Multiplexed Salivary miRNA Quantification for Predicting Severe COVID-19 Symptoms in Children Using Ligation-RPA Amplification Assay

Md. Ahasan Ahamed[1,2,#], Zhikun Zhang[2,#], Aneesh Kshirsagar[2], Anthony J. Politza[3], Usha Sethuraman[4], Srinivasan Suresh[5], Steven Hicks[6], Feng Guo[1], and Weihua Guan[1]*

[1]Department of Intelligent Systems Engineering, Luddy School of Informatics, Computing, and Engineering, Indiana University, Bloomington, Indiana, 47408, USA

[2]Department of Electrical Engineering, Pennsylvania State University, University Park, PA, 16802, United States

[3]Department of Biomedical Engineering, Pennsylvania State University, University Park, PA, 16802, United States

[4]Division of Emergency Medicine, Department of Pediatrics, Children's Hospital of Michigan/Central Michigan University, Detroit, MI 48201, United States

[5]Divisions of Health Informatics and Emergency Medicine, Department of Pediatrics, University of Pittsburgh School of Medicine and UPMC Children's Hospital of Pittsburgh, Pittsburgh, PA 15224, United States

[6]Department of Pediatrics, Division of Academic General Pediatrics, Penn State Neuroscience Institute, Pennsylvania State University, Hershey, PA, 17033, United States

[#] These authors contributed equally to this work.

* Corresponding author: email address: guanw@iu.edu

Table of contents **Page**

Section 1. Methods and Methods---------- 2
Section 2. Supplementary Figures and Legends---------- 6-10
Figure S1---------- 6
Figure S2---------- 7
Figure S3---------- 8
Figure S4---------- 9
Figure S5---------- 10
Section 3. Supplementary Tables---------- 11-15
Table S1-3---------- 11
Table S4---------- 12
Table S5---------- 13
Table S6---------- 14
Table S7---------- 15
Section 4. Supplementary Codes---------- 16-63
Section 5. References ---------- 63

Section 1. Methods and Methods

Materials and Chemicals

miRNA sequences, primers, and probes were synthesized, and nuclease-free water was purchased from Integrated DNA Technologies (IDT). Clinical samples were supplied from Dr. Hicks's Lab and collected at Central Michigan University and the University of Pittsburgh Medical Center (UPMC). DNA purification kits (#28104) and miRNA extraction kits (#217004) were purchased from Qiagen. The RPA amplification kit, TwistAmp® Liquid Basic (#TALQBAS01), was purchased from TwistDx. Gel Loading Dye, Purple (6X) (#B7025), SplintR ligase (# M0375L), 10x T4 DNA Ligase Reaction Buffer (#B0202S), and extreme thermostable single-stranded DNA binding protein (ET SSB, # M2401S) were obtained from New England Biolabs Inc (NEB). SYBR™ Safe DNA Gel Stain (#S33102), Ultra Low Range DNA Ladder (#10597012), dNTP mix (10 mM, #R0192), 200 µL sterile pipette tips (#02707430) and SYTO™ 9 Green Fluorescent Nucleic Acid Stain (100mL, # S34854) were bought from Thermo Fisher Scientific (Thermo Scientific, USA). Chloroform (#366927) was purchased from Millipore Sigma.

Saliva Collection

This prospective cohort study involved a convenience sample of children under 18 years of age evaluated in the emergency departments of two children's hospitals between March 2021 and February 2022. Children were excluded if they were pregnant, had recent dental infections, head or neck trauma, active seizures, presented for psychiatric evaluation, or were accompanied by a non-English or non-Spanish speaking guardian. Saliva samples were collected using a standardized procedure after an oral rinse with tap water. A highly absorbent swab was placed in the sublingual and parotid regions for 10-20 seconds. The samples were mixed with a nucleic acid stabilization reagent from Genotek kits (Catalog #: ORE-100, Kanata, ON, Canada) to deactivate the virus and preserve RNA integrity. The research team, trained through manufacturer-provided video instructions, ensured correct sample acquisition. Samples were stored at room temperature and shipped monthly to the Core facility for processing. Quality control measures were strictly followed, with samples failing to meet criteria being re-analyzed or excluded from further analysis

Ligation-RPA-based miRNA Amplification

Design. Our designed target, probes, and primers were aligned precisely for target miRNA 1273, 296, and 29 and confirmed the specificity of the sequence by BLAST search. The PrimerQuest Tool of IDT and the Python-based package PrimedRPA were used to design primers and probes.[1] All sequences are summarized in **Table S1-3**.

Hybridization and Annealing. The Hybridization reaction of the annealing mixture was a total of 8 mL volume, including the final concentration of 100 nM of 4 mL miRNA, 2 mL of probe P1, and 2 mL of probe P2 was heated to 85 ^{o}C for 2-3 min and immediately placed on ice for annealing. The detailed recipe is discussed in **Table S4**.

Ligation. The ligation reaction, with a total volume of 2 mL, including 0.2 mL 10x ligase buffer, 0.1 mL SpintR ligase enzyme, 0.13 mL ET SSB enzyme, and 1.57 mL water. A ligation reaction was done in 15 min at 37 ˚C. Finally, 2 mL of ligase products were mixed with 8 mL of annealed samples. The detailed recipe is discussed in **Table S4**.

RPA assay. The total volume of RPA reaction is 25 mL, where 5 mL ligation products were added to the 20 mL RPA liquid master mix. The RPA mixture contained 0.425 mL of 1.5 mM of both primers, 12.5 mL of 2×reaction buffer, 1.15 mL of 10 mM dNTPs, 2.5 mL of 10×Basic E-mix, 1.25 mL of 20×core reaction, 0.5 mL of 100-500 mM SYTO 9, and lastly, 1.25 mL of 280 mM

MgOAc. The mixture with miRNA was then performed on the TCAN plate reader or portable analyzer at 37 °C for 20 min, with fluorescence acquired every 30 s. Supplementary **Figures S2D–H** detail the optimization of probe and primer concentrations for the ligation-RPA assay. The detailed recipe is discussed in **Table S4,** and the reaction cost for a single run is provided in **Table S5**.

Gel analysis. The nucleic acids were extracted from the ligation products and amplicons from RPA and detected by the 5% agarose gel electrophoresis performed at 5 V/cm using SYBR™ Safe DNA Gel Stain. The required bands are then analyzed using the BIO-RAD GelDoc Go Imaging System with a 20-second exposure.

Instrumentation

Portable analyzer. The analyzer was developed using PTC Creo software and 3D fabricated with a MakerBot Method X printer. The printed circuit boards (PCBs) were designed using Autodesk Eagle and produced by OSH Park. We assembled the electronic components and the Microcontroller Unit (MCU) onto the PCB using soldering. The thermal module comprises a resistive-heating element (PWR263S-20-2R00J, Digi-Key) affixed to the underside of the aluminum heating block using a thermal compound (AATA-5G, Arctic Alumina). A thermistor (95C0606, Digi-Key) is used in the core of the heating block for the feedback system. An N-channel power MOSFET (63J7707, Digi-Key) controls the temperature stability using a negative feedback mechanism. The optical module was designed using an adjustable Res Cermet Trimmer (3296W, Digi-Key) for LED intensity control, a broad-spectrum IC Color Sensor (AS7341, Digi-Key) for optical detection, and a blue SMD LED (1497-1138-1, Digi-Key) for fluorescence excitation. A voltage regulator is used for Power stabilization for the optical components, including several capacitors. An I2C multiplexer expanded the sensor's capacity. The smartphone GUI, created with MIT App Inventor, enables control and communication with the device via Bluetooth. The cost of Materials for the analyzer is provided in **Table S6**.

Portable Centrifuge. The centrifuge was designed using PTC Creo and 3D fabricated using a MakerBot Method X printer.[2] Soldering and jumper wires were used to connect the internal components of the device. The DC motor was purchased from Amazon (Autotoolhome 12V), the microcontroller used was an Arduino Nano (Arduino.cc), and a 28.8 Wh Li-ion battery was purchased from Digikey (Daytona Industries). All other electronic components, resistors, MOSFETs, switches, LEDs, and buttons, were purchased from Digikey.

Clinical Sample Extraction

Portable extractions were achieved using the miRNeasy Mini Kit from Qiagen. Following their quick-start protocol, extractions were performed with 350 µL sample, 700 µl QIAzol Lysis Reagent, 900 µL 100% Ethanol, 700 µl Buffer RWT, 1000 µl Buffer RPE, 500 µL 95% Ethanol, and 40 µL RNase-free water. During all extraction stages, the portable device runs at max speed (6000 rpm/ 1743 rcf.).[2] Details are shown in Supplementary **Video S1**.

