While this does not cause an error, RISER will not receive any data during this additional time. After MinKNOW finishes the first mux scan, RISER should be started. Advanced users also have the option of specifying their own model, an alternative signal input length or an alternative trim length for removing the polyA tail and sequencing adapter.

While RISER is running, it will output real-time progress updates to the console window from which it was run (**Supp. Note 2**). This includes a summary of the settings used, as well as a summary of the sequencing decisions made for each batch of reads received. A more detailed version of this information is written to a log file (**Supp. Note 3**). In addition, a .csv file will be generated that lists, for each read, its sequencing channel, the probability that RISER predicted it was coding or non-coding and the final accept or reject decision made (**Supp. Table 5**).

A "reject_all" script is provided to users to test that the communication with the ReadUntil API is working before they use RISER for a real sequencing run, similar to the "unblock-all" script provided by ReadFish[7]. This test can be performed while replaying a bulk FAST5 file with MinKNOW.

Testing on independent HeLa sample

The RISER model performance was evaluated using the reserved HeLa dataset, which was prepared as described in "Data Preparation". The resulting test set comprised 394,036 reads evenly split between protein-coding and non-coding classes. Performance was measured using accuracy, TPR, FPR, precision and AUROC. TPR, FPR and precision were each calculated considering both protein-coding and non-coding as the target class.

Testing in simulated sequencing environment

We used a bulk FAST5 file from a sequencing run of poly(A)$^+$ selected RNA from an REH cancer cell line for playback testing. Using the default MinKNOW (v21.11.9, core v4.5.4) run settings, we replayed the bulk file 3 times for 6 hours per condition: (1) without RISER (as a control), (2) with RISER targeting the coding class, and (3) with RISER targeting the non-coding class. Testing was performed on a desktop computer running Ubuntu 18.04 with one NVIDIA® GeForce® GTX 1650 GPU and python v3.6.9.

The sequenced reads were basecalled using Guppy with options "*--flowcell FLO-MIN106 --kit SQK-RNA002*", then mapped to the reference transcriptome (GRCh38.p13 release 34) by minimap2 (v2.17)[23] with options *"-ax map-ont --secondary=no -t 15*". For each of the RISER runs, the distributions of read lengths for mapped coding and non-coding RNAs were compared using a Wilcoxon rank sum test with continuity correction (H_1: on-target > off-target). Reads that mapped to the GENCODE "protein-coding" biotype were considered as "coding" RNAs, while reads that mapped to the GENCODE biotypes "Mt_rRNA", "Mt_tRNA", "miRNA", "misc_RNA", "rRNA", "scRNA", "snRNA", "snoRNA", "ribozyme", "sRNA", "scaRNA" and "lncRNA" were considered as "non-coding" RNAs.

Testing during live MinION sequencing runs

Sample preparation

HeLa cells (human cervical cancer) and HEK293 (human embryonic kidney) cells were purchased from American Type Culture Collection (ATCC). HeLa cells were confirmed *via* short tandem repeat (STR) profiling with CellBank Australia. HeLa cells were grown in DMEM medium (Gibco) supplemented with 10% fetal bovine serum (FBS) and 1× antibiotic-antimycotic solution (Sigma). HEK293 cells were supplier-certified and grown in DMEM media supplemented with 10% FBS. Cells were cultured following the protocol described in [15]. Cell cultures were propagated in a 1:3 split, with replenishment of media every 4 days. HeLa and HEK293 culture, cell pellet collection, cell lysis, polyadenylated RNA extraction and RNA cleanup with SPRI beads were performed as described previously[15].

MinION sequencing

The flow cell priming and library sequencing protocol were performed as described previously[15]. Nanopore sequencing was performed using an ONT MinION Mk1B with R9.4.1 flow cells, for 24 hours per run. Four runs were conducted for each of the HeLa and HEK293 cell lines. The flow cell was prepared and loaded as previously described[15]. For each run, the default settings for the MinKNOW software (v22.05.5, core v5.1.0) were used. The SQK-RNA002 kit was selected and the bulk file output was turned on.

RISER usage

The four HeLa runs were conducted as follows: (1) with RISER targeting the noncoding class, (2) with RISER targeting the coding class, (3) without RISER (as a control), and (4) the flow cell channels were split into three groups to simultaneously test three RISER target classes: coding, non-coding, and no target (control), and thereby remove the effect of inter-flow cell variability. The four HEK293 runs were performed the same way, however the split flow cell was run first. In each run, RISER was executed after the first MUX scan was completed. To target the coding class RISER was run with options "*--target coding --duration 24*", while to target the noncoding class RISER was run with options "*--target noncoding --duration 24*". For the split flow cell run, the channel numbers divisible by 3 were used to target the coding class, the channel numbers with remainder 1 when divided by 3 were used as a control without RISER, while remaining channels were used to target the noncoding class. Since there are 512 channels in the MinION flow cell, this approach assigns 170 channels per condition with 2 channels leftover. Reads from the additional channels were discarded bioinformatically after the run to achieve an equal number of channels per condition. All runs were performed on a computer running Ubuntu 20.04 with one NVIDIA GeForce GTX 1650 Ti GPU and python v3.8.10.

Analysis

Data preprocessing

Reads were first basecalled using Guppy with options "*--flowcell FLO-MIN106 --kit SQK-RNA002*", then mapped to the reference transcriptome (GRCh38.p13 release 34) by minimap2 (v2.17)[23] with options *"-ax map-ont -uf --secondary=no -t 15*". To retain only primary alignments to the forward strand, alignments were filtered using samtools[24] with option "*-F 2324*".

Transcript coverage (the fraction of the mapped transcript covered by the alignment) for each of the remaining alignments was then computed by parsing the alignment CIGAR string. Alignment matches ("M"), sequence matches ("=") and sequence mismatches ("X") were summed to compute the length of the aligned part of the transcript. Coverage was obtained by dividing this alignment length by the transcript length.

The remainder of analyses were performed using the sequence lengths output by Guppy, the filtered alignments, the computed transcript coverage and the decisions *.csv* file output by RISER. For all analyses, reads that mapped to the GENCODE "protein-coding" biotype were considered as "coding" RNAs, while reads that mapped to the GENCODE biotypes "Mt_rRNA", "Mt_tRNA", "miRNA", "misc_RNA", "rRNA", "scRNA", "snRNA", "snoRNA", "ribozyme", "sRNA", "scaRNA" and "lncRNA" were considered as "non-coding" RNAs.

Split flow cell analysis

For each of the three split flow cell conditions, the distributions of the lengths of reads assessed by RISER for mapped coding and non-coding RNAs were compared using a Wilcoxon rank sum test with continuity correction (H_1: on-target > off-target). Outliers were excluded. To plot the per-base transcript coverage for each of the example protein-coding, lncRNA and mtRNA transcripts, the read depth for each split flow cell condition was computed at each reference position using samtools (v1.10) depth (with default options). Percentage depth at each reference position was then calculated by dividing read depth by the total number of reads in the relevant condition. On the other hand, to obtain the transcript coverage per read, for each of the three split flow cell conditions, the distributions of transcript coverage per read for mapped coding and non-coding RNAs were computed.

Read enrichment was calculated as follows. The proportions of coding and non-coding reads were computed as the number of reads in each class divided by the total number of reads in both classes, considering reads that RISER accepted through the pore. Relative enrichment was computed by dividing the proportions by those in the control condition. Relative enrichment was also computed for lncRNAs and mtRNAs using the same approach. Nucleotide enrichment was calculated similarly. The proportions of coding and non-coding nucleotides were computed as the sum of read lengths in each class divided by the sum of read lengths in both classes. This was calculated for reads that RISER accepted through the pore, as well as for all sequenced reads. Relative enrichment was computed by dividing the proportions by those in the control condition. Relative enrichment was also computed for lncRNA and mtRNA using the same approach, considering reads that RISER accepted through the pore.

The analysis of available pores over time was done as follows. During each MinION run, MinKNOW conducted a pore scan every 1.5 hours to assess pore health. The number of available pores found in each pore scan was extracted from the "pore_scan_data_[run_id].csv" file output by MinKNOW by counting the number of "single_pore" entries per scan for the channels in each condition. We then computed the number of available pores as a percentage of pores available in the first pore scan.

Flow cell per condition analysis

For the runs that used a whole flow cell per condition, we repeated the above analyses: (1) read length distributions, (2) transcript coverage per read, (3) enrichment in terms of reads and nucleotides for coding RNA, non-coding RNA, lncRNA and mtRNA. However, we did not need to split reads into conditions based on channel number and instead considered reads from all channels in each flow cell.

Data availability

Nanopore DRS signals for human heart were obtained from the European Nucleotide Archive (ENA) under accession number PRJEB40410[25]. Nanopore DRS signals for HEK293 cells were obtained from the ENA under accession number ENA PRJEB40872[26] and from the NCBI Gene Expression Omnibus (GEO) under accession number TBD[15]. Nanopore DRS signals for HeLa cells were obtained from the NCBI GEO under accession number GSE211762[16]. Nanopore DRS signals generated in this study have been deposited at NCBI GEO under accession number TBD.

Code availability

RISER is freely available from https://github.com/comprna/riser under the GNU General Public License v3.0.

Acknowledgements

We thank Carolina Correa Ospina and the Biomolecular Resource Facility at the John Curtin School of Medical Research, Australian National University, for the provision of a MinION sequencing device for testing and for providing excellent tuition on the operation of the MinKNOW software. For software development and testing, the National Computational Infrastructure (Australia) supercomputing resources were used through ANUMAS and NCMAS access grants. We also thank employees from Oxford Nanopore Technologies for clarification of the ReadUntil API parameters.

Funding

This research was supported by the Australian Research Council (ARC) Discovery Project grants DP210102385 (to EE) and DP220101352 (to NH and EE), by a grant from the Bootes Foundation (to NS), by an Australian Government Research Training Program (RTP) scholarship (to AS), and by the National Health and Medical Research Council (NHMRC) Investigator Grant GNT1175388 (to NS).

Author Contributions

A.S. conceived the idea for the project. E.E. supervised the whole project. A.S. designed and developed RISER, including data preparation, model development and evaluation, and software development and integration with the ReadUntil API. N.H. and N.S. prepared and sequenced the GM24385 cell line for training. A.R. performed the sequencing for the RISER live sequencing experiments. N.S. supervised the RISER live sequencing experiments and advised on their design. A.S., E.E. and N.S. contributed to the interpretation of the sequencing experiment results. A.S. took the lead in writing the manuscript and produced all the figures, with essential inputs from E.E., N.S., N.H. and A.R.

Competing Interests

A.S. has received funds from Oxford Nanopore Technologies for travel and accommodation to speak at the Oxford Nanopore Technologies conference London Calling 2022. The authors declare no other competing interests.

References

1. Workman, R. E. *et al.* Nanopore native RNA sequencing of a human poly(A) transcriptome. *Nat. Methods* **16**, 1297–1305 (2019).
2. Soneson, C. *et al.* A comprehensive examination of Nanopore native RNA sequencing for characterization of complex transcriptomes. *Nat. Commun.* **10**, 3359 (2019).
3. Hardwick, S. A. *et al.* Targeted, High-Resolution RNA Sequencing of Non-coding Genomic Regions Associated With Neuropsychiatric Functions. *Front. Genet.* **10**, 309 (2019).
4. Naarmann-de Vries, I. S., Eschenbach, J. & Dieterich, C. Improved nanopore direct RNA sequencing of cardiac myocyte samples by selective mt-RNA depletion. *J. Mol. Cell. Cardiol.* **163**, 175–186 (2022).
5. Mercer, T. R. *et al.* Targeted sequencing for gene discovery and quantification using RNA CaptureSeq. *Nat. Protoc.* **9**, 989–1009 (2014).
6. Jang, J. S. *et al.* Comparative evaluation for the globin gene depletion methods for mRNA sequencing using the whole blood-derived total RNAs. *BMC Genomics* **21**, 890 (2020).
7. Payne, A. *et al.* Readfish enables targeted nanopore sequencing of gigabase-sized genomes. *Nat. Biotechnol.* **39**, 442–450 (2021).
8. Ulrich, J.-U., Lutfi, A., Rutzen, K. & Renard, B. Y. ReadBouncer: precise and scalable adaptive sampling for nanopore sequencing. *Bioinformatics* **38**, i153–i160 (2022).
9. Kovaka, S., Fan, Y., Ni, B., Timp, W. & Schatz, M. C. Targeted nanopore sequencing by real-time mapping of raw electrical signal with UNCALLED. *Nat. Biotechnol.* **39**, 431–441 (2021).
10. Danilevsky, A., Polsky, A. L. & Shomron, N. Adaptive sequencing using nanopores and deep learning of mitochondrial DNA. *Brief. Bioinform.* **23**, (2022).
11. Bava, F.-A. *et al.* CPEB1 coordinates alternative 3'-UTR formation with translational regulation. *Nature* **495**, 121–5 (2013).
12. Andreassi, C. & Riccio, A. To localize or not to localize: mRNA fate is in 3'UTR ends. *Trends Cell Biol.* **19**, 465–74 (2009).
13. Feng, Y., Zhang, Y., Ying, C., Wang, D. & Du, C. Nanopore-based fourth-generation DNA sequencing technology. *Genomics. Proteomics Bioinformatics* **13**, 4–16 (2015).

14. Sainath, T. N., Mohamed, A., Kingsbury, B. & Ramabhadran, B. Deep convolutional neural networks for LVCSR. in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing* 8614–8618 (IEEE, 2013). doi:10.1109/ICASSP.2013.6639347

15. Sneddon, A., Acera Mateos, P., Shirokikh, N. & Eyras, E. Language-Informed Basecalling Architecture for Nanopore Direct RNA Sequencing [PREPRINT OR COMPARABLE]. *bioRxiv* 2022.10.19.512968 (2022). doi:https://doi.org/10.1101/2022.10.19.512968

16. Mateos, P. A. *et al.* Simultaneous identification of m6A and m5C reveals coordinated RNA modification at single-molecule resolution [PREPRINT OR COMPARABLE]. *bioRxiv* 2022.03.14.484124 (2022). doi:doi.org/10.1101/2022.03.14.484124

17. Teng, H. *et al.* Chiron: translating nanopore raw signal directly into nucleotide sequence using deep learning. *Gigascience* **7**, (2018).

18. LeCun, Y., Kavukcuoglu, K. & Farabet, C. Convolutional networks and applications in vision. in *Proceedings of 2010 IEEE International Symposium on Circuits and Systems* 253–256 (IEEE, 2010). doi:10.1109/ISCAS.2010.5537907

19. He, K., Zhang, X., Ren, S. & Sun, J. Deep Residual Learning for Image Recognition. in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* 770–778 (IEEE, 2016). doi:10.1109/CVPR.2016.90

20. Bai, S., Kolter, J. Z. & Koltun, V. An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling. (2018).

21. Paszke, A. *et al.* PyTorch: An imperative style, high-performance deep learning library. *Adv. Neural Inf. Process. Syst.* **32**, (2019).

22. Bao, Y. *et al.* SquiggleNet: real-time, direct classification of nanopore signals. *Genome Biol.* **22**, 298 (2021).

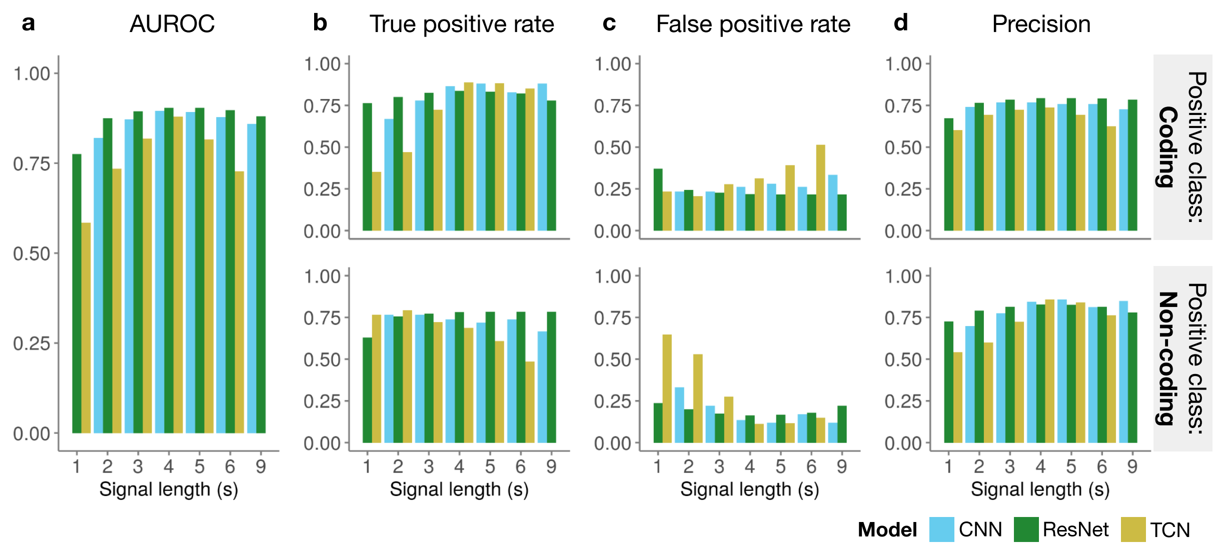
23. Li, H. Minimap2: pairwise alignment for nucleotide sequences. *Bioinformatics* **34**, 3094–3100 (2018).