Data analysis method and statistics

All data processing and figure generation were completed using Python and Origin 2021. The threshold was analyzed using a triplicate of NTC samples with $\mu + 3\sigma$. Positive samples are classified using Time to Positive (Tt) when RFU reaches a threshold of $\mu + 3\sigma$. Severe and non-severe concentration was calculated with T-test statistics results using where ($p<0.05$). Correlation and linearity were computed using SciPy and least squares regression.

Section 2: Supplementary Figures and Legends

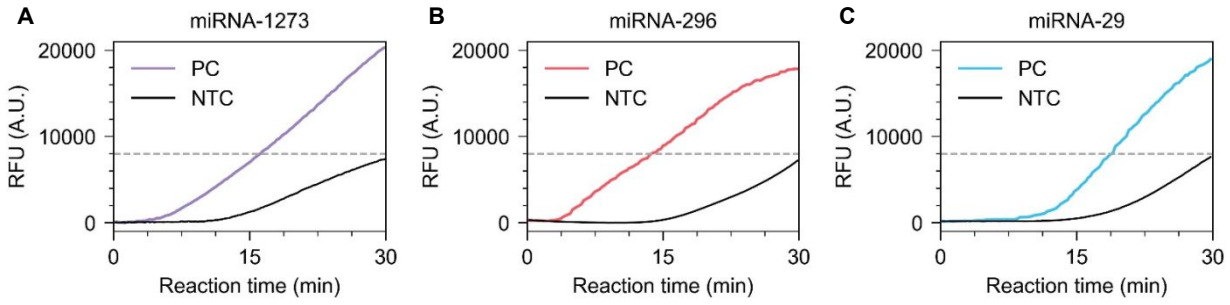


Figure S1A-C. Raw fluorescence signal (RFU) of positive control (PC) and no template control (NTC) of the miRNA targets, miRNA -1273, 296, and 29, respectively.

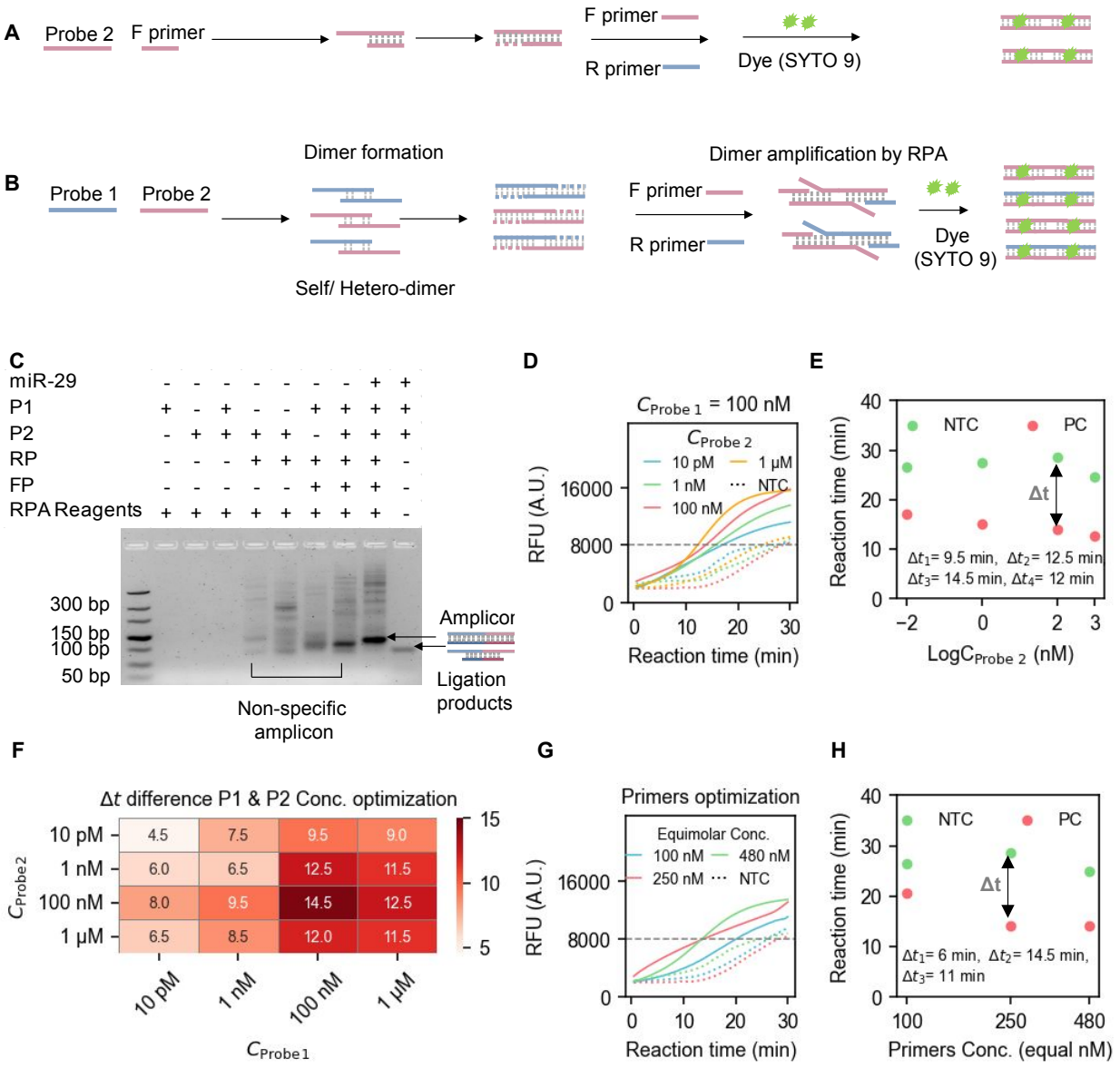


Figure S2. Optimization of probe and primer concentrations for miRNA-mediated ligation-RPA assay. (**A**) The dimer formation mechanism of the effect of P2 and F-primer on the ligation/ RPA detection. (**B**) the principle of cross dimerization of probes P1 and P2 and molecular interaction on the amplification, (**C**) gel image for nucleic acids extracted from ligation/ RPA amplicons within various conditions. (**D**) Real-time fluorescence amplification curves for varying concentrations of Probe 2 (10 pM to 1 µM),[3,4] with Probe 1 fixed at 100 nM. (**E**) plots the corresponding time-to-threshold (Tt) for PC and NTC controls versus log[Probe 2], revealing the greatest separation (Δt ≈ 14.5 min) at 100 nM. (**F**) summarizes Δt across the full Probe 1/Probe 2 matrix, again pinpointing the 100 nM/100 nM pair as optimal.(**G**) Primer titration (equimolar 100, 250, 480 nM)[3,4] identifies 250 nM as the fastest-amplifying condition. (**H**) confirms this choice by yielding the largest PC–NTC gap (Δt ≈ 14.5 min). The threshold was set at 7997 A.U.