24. Li, H. *et al.* The Sequence Alignment/Map format and SAMtools. *Bioinformatics* **25**, 2078–9 (2009).

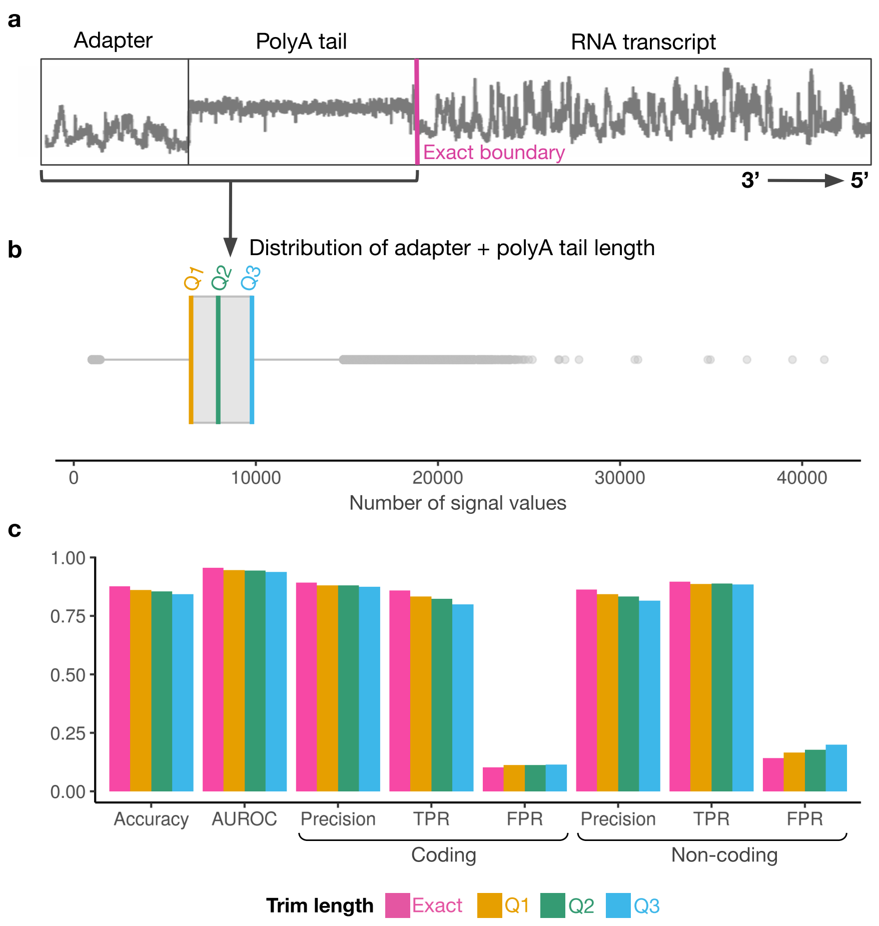
25. de la Rubia, I. *et al.* RATTLE: reference-free reconstruction and quantification of transcriptomes from Nanopore sequencing. *Genome Biol.* **23**, 153 (2022).

26. Pratanwanich, P. N. *et al.* Identification of differential RNA modifications from nanopore direct RNA sequencing with xPore. *Nat. Biotechnol.* **39**, 1394–1402 (2021).

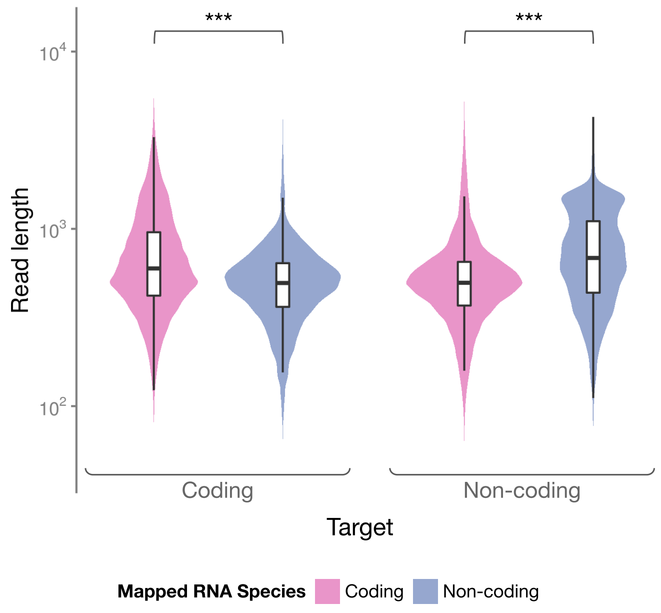
Extended Data



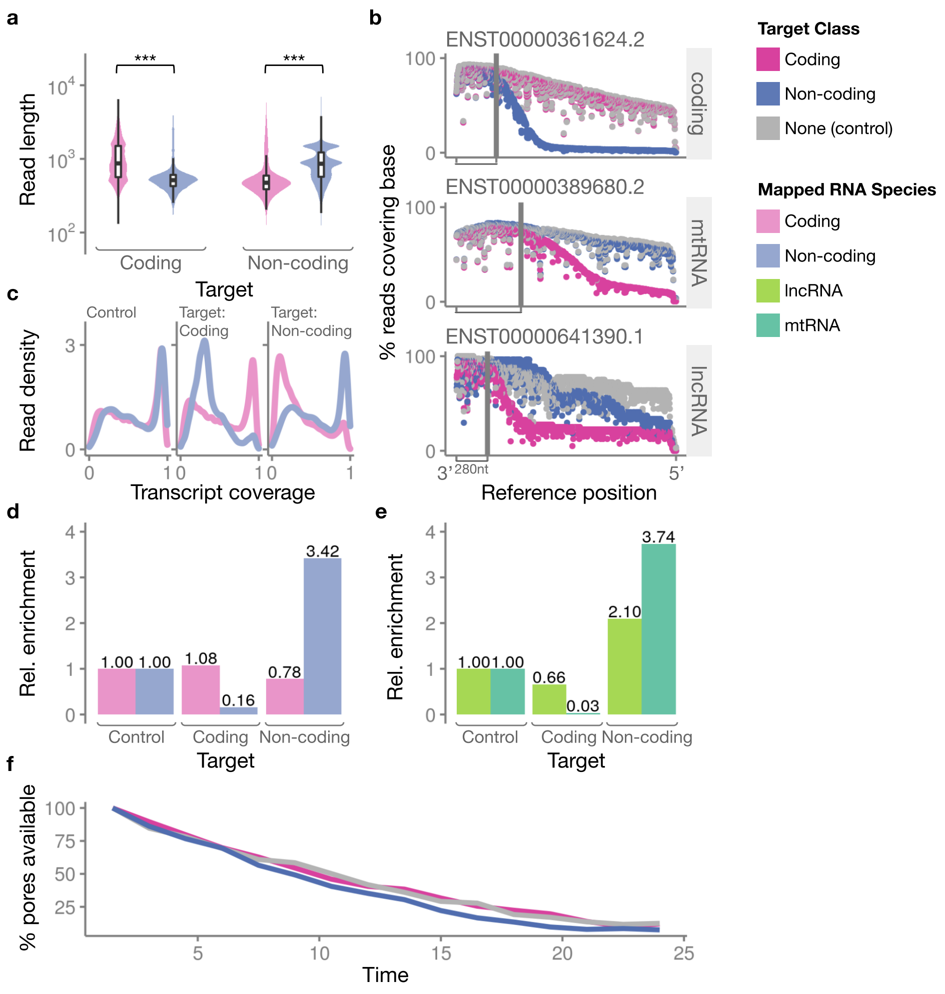
Extended Data Figure 1: Evaluation of candidate RISER models. **a-d,** Model performance on the test set for each candidate input signal length (x-axes) in seconds, color-coded by three convolutional network architectures: ”vanilla” convolutional neural network (CNN) (cyan), residual network (ResNet) (dark green), temporal convolutional network (TCN) (dark yellow). We show the AUROC (**a**), true positive rate (TPR) (**b**), false positive rate (FPR) (**c**) and precision (**d**) for both coding (upper panel) and non-coding (lower panel) targets.