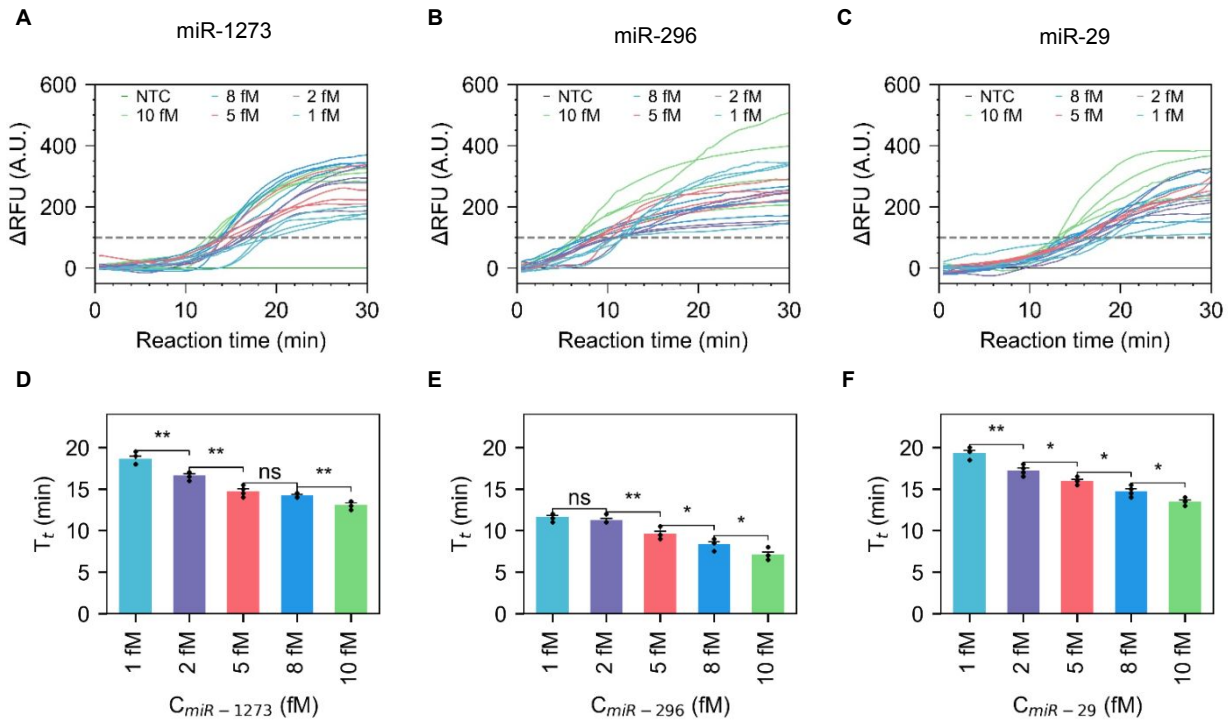


Figure S3. (**A-C**) RFU signal of the miRNA targets, from 100 pM to 1 fM, showing mean signal from 10 fM to 1 fM. (**D-F**) Bar plot showing the T-test result for miRNA -1273, 296, and 29, respectively.

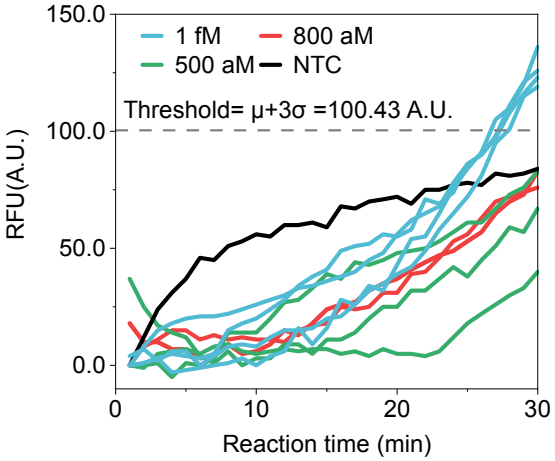


Figure S4. Raw RFU signal of 1F, 800 aM, 500 aM, and NTC of miRNA-1273, 296, and 29.

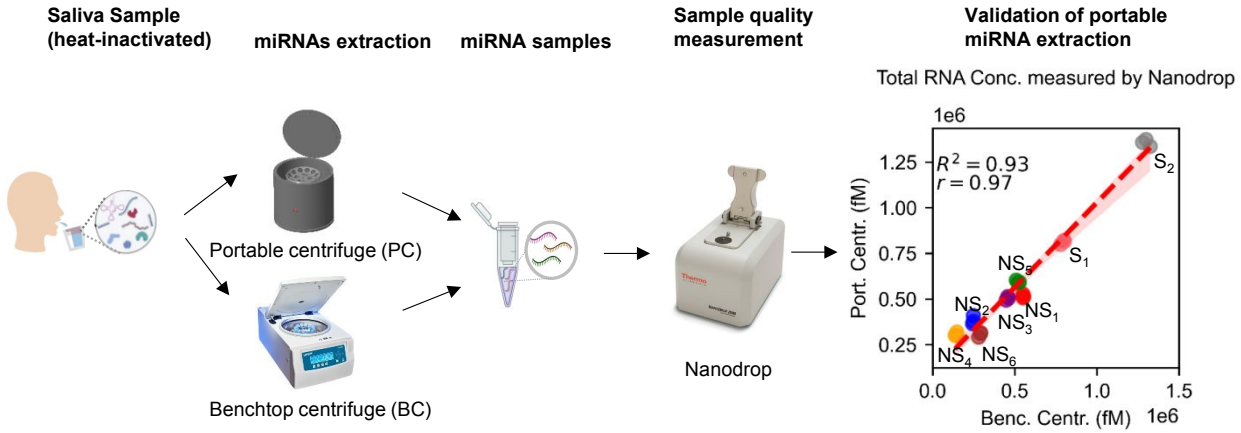


Figure S5. Validation of portable device extraction. Total RNA of 6 non-severe and 2 severe samples were measured using Nanodrop. R^2 and r value represent a strong correlation between the portable centrifuge and benchtop centrifuge.

Section 3. Supplementary Tables

Table S1. Ligation/RPA primer set targeting miRNA 1273

	Sequence (5' → 3')
Forward	TTTGTCCGAAACGACGTACAGCCATT
Reverse	TTCTACGTCGGGGTATTTACTAGCTG

Table S2. Ligation/RPA primer set targeting miRNA 296

	Sequence (5' → 3')
Forward	TTTCCACAGGGTTTGTGTACGAGATT
Reverse	TTCTCCGGTGTCCGTTCGCACTGTAG

Table S3. Ligation/RPA primer set targeting miRNA 29

	Sequence (5' → 3')
Forward	TTTCCACAGGGTTTGTGTACGAGATT
Reverse	TTCTACGTCGGGGTATTTACTAGCTG

Table S4: Recipe of reaction

Component	Volume (µL)/rxn	Final Concentration
Hybridization reaction		
miRNA (Clinical sample)	6.00	100 fM
Probe 1	1.00	100 nM
Probe 2	1.00	100 nM
Hybridize at 85°C for 2-3 min, Put it immidiately on ice box for anealing	Total vol= 8 µL	
Ligation reaction		
10x Ligase Buffer (commercial)	0.2	1 X
SpintR ligase (commercial)	0.10	25 U/µl
EB SSB (commercial)	0.13	65 ng
UltraPure Water	1.57	
	Total vol= 2 µL	
Incubate at 37°C for 10 min Annealed miRNA from Hybridization reaction	8 µL	
Ligation miRNA products	Total vol= 10 µL	
RPA reaction		
Forward Primer	0.845	250 nM
Reverse Primer	0.845	250 nM
2x reaction RPA buffer	25.00	1 X
Ligation miRNA products	10.00	-
SYTO 9	1.00	10 µM
10x Basic E-mix (commercial)	5.00	1 X
dNTPs mix (commercial)	2.30	460 µM
20x Core Reaction Mix (commercial)	2.50	1 X
Master mixture	Total vol= 47.50 µL	
Last step: Magnesium Acetate	2.50 µL	
Incubate at 37-39°C for 30 min	Total vol= 50 µL	

Table S5: Cost per reaction

Function	Reagents and Material	Function	Vendor	Usable Vol (mL)	Stock Unit Cost ($)	Volume (µL/rxn)	Cost/rxn
Saliva miRNA extraction	Chloroform	miRNA extraction	Sigma-Aldrich	80.00	$ 85.70	175.00	$ 0.19
	Ethanol	miRNA extraction	Sigma-Aldrich	3200.00	$ 456.00	656.25	$ 0.09
	miRNeasy Mini Kit	miRNA extraction	Qiagen		$ 506.00		$ 2.00
					Extraction Cost per Saliva Sample		**$ 2.28**
Each reaction tube	RPA reaction kit	RPA master mix	Twist dx	4.20	$ 260.00	43.75	$ 2.71
	miRNA 1273 F, 100 µM	RPA master mix	IDT	1.10	$ 63.22	1.06	$ 0.06
	miRNA 1273 R, 100 µM	RPA master mix	IDT	1.25	$ 63.22	1.06	$ 0.05
	miRNA 1273 P1, 100 µM	Ligation master mix	IDT	1.55	$ 71.68	1.25	$ 0.06
	miRNA 1273 P2, 100 µM	Ligation master mix	IDT	1.16	$ 95.59	1.25	$ 0.10
	miRNA 1273	miRNA sample	From Hospital	1.50		5.00	
	Syto 9, 5mM	RPA master mix	ThermoFisher	1.00	$ 359.00	1.25	$ 0.45
	Ligase Buffer	Ligation master mix	NEB	4.00	$ 28.00	0.25	$ 0.10
	SpintT ligase	Ligation master mix	NEB	10000.00	$ 111.00	0.13	$ 0.10
	EB SSB	Ligation master mix	NEB	400.00	$ 183.00	0.16	$ 0.10
	dNTPs	RPA master mix	ThermoFisher	0.80	$ 115.65	2.88	$ 0.42
						Per reaction Cost	$ 4.15
					Quantity	**Unit Cost**	**Total**
Accessories	microcentrifuge tubes1.5 mL	Tubes	ThermoFisher		1	$ 0.30	$ 0.30
	PCR tubes 200 µL	Tubes	ThermoFisher		1	$ 0.19	$ 0.19
	Pipetter tip (10 µL,20 µL, 200 µL)	Pipetteting	Fisher Scientific		3	$ 0.10	$ 0.14
					Accessory Cost in each Salaiva Sample		**$ 0.33**
		Total Per sample					**$ 6.76**