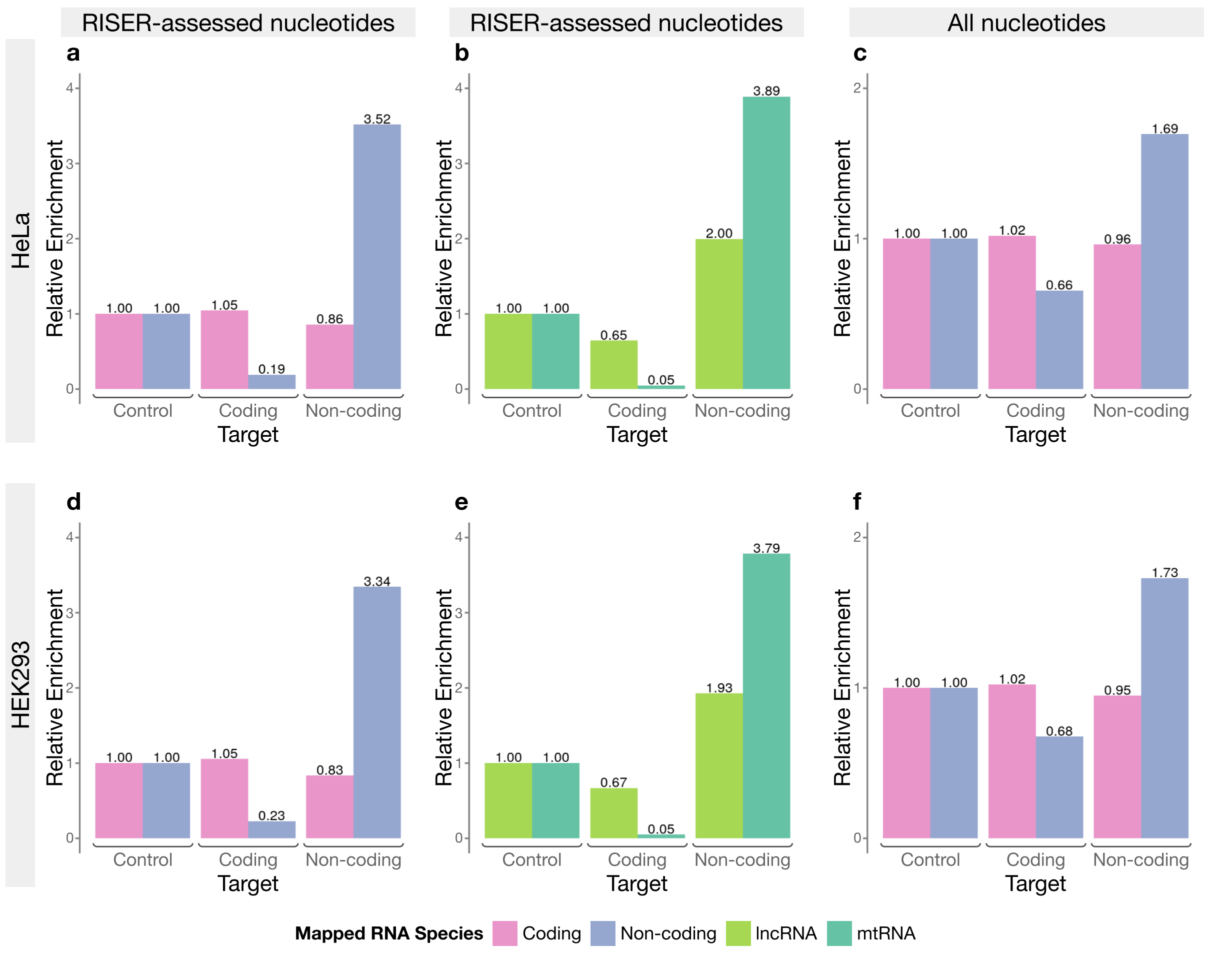
Extended Data Figure 2: Removal of the sequencing adapter and polyA tail from the nanopore signal. **a,** Example DRS signal from the 3' to 5' direction with boundaries between the sequencing adapter and polyA tail (black) and between the polyA tail and RNA transcript (pink) shown. **b,** Distribution of the number of signal values (x-axis) comprising the adapter and polyA tail for the training dataset, with quartiles highlighted. **c,** Model performance on the test set to classify coding or non-coding RNA when signals were trimmed at the exact boundary between the polyA tail and RNA transcript (pink), as well as by each candidate fixed trim length: quartile 1 (Q1) (orange), quartile 2 (Q2) (green) and quartile 3 (Q3) (blue).



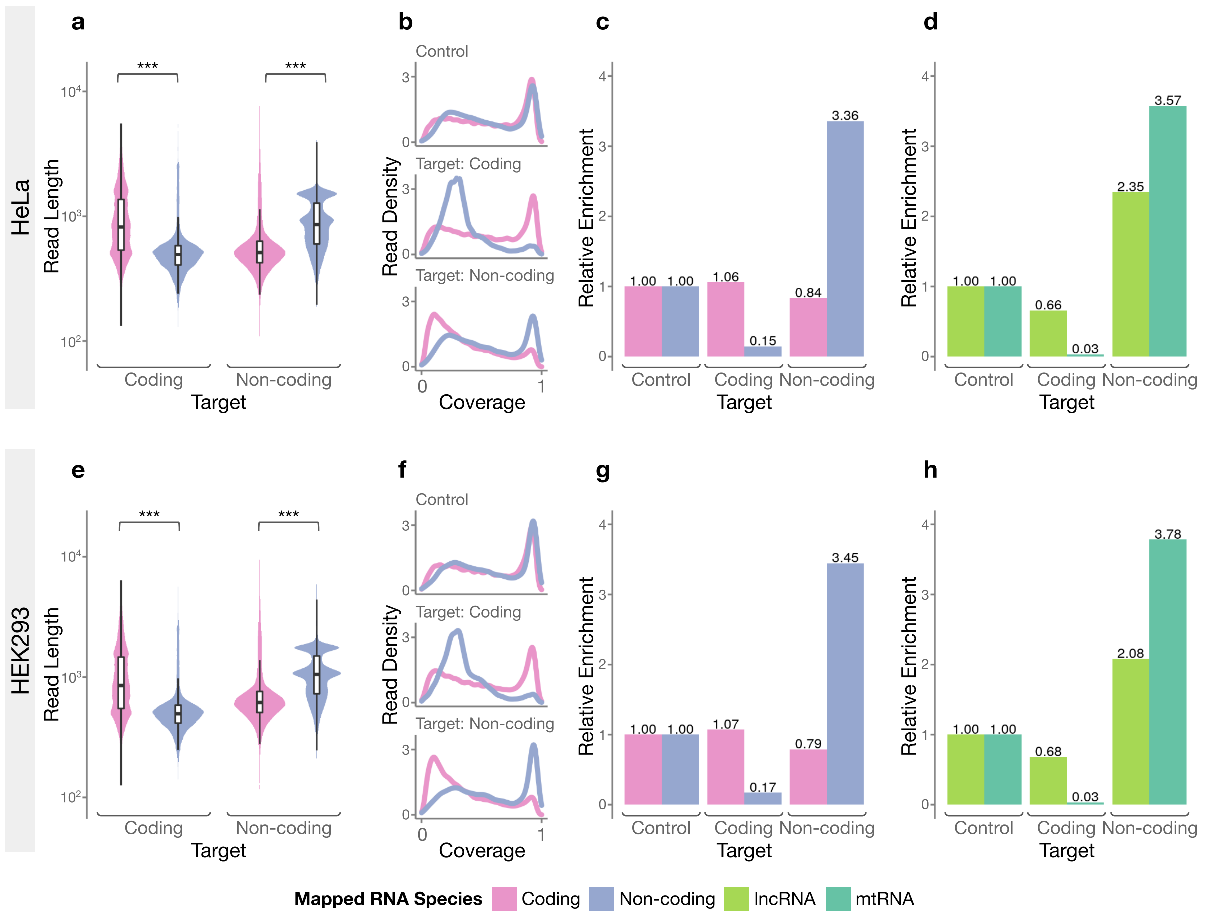
Extended Data Figure 3: RISER reduces the length of off-target reads in a MinKNOW playback run using. The distribution of read lengths for the mapped RNA species when the RISER target class was coding (left panel) and non-coding (right panel) in a playback of an REH cancer cell line sample. The mapped RNA species is color-coded (light pink: coding, light blue: non-coding). For both targets, the read lengths of the mapped RNA species were compared using a Wilcoxon rank sum test with continuity correction (H_1: on-target > off-target) (p-value < 2.22E-16 for both target classes). Outliers were not included.



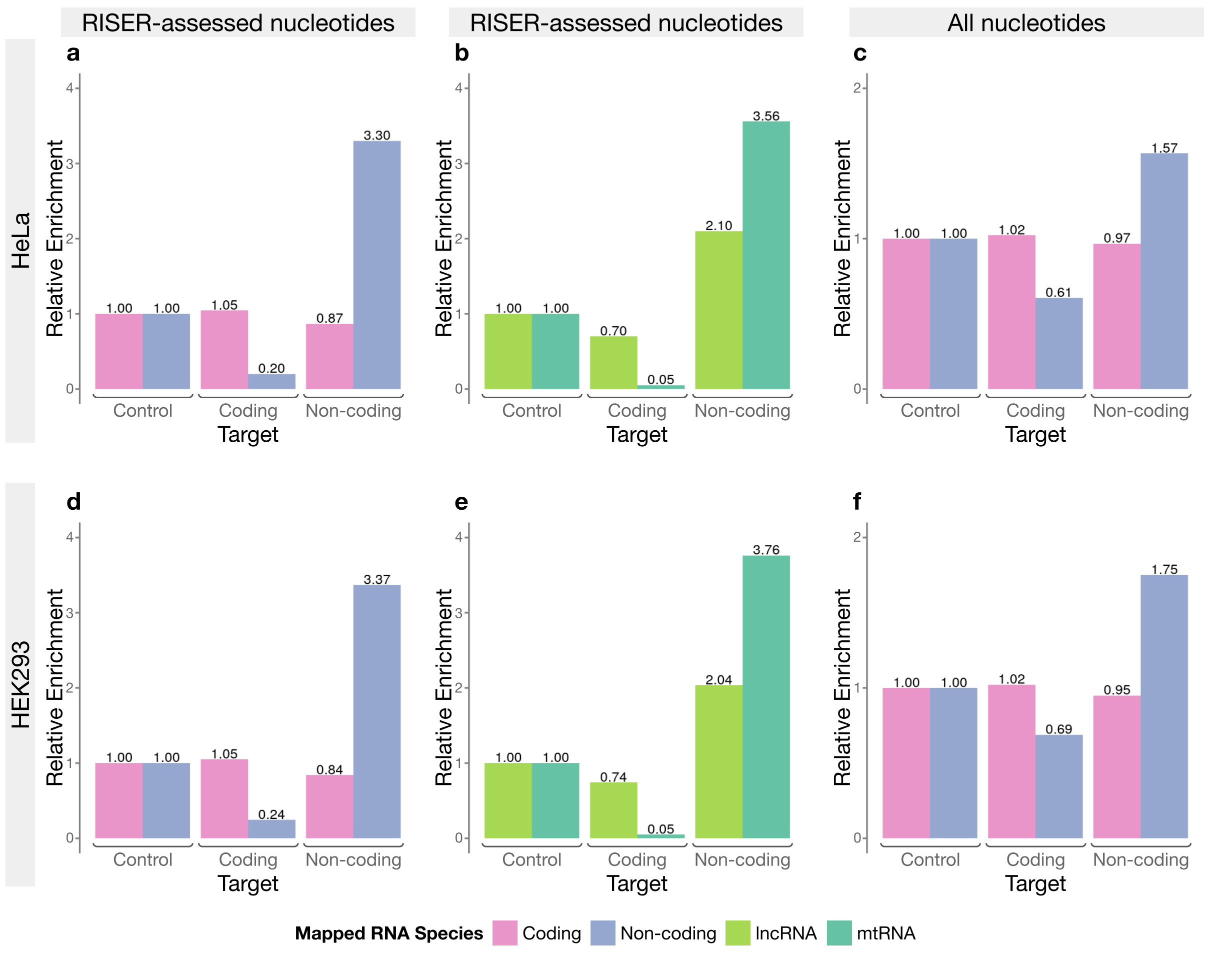
Extended Data Figure 4: RISER enables real-time enrichment of RNA species in HEK293 cells. a-f, RISER performance during real-time sequencing of poly(A)$^+$ RNA from HEK293 cells, using a MinION flow cell split into three RISER target classes: coding, non-coding, and none (i.e., control). The RISER target class is color-coded using a dark color scheme (dark pink: coding, dark blue: non-coding, grey: control), while the mapped RNA species is color-coded using a light color scheme (light pink: coding, light blue: non-coding). **a,** The distribution of read lengths for the mapped RNA species when the RISER target class was coding (left panel) and non-coding (right panel). For both targets, the read lengths of the mapped RNA species were compared using a Wilcoxon rank sum test with continuity correction (H_1: on-target > off-target) (p-value < 2.22E-16 for both target classes). **b,** Sequencing coverage per base for an example protein-coding RNA (upper panel), mtRNA (middle panel) and lncRNA (lower panel). The reference positions (x-axes) are ordered 3' to 5'. We indicate the position 280nt upstream of the 3' end, which approximately corresponds to the RISER input length. **c,** The distribution of the proportion of transcript covered by each sequenced read. **d,** The enrichment of each mapped RNA species, calculated as the proportion of reads that RISER accepted through the pore for each mapped RNA species, relative to the proportions in the control where RISER was not used. **e,** The enrichment of lncRNA and mtRNA, calculated using the same approach as (**d**). **f**, Available pores as a percentage of the initial number of available pores, as recorded during the 1.5 hourly pore scans across the 24h duration of the sequencing run.