Table S6. Cost for portable Analyzer

Components	Vendor	Function	Unit Cost ($)	Quantity	Extended Cost
DC/DC step-down	Adafruit	Voltage converter	8.96	1	8.96
RGB led	Adafruit	Excitation source	0.5	8	4
0.1” spaced pin strip (male and female)	Adafruit	Connect the LED and spectral sensor PCBs to Raspberry Pi	0.9	1	0.9
Thermistor	Digikey	Temperature feedback	7.22	1	7.22
4 channel 16-bit ADC	Digikey	Convert voltage to digital values	6.35	1	6.35
Spectral sensor	Digikey	Fluorescence detection	7.5	8	60
DC barrel jack	Digikey	Power connector	0.76	1	0.76
0.1 and 10 µF SMD capacitors and 10k ohm SMD resistor	Digikey	Passive components for power regulation of ICs	0.02	40	0.8
2 ohm 1% 25W power resistor	Digikey	Heat generation	7.63	4	30.52
Raspberry Pi Microprocessor board	Digikey	Electronic coordination and data logging	10	1	10
SPDT switch	Digikey	Power delivery	0.07	1	0.07
Two PCBs for LED and spectral sensor	OSH park	Interface LED and spectral sensor with Raspberry Pi	12	1	12
Machined aluminum heat block	Hubs	Aluminum reaction tube incubator	65.5	1	65.5
Housing fabrication	In-house	3D printed housing of the analyzer	7	1	7
				Total	**$214.08**

Table S7. Cost for portable Centrifuge

Components	Vendor	Description	Function	Unit Cost ($)	Quantity	Extended Cost
ABS Filament	MakerBot	3D Print ABS Filament	Electrical Housing	60.00	0.5	30.00
Magnet	KJ Magnetics	1/8” circular magnets	Electrical Housing	0.14	16	2.24
DC motor	Amazon	6-12 V Mini DC Motor	DC Motor	6.89	1	6.89
Arduino	Arduino	Arduino Nano	Microcontroller	24.90	1	24.90
Power Switch	Digikey	SPDT Switch	Power Delivery	0.07	1	0.07
Battery	Digikey	11.1 V Li-ion 28.9 Wh	Power Delivery	27.54	1	27.54
DC Plug	Digikey	DC barrel jack	Power connector	0.76	1	0.76
LED	Digikey	Rectangular LED bar	Power indicator	2.09	1	2.09
DC Converter	Adafruit	UBEC DC/DC Step-Down Converter	Arduino power delivery	9.95	1	9.95
Resistor	Digikey	510 Ω (ohm) Resistor	LED Resistor	0.01	1	0.01
MOSFET	Digikey	N channel Power MOSFET	PWM Switch for Motor	0.88	1	0.88
Jumper wires	Adafruit	Jumper wires Connections	Electrical Connections	0.25	10	2.50
12 V Power Supply	Amazon	12V Power supply	Power Delivery	9.99	1	9.99
					Total	**$117.82**

Section 4. Supplementary Codes

Figure 1d-f and Figure S1a-c code:

```
import pandas as pd
import matplotlib.pyplot as plt
from pathlib import Path
import pathlib

# Set global font family and size
plt.rcParams['font.family'] = 'Arial'
plt.rcParams['font.size'] = 9

# File path and names
file_path = Path(r' Please paste the drive link ')
file_name_1 = 'Figure 1 and S1.xlsx'

# miRNA names and their corresponding sheet names and PC colors
miRNA_sheets = {'1273': '#9881BD', '296': '#F85662', '29': '#39C0ED'}

# Set threshold
threshold = 7997

# Read each sheet and plot
for miRNA, color in miRNA_sheets.items():
    df = pd.read_excel(file_path / file_name_1, sheet_name=miRNA)

    # Extract and process time
    time = df.iloc[:, 0] / 60  # Convert time to minutes
    ntc_data = df.iloc[:, 1]
    pc_data = df.iloc[:, 2]
```

```
    # Plotting RFU (A.U.)
    fig, ax = plt.subplots(figsize=(2.1, 1.5))
    ax.plot(time, pc_data, label='PC', color=color, linewidth=1.2)
    ax.plot(time, ntc_data, label='NTC', color='black', linewidth=0.8)

    ax.axhline(y=threshold, color='darkgray', linestyle='--', linewidth=0.8)

    # Customize the plot appearance
    ax.set_xlim(0, 30)  # Set x-limit to 0-30 minutes
    ax.set_ylim(-1000, 21000)
    ax.set_xticks(range(0, 31, 15))
    ax.set_xlabel('Reaction time (min)')
    ax.set_ylabel('RFU (A.U.)')
    ax.set_title(f'miRNA-{miRNA}', fontsize=9, fontname='Arial')

    # Customize legend
    legend = ax.legend(loc='upper left', frameon=False)
    plt.setp(legend.get_texts(), fontname='Arial')
    for line in legend.get_lines():
        line.set_linewidth(1)

    # Customize the spines
    for spine in ax.spines.values():
        spine.set_edgecolor('black')
        spine.set_linewidth(0.5)

    # Adding major and minor axis labels (ticks) at the bottom and left
    ax.tick_params(axis='both', which='major', direction='out', length=4, width=0.5, colors='black',
labelsize=9)
    ax.tick_params(axis='both', which='minor', direction='out', length=2, width=0.5, colors='black',
labelsize=9)
```

```
ax.minorticks_on()

# Removing grid
ax.grid(False)

# Setting background to white
ax.set_facecolor('white')

# Define the output directory
output_directory = pathlib.Path(r' Please paste the drive link ')
output_directory.mkdir(parents=True, exist_ok=True)  # Ensure the directory exists

# Save the plot as a high-resolution JPEG file
output_file = output_directory / f'RFU_{miRNA}.jpg'
fig.savefig(output_file, format='jpeg', dpi=600, bbox_inches='tight')

# Show the plot
plt.tight_layout()
plt.show()

# Plotting ΔRFU (A.U.) (PC - NTC)
delta_rfu = pc_data - ntc_data
fig, ax = plt.subplots(figsize=(1.3, 1.5))
ax.plot(time, delta_rfu, label='PC', color=color, linewidth=1.2)  # Use the same color for PC
ax.axhline(y=0, color='black', linestyle='-', linewidth=0.8, label='NTC')
ax.axhline(y=threshold, color='darkgray', linestyle='--', linewidth=0.8)  # Commented out

# Customize the plot appearance
ax.set_xlim(0, 30)  # Set x-limit to 0-30 minutes
ax.set_ylim(-1000, 13000)
ax.set_xticks(range(0, 31, 15))
```

```
    ax.set_xlabel('Reaction time (min)')
    ax.set_ylabel('ΔRFU (A.U.)')
    ax.set_title(f'miRNA-{miRNA}', fontsize=9, fontname='Arial')

    # Customize legend to show NTC and PC
    legend = ax.legend(loc='upper left', frameon=False, handlelength=0.75, bbox_to_anchor=(0.0,
1.05))
    plt.setp(legend.get_texts(), fontname='Arial')
    for line in legend.get_lines():
        line.set_linewidth(1)

    # Customize the spines
    for spine in ax.spines.values():
        spine.set_edgecolor('black')
        spine.set_linewidth(0.5)

    # Adding major and minor axis labels (ticks) at the bottom and left
    ax.tick_params(axis='both', which='major', direction='out', length=4, width=0.5, colors='black',
labelsize=9)
    ax.tick_params(axis='both', which='minor', direction='out', length=2, width=0.5, colors='black',
labelsize=9)
    ax.minorticks_on()

    # Removing grid
    ax.grid(False)

    # Setting background to white
    ax.set_facecolor('white')

    # Define the output directory
    output_directory = pathlib.Path(r' Please paste the drive link ')
```

```
output_directory.mkdir(parents=True, exist_ok=True)  # Ensure the directory exists

# Save the plot as a high-resolution JPEG file
output_file = output_directory / f'Delta_RFU_{miRNA}.jpg'
fig.savefig(output_file, format='jpeg', dpi=600, bbox_inches='tight')

# Show the plot
plt.tight_layout()
plt.show()
```

Figure 2b-j and S3a-f code:

```
import pandas as pd
import numpy as np
from pathlib import Path
import matplotlib.pyplot as plt
from scipy import stats
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score

# Set global font family and size
plt.rcParams['font.family'] = 'Arial'
plt.rcParams['font.size'] = 9

# Define file path
file_path = Path(r' Please paste the drive link ')
file_name = 'Figure 2 and S3a-f.xlsx'

# miRNA names
miRNAs = ['1273', '296', '29']
markers = ['s', 'o', '^']  # Different markers for each miRNA's linear plot

# Define concentration colors
colors = {
    '100 pM': '#B79F00',
    '10 pM': '#00A087',
    '1 pM': '#E84A98',
    '100 fM': '#4A5CB2',
    '10 fM': '#79D87D',
    '8 fM': '#2297E6',
    '5 fM': '#F96771',
```

```
    '2 fM': '#7570B3',
    '1 fM': '#4DBBD5'
}

# Initialize figure with 3x3 subplots
fig, axs = plt.subplots(3, 3, figsize=(7.2, 5), sharex=False)

# Loop through each miRNA
for idx, (name, marker) in enumerate(zip(miRNAs, markers)):
    # Load data for each miRNA (same as before)
    data = pd.read_excel(file_path / file_name, sheet_name=name)
    time = data.iloc[:, 0]
    ntc = data.iloc[:, 1]
    ntc_original = data.iloc[:, 26]

    concentrations = {
        '100 pM': data.iloc[:, 2],
        '10 pM': data.iloc[:, 3],
        '1 pM': data.iloc[:, 4],
        '100 fM': data.iloc[:, 5],
        '10 fM': [data.iloc[:, 6], data.iloc[:, 7], data.iloc[:, 8], data.iloc[:, 9]],
        '8 fM': [data.iloc[:, 10], data.iloc[:, 11], data.iloc[:, 12], data.iloc[:, 13]],
        '5 fM': [data.iloc[:, 14], data.iloc[:, 15], data.iloc[:, 16], data.iloc[:, 17]],
        '2 fM': [data.iloc[:, 18], data.iloc[:, 19], data.iloc[:, 20], data.iloc[:, 21]],
        '1 fM': [data.iloc[:, 22], data.iloc[:, 23], data.iloc[:, 24], data.iloc[:, 25]]
    }

    # Function to find Tt values
    def find_Tt_values(samples, time, threshold):
        Tt_values = []
        for label, sample_group in samples.items():
```

```
            for sample in sample_group if isinstance(sample_group, list) else [sample_group]:
                cross_points = np.where(sample > threshold)[0]
                if len(cross_points) > 0:
                    Tt_value = time[cross_points[0]]
                    Tt_values.append(Tt_value)
                else:
                    Tt_values.append(np.nan)
        return np.array(Tt_values)

    # Call the function to calculate Tt values
    threshold = 100.1
    Tt_values = find_Tt_values(concentrations, time, threshold)

    # Plot 1: NTC and concentration curves (top row)
    ax = axs[0, idx]
    ax.plot(time, ntc, label='NTC', color='green', linewidth=0.5)
    for label, sample_group in concentrations.items():
        color = colors[label]
        for i, sample in enumerate(sample_group if isinstance(sample_group, list) else [sample_group]):
            ax.plot(time, sample, label=label if i == 0 else "", color=color, linewidth=0.5)
    ax.axhline(y=threshold, color='grey', linestyle='--', linewidth=0.8)
    ax.set_xlim(0, 30)
    ax.set_ylim(-50, 1600)
    ax.set_yticks([0, 450, 900])
    ax.set_ylabel('ΔRFU (A.U.)')
    ax.set_xlabel('Reaction time (min)', fontsize=9, fontname='Arial')
    legend = ax.legend(fontsize=7, ncol=4, frameon=False, loc='upper center', bbox_to_anchor=(0.5, 1.05), columnspacing=0.4, handletextpad=0.2, labelspacing=0.2, handlelength=0.3)
```

```
    # Plot 2: Linear regression plot (middle row)
    log_concentrations = {
        '100 pM': np.log10(100e3),
        '10 pM': np.log10(10e3),
        '1 pM': np.log10(1e3),
        '100 fM': np.log10(100),
        '10 fM': np.log10(10),
        '8 fM': np.log10(8),
        '5 fM': np.log10(5),
        '2 fM': np.log10(2),
        '1 fM': np.log10(1)
    }

    log_concentration_values_extended = []
    for label in concentrations.keys():
        count = len(concentrations[label]) if isinstance(concentrations[label], list) else 1
        log_concentration_values_extended.extend([log_concentrations[label]] * count)

    log_concentration_values_extended = np.array(log_concentration_values_extended)
    Tt_values_nonan = np.array([val for val in Tt_values if not np.isnan(val)])
    log_concentration_values_nonan = np.array([log_concentration_values_extended[i] for i in range(len(Tt_values)) if not np.isnan(Tt_values[i])])

    model = LinearRegression()
    model.fit(log_concentration_values_nonan.reshape(-1, 1), Tt_values_nonan)
    Tt_pred = model.predict(log_concentration_values_nonan.reshape(-1, 1))

    r2 = r2_score(Tt_values_nonan, Tt_pred)
    r_value = np.corrcoef(log_concentration_values_nonan, Tt_values_nonan)[0, 1]

    ax = axs[1, idx]
```

```
    ax.scatter(log_concentration_values_nonan, Tt_values_nonan, edgecolor='black', facecolor='none', s=30, marker=marker)
    ax.plot(log_concentration_values_nonan, Tt_pred, color='red', linewidth=0.5)
    ax.set_ylabel('T$_t$ (min)')
    ax.text(0.95, 0.90, f'$R^2$={r2:.2f}\n$r$={r_value:.2f}', transform=ax.transAxes, fontsize=9, verticalalignment='top', horizontalalignment='right')

    ax.set_xlim(-0.3, 5.3)
    ax.set_ylim(-1, 22)
    ax.set_xticks(np.arange(0, 5.1, 1))
    ax.set_xlabel(f'LogC$_{{miR-{name}}}$(fM)', fontsize=9, fontname='Arial')

    # Plot 3: Bar plot with means, errors, and scatter data points
    Tt_data = {label: Tt_values[[i for i, val in enumerate(log_concentration_values_extended) if val == log_concentrations[label]]] for label in log_concentrations}
    fixed_sd = 0.05
    means = [np.nanmean(Tt_data[key]) for key in Tt_data]
    errors = [fixed_sd if len(Tt_data[key]) == 1 else np.nanstd(Tt_data[key]) / np.sqrt(len(Tt_data[key])) for key in Tt_data]
    concentration_labels = list(concentrations.keys())[::-1]  # Reverse order for correct label arrangement
    means = means[::-1]  # Reverse means to match the label order
    errors = errors[::-1]  # Reverse errors to match the label order
    positions = np.arange(len(concentration_labels))  # Set positions for each concentration level

    ax = axs[2, idx]  # Bottom row of plots for bar plots

    # Plot bars with means and standard errors
    for i, mean in enumerate(means):
        ax.bar(positions[i], mean, width=0.6, color=colors[concentration_labels[i]], edgecolor='none', capsize=1.5, linewidth=0.5)
```

```
# Scatter points on top of the bar plot with smaller diamond symbols
for i, key in enumerate(concentration_labels):
    samples = Tt_data[key]
    x = np.full_like(samples, positions[i])  # Center the points at the position of the bars
    ax.scatter(x, samples, color='black', s=0.5, marker='D')  # Use smaller diamond symbols

# Perform pairwise T-tests and collect p-values
p_values = []
for i in range(len(concentration_labels) - 1):
    samples1 = Tt_data[concentration_labels[i]]
    samples2 = Tt_data[concentration_labels[i + 1]]
    if len(samples1) > 1 and len(samples2) > 1:
        _, p_value = stats.ttest_ind(samples1, samples2)
        p_values.append((i, i + 1, p_value))
    else:
        mean1, mean2 = means[i], means[i + 1]
        t_stat = (mean1 - mean2) / np.sqrt((fixed_sd**2 / 1) + (fixed_sd**2 / 1))
        p_value = 2 * (1 - stats.t.cdf(abs(t_stat), df=1))
        p_values.append((i, i + 1, p_value))

# Define significance level labels
def significance_label(p_value):
    if p_value < 0.001:
        return '***'
    elif p_value < 0.01:
        return '**'
    elif p_value < 0.05:
        return '*'
    else:
        return 'ns'
```

```
    # Add significance bars and labels
    y_offset = 0.5
    for (i, j, p_value) in p_values:
        y = max(means[i], means[j]) + y_offset
        ax.plot([positions[i], positions[i], positions[j], positions[j]], [y, y + y_offset, y + y_offset, y], lw=0.5, color='black')
        ax.text((positions[i] + positions[j]) * 0.5, y + y_offset, significance_label(p_value), ha='center', va='bottom', fontsize=8)

    # Set x-axis labels and other plot properties
    ax.set_xticks(positions)
    ax.set_xticklabels(concentration_labels, rotation=90)
    ax.set_xlabel(f'C$_{{miR-{name}}}$ (fM)')
    ax.set_ylabel('T$_t$ (min)')
    ax.set_ylim(0, 24)
    ax.set_yticks(np.arange(0, 24, 5))

# Set spine thickness for all subplots
for ax_row in axs:
    for ax in ax_row:
        for spine in ax.spines.values():
            spine.set_linewidth(0.5)

# Finalize layout and save the figure
plt.tight_layout()
plt.savefig(file_path / 'Figure_2_plot.png', dpi=600)
```