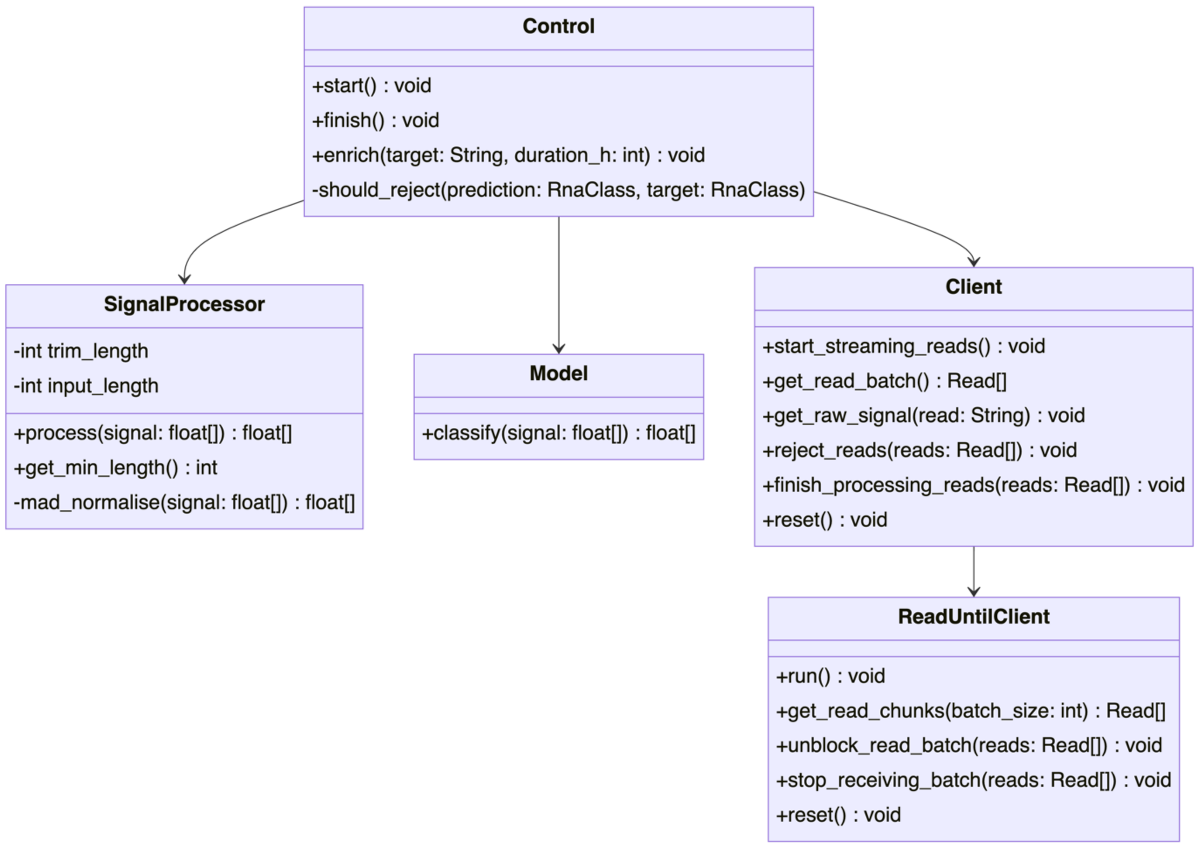
Extended Data Figure 5: RISER enriches the nucleotide sequencing depth of the target RNA species. Relative nucleotide enrichment (y-axes) achieved by RISER during real-time sequencing of poly(A)$^+$ RNA from HeLa cells (**a-c**) and HEK293 cells (**d-f**), using a MinION flow cell split into three RISER target classes: coding, non-coding, and none (i.e., control). The mapped RNA species is color-coded (light pink: coding, light blue: non-coding, lime green: lncRNA, dark green: mtRNA. **a,d,** The enrichment of each mapped RNA species, calculated as the proportion of nucleotides that RISER accepted through the pore for each mapped RNA species, relative to the proportions in the control where RISER was not used. **b,e,** The enrichment in terms of nucleotides of lncRNA and mtRNA, calculated using the same approach as (**a,d**). **c,f,** The enrichment of each mapped RNA species, calculated as the proportion of all sequenced nucleotides for each mapped RNA species, relative to the proportions in the control where RISER was not used.



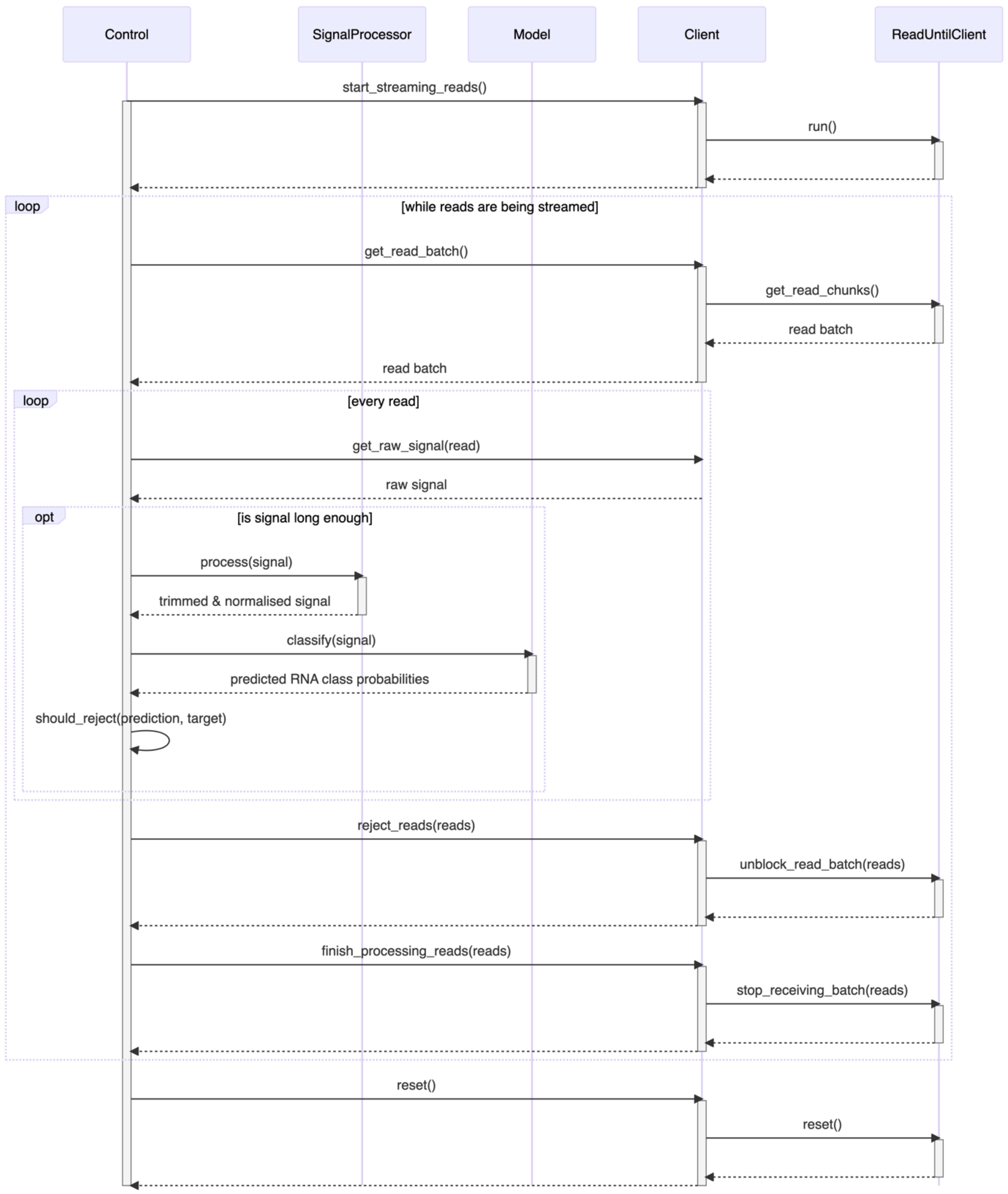
Extended Data Figure 6: Enrichment and depletion of RNA species using a whole flow cell per condition. RISER performance during real-time sequencing of poly(A)$^+$ RNA from HeLa cells (**a-d**) and HEK293 cells (**e-h**), using a MinION flow cell per condition: coding as the RISER target class, non-coding as the RISER target class, and no RISER as a control. The mapped RNA species is color-coded (light pink: coding, light blue: non-coding, lime green: lncRNA, dark green: mtRNA. **a,e,** The distribution of read lengths for the mapped RNA species when the RISER target class was coding (left panel) and non-coding (right panel). For both targets, the read lengths of the mapped RNA species were compared using a Wilcoxon rank sum test with continuity correction (H_1: on-target > off-target) (p-value < 2.22E-16 for both target classes). **b,f,** The distribution of the proportion of transcript covered by each sequenced read. **c,g,** The enrichment of each mapped RNA species, calculated as the proportion of reads that RISER accepted through the pore for each mapped RNA species, relative to the proportions in the control where RISER was not used. **d,h,** The enrichment of lncRNA and mtRNA, calculated using the same approach as (**c,g**).



Extended Data Figure 7: RISER enriches the nucleotide sequencing depth of the target RNA species when tested using a whole flow cell per condition. RISER performance during real-time sequencing of poly(A)$^+$ RNA from HeLa cells (**a-c**) and HEK293 cells (**d-f**), using a MinION flow cell per condition: coding as the RISER target class, non-coding as the RISER target class, and no RISER as a control. The mapped RNA species is color-coded (light pink: coding, light blue: non-coding, lime green: lncRNA, dark green: mtRNA. **a,d,** The enrichment of each mapped RNA species, calculated as the proportion of nucleotides that RISER accepted through the pore for each mapped RNA species, relative to the proportions in the control where RISER was not used. **b,e,** The enrichment in terms of nucleotides of lncRNA and mtRNA, calculated using the same approach as (**a,d**). **c,f,** The enrichment of each mapped RNA species, calculated as the proportion of all sequenced nucleotides for each mapped RNA species, relative to the proportions in the control where RISER was not used.


Control
+start() : void
+finish() : void
+enrich(target: String, duration_h: int) : void
-should_reject(prediction: RnaClass, target: RnaClass)
SignalProcessor
-int trim_length
-int input_length
+process(signal: float[]) : float[]
+get_min_length() : int
-mad_normalise(signal: float[]) : float[]
Model
+classify(signal: float[]) : float[]
Client
+start_streaming_reads() : void
+get_read_batch() : Read[]
+get_raw_signal(read: String) : void
+reject_reads(reads: Read[]) : void
+finish_processing_reads(reads: Read[]) : void
+reset() : void
ReadUntilClient
+run() : void
+get_read_chunks(batch_size: int) : Read[]
+unblock_read_batch(reads: Read[]) : void
+stop_receiving_batch(reads: Read[]) : void
+reset() : void

Extended Data Figure 8: Class diagram for the RISER software using unified modelling language (UML) notation. Each box represents a class and contains three sections: name, attributes, and method definitions. Attributes and method definitions have assigned access privileges, with ”+” and ”-" indicating public and private visibility, respectively. Arrows denote associations between classes, i.e., the class at the arrow's tail has an instance of the class at the tip of the arrow. For simplicity, logging, csv writing functionality and optional method parameters have not been shown. Only methods in the ONT ReadUntilClient (v3.0.0) that have been utilized by RISER are shown.



Extended Data Figure 9: Sequence diagram for the RISER software using unified modelling language (UML) notation. The diagram illustrates the timeline (top-to-bottom) of interactions between RISER components during a targeted sequencing run. Each blue box represents an object (class instance), with vertical blue lifelines extending beneath them. The white vertical bars denote when an object is busy (either executing a task or awaiting a message). Arrows denote synchronous messages (the sender requires a reply before continuing), with solid arrows representing messages from the sender to the receiver and dotted arrows denoting the return message from receiver to sender. Conditional functionality is enclosed within an "opt" rectangle, with the condition listed in square brackets. Loops are also enclosed within a "loop" rectangle, with the loop condition listed in square brackets.

Supplementary Information

Supplementary Table 1: MinION DRS datasets used in this study for model development and evaluation. The HEK293-B dataset was composed by taking the second and third biological replicates for both the METTL3 knockout and wild-type samples in ENA PRJEB40872 and combining all technical replicates. Only data from HeLa WT cells was used from GSE211762.

Dataset ID	Cell line / tissue type	Accession #	Sequencing kit	Flow cell	Pore	Used for training?
Heart	Human heart	ENA PRJEB40410[25]	SQK-RNA002	FLO-MIN106	R9.4.1	Yes
HEK293-A	HEK293	TBD [15]	SQK-RNA002	FLO-MIN106	R9.4.1	Yes
HEK293-B	HEK293	ENA PRJEB40872[26]	SQK-RNA002	FLO-MIN106	R9.4.1	Yes
GM24385	GM24385	TBD	SQK-RNA002 (modified)	FLO-MIN106	R9.4.1	Yes
HeLa	HeLa	GSE211762[16]	SQK-RNA002	FLO-MIN106	R9.4.1	No

Supplementary Table 2: Hyperparameter optimization results for ResNet. "O" and "A" denote the "bottleneck" and "basic" residual block types, respectively, defined in [19].

Input signal length	Batch size	Learning rate	Block type	Num. layers (l)	Kernel size (k)	Blocks (b)	Channels (c)	Num. trainable params	Val. accuracy (%)
4s	32	0.001	O	4	13	2,2,2,2	20,30,45,67	75,374	76.44
4s	1000	0.001	O	4	13	2,2,2,2	20,30,45,67	75,374l	75.26
4s	32	0.001	O	4	19	2,2,2,2	16,24,36,54	49,086	76.23
4s	16	0.001	O	4	19	2,2,2,2	20,30,45,67	75,494	76.67
4s	32	0.0001	O	4	19	2,2,2,2	20,30,45,67	75,494	75.41
4s	32	0.001	O	4	19	2,2,2,2	20,30,45,67	75,494	76.59
4s	32	0.001	O	4	19	2,2,2,2	20,30,45,67	24,990	74.68
4s	1000	0.0001	O	4	19	2,2,2,2	20,30,45,67	75,494	73.18
4s	1000	0.001	O	4	19	2,2,2,2	20,30,45,67	75,494	75.41
4s	32	0.001	O	4	19	2,2,2,2	20,40,80,160	278,602	76.8
4s	32	0.001	O	4	19	2,2,2,2	32,48,72,108	191,562	76.75
4s	32	0.001	O	4	19	2,2,2,2	32,64,128,256	708,418	77.08
4s	1000	0.001	O	4	19	2,2,2,2	32,64,128,256	708,418	76.07
4s	32	0.001	O	4	19	2,3,4,2	20,30,45,67	106,154	77.08
4s	1000	0.001	O	4	19	2,3,4,2	20,30,45,67	106,154	75.92
4s	32	0.001	O	4	19	3,4,23,3	20,30,45,67	379,514	78.75
4s	32	0.001	O	4	19	3,4,6,3	128, 256, 512, 1024	7,670,402	77.73
4s	32	0.001	O	4	19	3,4,6,3	20,30,45,67	166,844	78.1

4s	32	0.001	O	4	19	3,4,6,3	256, 512, 1024, 2048	30,596,354	77.43
4s	32	0.001	O	4	19	3,4,6,3	32,64,128,256	487,394	77.94
4s	32	0.001	O	4	19	3,4,6,3	64, 128, 256, 512	1,928,258	78.16
4s	32	0.001	O	4	25	2,2,2,2	20,30,45,67	75,614	76.56
4s	1000	0.001	O	4	25	2,2,2,2	20,30,45,67	75,614	75.35
4s	32	0.001	O	5	19	2,2,2,2,2	20,30,45,67,100	168,384	78.09
4s	1000	0.001	O	5	19	2,2,2,2,2	20,30,45,67,100	168,384	76.8
4s	32	0.001	O	6	19	2,2,2,2,2,2	20,30,45,67,100,150	375,684	79.32
4s	32	0.001	O	7	19	2,2,2,2,2,2,2	20,30,45,67,100,150,225	840,384	80.16
4s	32	0.001	A	4	19	2,2,2,2	20,30,45,67	102,120	78.23
4s	1000	0.001	A	4	19	2,2,2,2	20,30,45,67	102,120	77.14
4s	32	0.001	A	4	19	3,4,6,3	32,64,128,256	2,041,090	79.76
4s	32	0.001	A	4	19	3,4,6,3	64, 128, 256, 512	8,139,266	79.36
4s	32	0.0001	A	10	19	1,1,1,1,1,1,1,1,1,1	20,30,45,67,100,150,225,337,505,757	5,869,558	80.17
4s	**32**	**0.001**	**A**	**10**	**19**	**1,1,1,1,1,1,1,1,1,1**	**20,30,45,67,100,150,225,337,505,757**	**5,869,558**	**81.12**

Supplementary Table 3. Hyperparameter optimization results for the Temporal Convolutional Network (TCN).