Figure 3a-b code:

```
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.gridspec import GridSpec
from pathlib import Path
import numpy as np

# Set global font family and size
plt.rcParams['font.family'] = 'Arial'
plt.rcParams['font.size'] = 9

# Define file path
file_path = Path(r' Please paste the drive link ')
file_name_1 = 'Figure 3.xlsx'
# List of miRNA names
miRNA_names = ['1273', '296', '29']

# Define layout scaling factors to fit within the 6.4 inches width and 6.5 inches height constraint
total_width = 3  # Total width of the figure in inches
total_height = 3  # Total height of the figure in inches
nrows = 5  # 5 rows for data
ncols = 3  # Number of columns
fig_width = total_width  # Total width of the figure
fig_height = total_height  # Total height of the figure

# Set up the figure with GridSpec for better layout control
fig = plt.figure(figsize=(fig_width, fig_height))
gs = GridSpec(nrows, ncols, figure=fig, wspace=0.3, hspace=0.3)  # Adjust wspace and hspace for
spacing
```

```
# Define threshold value
threshold = 97.0

# List to store Tt values for each miRNA and corresponding sample types
Tt_values = {name: {'NTC': np.nan, 'miR-1273': np.nan, 'miR-296': np.nan, 'miR-29': np.nan,
'Triplex': np.nan} for name in miRNA_names}

# Loop through each miRNA name
for i, name in enumerate(miRNA_names):
    # Load the data from each sheet
    data = pd.read_excel(file_path / file_name_1, sheet_name=name)

    # Extract the data
    time = data.iloc[:, 0] / 60  # Convert time from seconds to minutes
    mixed = data.iloc[:, 1]
    miR_1273 = data.iloc[:, 2]
    miR_296 = data.iloc[:, 3]
    miR_29 = data.iloc[:, 4]
    ntc = data.iloc[:, 5]

    # Define labels and corresponding data in the sequence: NTC, miR-1273, miR-296, miR-29,
Mixed
    sample_data = [ntc, miR_1273, miR_296, miR_29, mixed]
    labels = ['NTC', 'miR-1273', 'miR-296', 'miR-29', 'Triplex']
    colors = ['#1AAA37', '#9881BD', '#F85662', '#C8A415', '#39C0ED']

    # Plot each type of data in the appropriate row for each sheet
    for j, (label, sample, color) in enumerate(zip(labels, sample_data, colors)):
        ax = fig.add_subplot(gs[j, i])  # Use GridSpec to position subplots
        ax.plot(time, sample, label=label, color=color, linewidth=0.8)
        ax.axhline(y=threshold, color='grey', linestyle='--', linewidth=0.8)  # Add threshold line
```

```
# Set axis limits and ticks
ax.set_xlim(0, 30)
ax.set_xticks([0, 10, 20, 30])
ax.set_ylim(-50, 500)
ax.set_yticks([0, 200, 400])

# Set axis labels for the top-left subplot
if i == 0 and j==2:
    ax.set_ylabel('ΔRFU (A.U.)', fontsize=9)

# Set x-axis labels for the bottom row
if i==1 and j==4:
    ax.set_xlabel('Reaction time (min)', fontsize=9)

# Calculate Tt value (time when the sample first crosses the threshold)
cross_points = np.where(sample > threshold)[0]
if len(cross_points) > 0:
    Tt_value = time.iloc[cross_points[0]]
else:
    Tt_value = np.nan

Tt_values[name][label] = Tt_value

# Customize spines
for spine in ax.spines.values():
    spine.set_edgecolor('black')
    spine.set_linewidth(0.5)

# Adding major and minor axis labels (ticks)
```

```
        ax.tick_params(axis='both', which='major', direction='out', length=4, width=0.5, colors='black', labelsize=7)  # Set tick label size to 7
        ax.tick_params(axis='both', which='minor', direction='out', length=1, width=0.5, colors='black', labelsize=7)
        # Remove y-axis tick labels for middle and right columns
        if i in [1, 2]:
            ax.set_yticklabels([])
        if j < 4:
            ax.set_xticklabels([])

        # Adding minor ticks
        ax.minorticks_on()

        # Removing grid
        ax.grid(False)
        # Setting background to white
        ax.set_facecolor('white')
# Save the plot of the main data
output_file = file_path / 'RFU_miRNA_panel.jpg'
plt.tight_layout()
fig.savefig(output_file, format='jpeg', dpi=1200, bbox_inches='tight')
plt.show()
###########Barplot###################
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.transforms import ScaledTranslation
# ---------------------------------------------------------------
# Tt_values
# ---------------------------------------------------------------
Tt_values = {
   '1273': {'NTC': 0, 'miR-1273': 1.5, 'miR-296': 0,   'miR-29': 0,  'Triplex': 1.4},
```

```
    '296' : {'NTC': 0, 'miR-1273': 0,   'miR-296': 3.0, 'miR-29': 0,  'Triplex': 3.8},
    '29'  : {'NTC': 0, 'miR-1273': 0,   'miR-296': 0,   'miR-29': 2.0, 'Triplex': 3.1},
}
# ---------------------------------------------------------------
# Plot parameters
# ---------------------------------------------------------------
bar_labels  = ['NTC', 'miR-1273', 'miR-296', 'miR-29', 'Triplex']
bar_colors  = ['#1AAA37', '#9881BD', '#F85662', '#C8A415', '#39C0ED']

fig_bar, ax_bar = plt.subplots(
    nrows=1, ncols=3,
    figsize=(2.75, 1),      # width 3", height 1.5"
    constrained_layout=True
)
# ---------------------------------------------------------------
# Build the three bar charts
# ---------------------------------------------------------------
for i, (miRNA, values) in enumerate(Tt_values.items()):
    # bar heights in the order of bar_labels
    bar_vals = [values[lbl] for lbl in bar_labels]
    # draw bars
    bars = ax_bar[i].bar(bar_labels, bar_vals, color=bar_colors, width=0.8)

    # y-axis
    ax_bar[i].set_ylim(0, 4)
    ax_bar[i].set_yticks([0, 2, 4])
    if i != 0:                      # hide ticks on cols 2 & 3
        ax_bar[i].set_yticklabels([])
    if i == 0:
        ax_bar[i].set_ylabel('Tt (min)', fontsize=9)
    # x-axis tick labels – rotate 90° and nudge rightward
```

```
    # --- x-axis tick labels – rotate 90° and center them on the tick ------------
    for lbl in ax_bar[i].get_xticklabels():
        lbl.set_rotation(90)
        lbl.set_ha('center')         # keep it centred on the tick
        lbl.set_va('top')            # hang from the baseline

        # move it straight **down** by 3 points so the letters sit on the tick
        dy = -3 / 72.                # 3-pt shift (in figure-ch coord)
        lbl.set_transform(lbl.get_transform() +
                          ScaledTranslation(0, dy, fig_bar.dpi_scale_trans))
    # annotate “ND” where value == 0
    for bar, val in zip(bars, bar_vals):
        if val == 0:
            ax_bar[i].text(bar.get_x() + bar.get_width()/2,
                           bar.get_height() + 0.1,
                           'ND',
                           ha='center', va='bottom',
                           fontsize=7, rotation=45)

    # tidy frame
    for spine in ax_bar[i].spines.values():
        spine.set_edgecolor('black')
        spine.set_linewidth(0.5)
    ax_bar[i].tick_params(axis='both', which='major',
                          direction='out', length=4, width=0.5, labelsize=7)

# ---------------------------------------------------------------
# Save / show
# ---------------------------------------------------------------
fig_bar.savefig('Tt_values_bar_plots.jpg', dpi=300, bbox_inches='tight')
plt.show()
```

Figure 4b-d code:

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from pathlib import Path
from scipy import stats

# Set global font family and size
plt.rcParams['font.family'] = 'Arial'
plt.rcParams['font.size'] = 9

# Define thresholds
threshold_benchtop = 7996  # Updated threshold for benchtop
threshold_portable = 100.1

# Define file path
file_path = Path(r' Please paste the drive link ')
file_name_1 = 'Figure 4.xlsx'
full_file_path = file_path / file_name_1

# Read portable data
portable_1273 = pd.read_excel(full_file_path, sheet_name='1273-portable')
portable_296 = pd.read_excel(full_file_path, sheet_name='296-portable')
portable_29 = pd.read_excel(full_file_path, sheet_name='29-portable')

# Read benchtop data
benchtop_1273 = pd.read_excel(full_file_path, sheet_name='1273-benchtop')
benchtop_296 = pd.read_excel(full_file_path, sheet_name='296-benchtop')
benchtop_29 = pd.read_excel(full_file_path, sheet_name='29-benchtop')
```

```
# miRNA names
miRNAs = ['1273', '296', '29']

# DataFrames
portable_dfs = [portable_1273, portable_296, portable_29]
benchtop_dfs = [benchtop_1273, benchtop_296, benchtop_29]

# Line width variable
lw = 0.8

# Define custom colors for 4 categories
ntc_color = '#00A087'  # Greenish color for NTC
positive_ref_color = '#2297E6'  # Light blue for positive ref
s_color = '#abdda4'  # For "S"
ns_color = '#fdae61'  # For "NS"

# Create figure and axes with updated size
fig_width = 6.75  # Total figure width in inches
fig_height = 4.5  # Updated total figure height in inches

fig, axes = plt.subplots(3, 3, figsize=(fig_width, fig_height), facecolor='white')

# Function to extract Tt values
def extract_Tt(df, threshold):
    time = df.iloc[:, 0]  # Time column
    Tt_values = []
    for col in df.columns[4:12]:  # Columns 4 to 11 (samples)
        RFU = df[col]
        # Find the time when RFU crosses the threshold
        idx = np.where(RFU >= threshold)[0]
        if len(idx) > 0:
```

```
            Tt = time.iloc[idx[0]]
        else:
            Tt = np.nan  # Threshold not reached
        Tt_values.append(Tt)
    return Tt_values

############################ RFU benchtop ##############################
for i, miRNA in enumerate(miRNAs):
    ax = axes[0, i]
    df = benchtop_dfs[i]
    time = df.iloc[:, 0]

    # Plot NTC
    ax.plot(time, df.iloc[:, 1], label='NTC', color=ntc_color, linewidth=lw)

    # Plot positive ref (100 fM and 1 fM with the same color)
    ax.plot(time, df.iloc[:, 2], label='PC', color=positive_ref_color, linewidth=lw)
    ax.plot(time, df.iloc[:, 3], color=positive_ref_color, linewidth=lw)

    # Plot samples: first half as "S", second half as "NS"
    for idx, col in enumerate(df.columns[4:9]):
        ax.plot(time, df[col], color=s_color, linewidth=lw, label='S' if idx == 0 else None)  # S for
first half
    for idx, col in enumerate(df.columns[9:12]):
        ax.plot(time, df[col], color=ns_color, linewidth=lw, label='NS' if idx == 0 else None)  # NS
for second half

    # Plot threshold line at 7996
    ax.axhline(y=threshold_benchtop, color='gray', linestyle='--', linewidth=lw)

    # Set y-axis limits and custom tick labels without repeated exponent
```

```
ax.set_ylim(0, 30000)
ax.set_yticks([0, 10000, 20000, 30000])
ax.set_yticklabels(['0', '1', '2', '3'])  # Remove exponent and simplify tick labels

# Add a multiplier notation at the top of the y-axis for benchtop plots
if i == 0:
    ax.annotate(r'$\times 10^4$', xy=(0, 1.02), xycoords='axes fraction',
                ha='center', va='bottom', fontsize=9)

# Set x-axis labels for the leftmost plot
if i == 1:
    ax.set_xticks([0, 10, 20, 30])
    ax.set_xlabel('Time (min)')

# Set spine properties
for spine in ax.spines.values():
    spine.set_edgecolor('black')
    spine.set_linewidth(0.5)

# Set title with Arial 9 font size
ax.set_title(f'miRNA {miRNA}', fontsize=9)

# Set y-label only for first column
if i == 0:
    ax.set_ylabel('RFU (A.U.)')
#else:
    # Remove y-tick labels from next two plots
    #ax.set_yticklabels([])

# Remove x-tick labels for first two rows
if i > 0:
```

```
        #ax.set_xticklabels([])
        ax.annotate(r'$\times 10^4$', xy=(0, 1.02), xycoords='axes fraction',
                    ha='center', va='bottom', fontsize=9)
        ax.set_xticks([0, 10, 20, 30])

    # Add legend with specified formatting
    if i == 0:
        ax.legend(fontsize=9, ncol=2, frameon=False, loc='upper center', bbox_to_anchor=(0.25,
1.05), columnspacing=0.3, handletextpad=0.2,  handlelength=0.7)

############################ RFU portable ##############################
for i, miRNA in enumerate(miRNAs):
    ax = axes[1, i]
    df = portable_dfs[i]
    time = df.iloc[:, 0]

    # Plot NTC
    ax.plot(time, df.iloc[:, 1], color=ntc_color, linewidth=lw)

    # Plot positive ref (100 fM and 1 fM with the same color)
    ax.plot(time, df.iloc[:, 2], color=positive_ref_color, linewidth=lw)
    ax.plot(time, df.iloc[:, 3], color=positive_ref_color, linewidth=lw)

    # Plot samples: first half as "S", second half as "NS"
    for idx, col in enumerate(df.columns[4:8]):
        ax.plot(time, df[col], color=s_color, linewidth=lw, label='S' if idx == 0 else None)
    for idx, col in enumerate(df.columns[8:12]):
        ax.plot(time, df[col], color=ns_color, linewidth=lw, label='NS' if idx == 0 else None)

    # Plot threshold line
    ax.axhline(y=threshold_portable, color='gray', linestyle='--', linewidth=lw)
```

```
    # Set y-axis limits and ticks for portable (without scientific notation)
    ax.set_ylim(-100, 500)
    ax.set_yticks([0, 200, 400])
    ax.set_yticklabels(['0', '2', '4'])
    ax.annotate(r'$\times 10^2$', xy=(0, 1.02), xycoords='axes fraction',ha='center', va='bottom', fontsize=9)
    # Set y-label only for first column
    if i == 0:
        ax.set_ylabel('RFU (A.U.)')
    #else:
        # Remove y-tick labels from next two plots
        #ax.set_yticklabels([])

    # Set x-tick labels only for the middle plot in the first column
    if i == 1:
        #ax.set_xticks([0, 10, 20, 30])
        ax.set_xlabel('Time (min)')
    #else:
        # Remove x-tick labels for second and third plots in the row
        #ax.set_xticklabels([])

############################ Tt comparison ##############################
markers = ['s', 'o', '^']  # Rectangle (square), circle, and triangle markers

for i, (miRNA, marker) in enumerate(zip(miRNAs, markers)):
    ax = axes[2, i]