Input signal length	Batch size	Learning Rate	Num. layers (l)	Num. channels (c)	Kernel size (k)	Dilation base (d)	Dropout (r)	# Trainable Params	% Val Accuracy
4s	32	0.0001	5	128	7	6	0.2	1,102,210	77.7
4s	32	0.0001	5	128	7	8	0.2	1,102,210	76.07

4s	32	0.0001	6	128	7	4	0.2	1,348,610	77.54
4s	32	0.0001	7	128	7	3	0.2	1,595,010	76.97
4s	32	0.0001	9	16	15	2	0.2	68,338	76.48
4s	32	0.0001	10	16	7	2	0	37,314	76.35
4s	32	0.0001	10	16	7	2	0.05	37,314	76.56
4s	32	0.0001	10	16	7	2	0.1	37,314	76.29
4s	16	0.0001	10	16	7	2	0.2	37,314	74.89
4s	32	0.00001	10	16	7	2	0.2	37,314	72.46
4s	32	0.0001	10	16	7	2	0.2	37,314	75.17
4s	32	0.001	10	16	7	2	0.2	37,314	72.89
4s	32	0.0001	10	16	9	2	0.2	47,074	76.03
4s	32	0.0001	10	16	11	2	0.2	56,834	76.72
4s	32	0.0001	10	32	7	2	0.2	147,330	76.49
4s	**32**	**0.0001**	**10**	**32**	**11**	**2**	**0.05**	**225,282**	**78.9**
4s	32	0.0001	10	64	7	2	0.2	585,474	77.37
4s	32	0.0001	10	64	11	2	0.05	897,026	78.59
4s	16	0.0001	10	128	7	2	0.2	2,334,210	76.7
4s	32	0.0001	10	128	7	2	0.2	2,334,210	75.76
4s	32	0.0001	11	16	5	2	0.2	30,450	73.68
4s	32	0.0001	12	16	3	2	0.2	21,538	71.23
4s	32	0.0001	12	128	3	2	0.2	1,319,170	75.59

Supplementary Table 4: Hyperparameter optimization results for the “vanilla” Convolutional Neural Network (CNN). The classifier “fc” denotes a 2-layer fully connected network with ReLu activation, “gap” denotes a global average pooling layer and “gap_fc” denotes a global average pooling layer followed by a single fully connected layer.

Input signal length	Batch size	Learning rate	Num. layers (l)	Blocks per layer (b)	Channels (c)	Kernel size (k)	Classifier (f)	# Trainable params	% Val accuracy
4s	32	0.001	4	1	20,30,45,67	3	fc	206,674,703	69.74
4s	32	0.001	4	1	20,30,45,67	3	gap	15,253	70.08
4s	32	0.001	4	1	20,30,45,67	3	gap_fc	15,523	70.15
4s	32	0.001	4	1	20,30,45,67	5	gap_fc	25,223	72
4s	32	0.001	4	1	20,30,45,67	7	gap_fc	35,193	72.08
4s	32	0.001	4	1	20,30,45,67	11	gap_fc	55,133	73.45
4s	32	0.001	4	1	32,64,128,256	3	gap_fc	130,114	70.79
4s	32	0.001	4	2	20,30,45,67	3	gap_fc	38,857	72.05
4s	32	0.001	4	4	20,30,45,67	3	gap_fc	86,065	50
4s	32	0.0001	4	4	20,30,45,67	3	gap_fc	86,065	73.9
4s	32	0.001	5	1	20,30,45,67,100	3	gap_fc	35,519	72.21
4s	32	0.001	6	1	20,30,45,67,100,150	3	gap_fc	80,769	74.12
4s	32	0.001	7	1	20,30,45,67,100,150,225	3	gap_fc	182,394	50
4s	32	0.0001	7	1	20,30,45,67,100,150,225	3	gap_fc	182,394	76.51
4s	32	0.001	8	1	20,30,45,67,100,150,225,337	3	gap_fc	410,430	50
4s	32	0.0001	8	1	20,30,45,67,100,150,225,337	3	gap_fc	2,069,942	79.38
4s	32	0.0001	8	1	20,40,80,160,320,640,1280,2560	3	gap_fc	13,116,682	78.99
4s	32	0.0001	8	1	32,64,128,256,512,1024,2048,4096	3	gap_fc	33,568,834	79.29

4s	32	0.001	10	1	20,20,30,30,45,45,67,67,100,100	3	gap_fc	89,223	50
4s	32	0.001	10	1	20,30,45,67,100,150,225,337,505,757	3	gap_fc	2,069,942	50
4s	32	0.0001	10	1	20,30,45,67,100,150,225,337,505,757	3	gap_fc	2,069,942	79.35
4s	32	0.0001	10	2	20,30,45,67,100,150,225,337,505,757	3	gap_fc	5,169,924	50
4s	32	0.00001	10	2	20,30,45,67,100,150,225,337,505,757	3	gap_fc	5,169,924	50
4s	32	0.000001	10	2	20,30,45,67,100,150,225,337,505,757	3	gap_fc	5,169,924	50
4s	32	0.0000001	10	2	20,30,45,67,100,150,225,337,505,757	3	gap_fc	5,169,924	50
4s	**32**	**0.0001**	**12**	**1**	**20,30,45,67,100,150,225,337,505,757,1135,1702**	**3**	**gap_fc**	**10,447,564**	**80.22**

Supplementary Note 1: RISER command structure.

```
usage: riser.py [-h] -t  -d  [-c] [-m] [-p] [-s]

optional arguments:
 -h, --help          show this help message and exit
 -t , --target       RNA species to enrich for. This must be either
                     {coding, noncoding}. (required)
 -d , --duration     Length of time (in hours) to run RISER for. This
                     should be the same as the MinKNOW run length.
                     (required)
 -c , --config       Config file for model hyperparameters. (default:
                     model/cnn_best_model.yaml)
 -m , --model        File containing saved model weights. (default:
                     model/cnn_best_model.pth)
 -p , --polya        Number of values to remove from the start of the raw
                     signal to exclude the polyA tail and sequencing
                     adapter signal from analysis. (default: 6481)
 -s , --secs         Number of seconds of transcript signal to use for
                     decision. (default: 4)
```

Supplementary Note 2: RISER output files - Console output.

```
Using cuda device
Usage: riser.py -t noncoding -d 48
All settings used (including those set by default):
--target         : Species.NONCODING
--duration_h     : 48
--config_file    : models/cnn_best_model.yaml
--model_file     : models/cnn_best_model.pth
--polyA_length   : 6481
--secs           : 4
Client is running.
Batch of 110 reads received: 59 long enough to assess, 46 of which were
rejected (took 0.3148s)
Batch of  93 reads received: 29 long enough to assess, 21 of which were
rejected (took 0.1376s)
Batch of 107 reads received: 32 long enough to assess, 24 of which were
rejected (took 0.1568s)
...
```

Supplementary Note 3: RISER output files - Log file.

```
2022-08-23T11:56:29 [RISER] INFO: Using cuda device
2022-08-23T11:56:31 [RISER] INFO: Usage: riser.py --target noncoding --
duration 24
2022-08-23T11:56:31 [RISER] INFO: All settings used (including those set
by default):
2022-08-23T11:56:31 [RISER] INFO: --target         : Species.NONCODING
2022-08-23T11:56:31 [RISER] INFO: --duration_h     : 24
2022-08-23T11:56:31 [RISER] INFO: --config_file    :
defaults/cnn_best_model.yaml
2022-08-23T11:56:31 [RISER] INFO: --model_file     :
defaults/cnn_best_model.pth
2022-08-23T11:56:31 [RISER] INFO: --polyA_length   : 6481
2022-08-23T11:56:31 [RISER] INFO: --secs           : 4
```

```
2022-08-23T11:56:31 [RISER] INFO: Client is running.
2022-08-23T11:56:32 [RISER] INFO: Batch of   0 reads received:  0 long
enough to assess,  0 of which were rejected (took 0.0000s)
2022-08-23T11:56:33 [RISER] INFO: Batch of 268 reads received:  0 long
enough to assess,  0 of which were rejected (took 0.0013s)
2022-08-23T11:56:34 [RISER] INFO: Batch of 284 reads received:  0 long
enough to assess,  0 of which were rejected (took 0.0014s)
2022-08-23T11:56:35 [RISER] INFO: Batch of 262 reads received:  0 long
enough to assess,  0 of which were rejected (took 0.0015s)
...
```

Supplementary Table 5: RISER output files - CSV file.

read_id	channel	probability_noncoding	probability_coding	prediction	target	decision
075391a9-2816-45b0-aebb-12b1f398fcd3	204	0.83	0.17	NONCODING	CODING	REJECT
afcbd456-1843-4322-85d2-7f001ef0dc01	176	0.24	0.76	CODING	CODING	ACCEPT
d8de6be4-4a01-42dc-b8ab-77abc8f818e1	373	0.36	0.64	CODING	CODING	ACCEPT
02cdeae6-3d5d-4615-bc61-7d5dd9a7217c	91	0.79	0.21	NONCODING	CODING	REJECT
4460c783-4663-4666-be4d-c52590fdff31	293	0.26	0.74	CODING	CODING	ACCEPT
…	…	…	…	…	…	…