    # Extract Tt values for benchtop and portable data
    Tt_benchtop = extract_Tt(benchtop_dfs[i], threshold_benchtop)
    Tt_portable = extract_Tt(portable_dfs[i], threshold_portable)
```

```
    # Convert to numpy arrays and remove NaNs
    Tt_benchtop, Tt_portable = np.array(Tt_benchtop), np.array(Tt_portable)
    mask = ~np.isnan(Tt_benchtop) & ~np.isnan(Tt_portable)
    Tt_benchtop, Tt_portable = Tt_benchtop[mask], Tt_portable[mask]

    # Plot scatter plot with specified marker and regression line
    ax.scatter(Tt_benchtop, Tt_portable, s=10, color='#409BD3', marker=marker)
    if len(Tt_benchtop) > 0 and len(Tt_portable) > 0:
        slope, intercept = np.polyfit(Tt_benchtop, Tt_portable, 1)
        fit_line = slope * np.array([Tt_benchtop.min(), Tt_benchtop.max()]) + intercept
        ax.plot([Tt_benchtop.min(), Tt_benchtop.max()], fit_line, color='red', linewidth=lw)

        # Calculate R and R^2 and add them to the plot
        r_value, _ = stats.pearsonr(Tt_benchtop, Tt_portable)
        r_squared = r_value ** 2
        ax.text(0.05, 0.95, f'r = {r_value:.2f}\nR² = {r_squared:.2f}', transform=ax.transAxes,
fontsize=9, verticalalignment='top')

    # Set x-axis and y-axis limits to be consistent across all plots in this section
    ax.set_xlim(3, 22)
    ax.set_ylim(3, 22)

    # Set tick marks for all plots
    ax.set_xticks([5, 10, 15, 20])
    ax.set_yticks([5, 10, 15, 20])

    # Only show tick labels for the first plot (i=0)
    if i == 0:
        ax.set_ylabel('Portable Tt')
    #else:
```

```
        # Remove x and y tick labels for the second and third plots
        #ax.set_xticklabels([])
        #ax.set_yticklabels([])

    # Set x-axis label only for the middle plot in this row
    if i == 1:
        ax.set_xlabel('Benchtop Tt')



############################## Figure adjustments ################################
# Add labels to the left side of the figure
fig.text(0.03, 0.845, 'Benchtop', ha='center', va='center', rotation='vertical', fontsize=9)
fig.text(0.03, 0.52, 'Portable', ha='center', va='center', rotation='vertical', fontsize=9)
#fig.text(0.03, 0.205, 'Comparison', ha='center', va='center', rotation='vertical', fontsize=9)

# Adjust layout and save
plt.tight_layout(rect=[0.03, 0, 1, 1])
output_file = file_path / 'Figure_4_output.png'
plt.savefig(output_file, dpi=300)
plt.show()
```

Figure 5a-f code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from pathlib import Path
from sklearn.linear_model import LinearRegression

# Set global font family and size
plt.rcParams['font.family'] = 'Arial'
plt.rcParams['font.size'] = 9

# miRNA names
miRNA_names = ['1273', '296', '29']

# Define file paths
file_path_1 = Path(r' Please paste the drive link ')
file_name_1 = 'Figure 5_114 samples.xlsx'
file_path_2 = Path(r' Please paste the drive link ')
file_name_2 = 'Figure 5_40 samples.xlsx'
# Number of samples
num_ns_samples_1 = 10
num_s_samples_1 = 10
num_ns_samples_2 = 30
num_s_samples_2 = 10

# Known concentrations and Tt values for each miRNA
known_concentrations_dict = {
    '1273': np.array([100, 20, 10, 5, 1]),
    '296': np.array([100, 20, 10, 5, 1]),
    '29': np.array([100, 20, 10, 5, 1])
```

```
}

known_Tt_dict = {
    '1273': np.array([9, 13, 14, 16, 18]),
    '296': np.array([4, 7, 10, 12.5, 13]),
    '29': np.array([10, 14, 17, 18.5, 20])
}

# Create a 2x3 grid for all miRNAs
# (We'll selectively label axes as requested.)
fig, axs = plt.subplots(2, 3, figsize=(3.55, 3.0))

def find_Tt_values(samples, time, threshold=100):
    """Return the time-to-threshold array for each sample column."""
    Tt_values = []
    for i in range(samples.shape[1]):
        sample = samples.iloc[:, i]
        cross_points = np.where(sample > threshold)[0]
        if len(cross_points) > 0:
            Tt_value = time.iloc[cross_points[0]]
            Tt_values.append(Tt_value)
    return np.array(Tt_values)

for idx, name in enumerate(miRNA_names):
    # Read data
    data_1 = pd.read_excel(file_path_1 / file_name_1, sheet_name=name, engine='openpyxl')
    data_2 = pd.read_excel(file_path_2 / file_name_2, sheet_name=name, engine='openpyxl')

    # Extract time columns
    time_1 = data_1.iloc[:, 0]
    time_2 = data_2.iloc[:, 0]
```

```
# File 1 concentrations
ntc1_1 = data_1.iloc[:, 1]
conc_100fM_1 = data_1.iloc[:, 3]
conc_1fM_1 = data_1.iloc[:, 9]
ns_samples_1 = data_1.iloc[:, 11:21].apply(pd.to_numeric, errors='coerce')
s_samples_1 = data_1.iloc[:, 21:31].apply(pd.to_numeric, errors='coerce')

# File 2 concentrations
ntc1_2 = data_2.iloc[:, 1]
ntc2_2 = data_2.iloc[:, 2]
conc_100fM_2 = data_2.iloc[:, 3]
conc_1fM_2 = data_2.iloc[:, 5]
ns_samples_2 = data_2.iloc[:, 7:37].apply(pd.to_numeric, errors='coerce')
s_samples_2 = data_2.iloc[:, 37:47].apply(pd.to_numeric, errors='coerce')

# Calculate Tt values for non-severe and severe samples (File 1, for example)
Tt_values_non_severe = find_Tt_values(ns_samples_1, time_1, threshold=100)
Tt_values_severe = find_Tt_values(s_samples_1, time_1, threshold=100)

# Top row -> ΔRFU vs. Time
ax_rfu = axs[0, idx]
ax_rfu.plot(time_1, ntc1_1, label='NTC', color='black', linewidth=0.5)
ax_rfu.plot(time_2, ntc1_2, color='black', linewidth=0.5)
ax_rfu.plot(time_1, conc_100fM_1, label='100fM', color='blue', linewidth=0.5)
ax_rfu.plot(time_2, conc_100fM_2, color='blue', linewidth=0.5)
ax_rfu.plot(time_1, conc_1fM_1, label='1fM', color='pink', linewidth=0.5)
ax_rfu.plot(time_2, conc_1fM_2, color='pink', linewidth=0.5)

ns_color, s_color = '#00B050', '#F85663'
for i in range(num_ns_samples_1):
```

```
        ax_rfu.plot(time_1, ns_samples_1.iloc[:, i], color=ns_color, linewidth=0.5)
    for i in range(num_s_samples_1):
        ax_rfu.plot(time_1, s_samples_1.iloc[:, i], color=s_color, linewidth=0.5)
    for i in range(num_ns_samples_2):
        ax_rfu.plot(time_2, ns_samples_2.iloc[:, i], color=ns_color, linewidth=0.5)
    for i in range(num_s_samples_2):
        ax_rfu.plot(time_2, s_samples_2.iloc[:, i], color=s_color, linewidth=0.5)

    ax_rfu.set_xlim(0, 30)
    ax_rfu.set_xticks([0, 15, 30])
    ax_rfu.set_ylim(-50, 800)
    ax_rfu.set_yticks([0, 400, 800])
    ax_rfu.axhline(y=101, color='gray', linestyle='--', linewidth=0.5)
    ax_rfu.set_title(f'miRNA-{name}', fontsize=9, fontname='Arial')

    # Legend
    ax_rfu.plot([], [], color=ns_color, label='NS')
    ax_rfu.plot([], [], color=s_color, label='S')
    legend = ax_rfu.legend(
        fontsize=7, ncol=2, frameon=False, loc='upper center',
        bbox_to_anchor=(0.5, 1.0), columnspacing=0.2, handletextpad=0.2, labelspacing=0.2,
handlelength=0.5
    )
    plt.setp(legend.get_texts(), fontname='Arial')

    for spine in ax_rfu.spines.values():
        spine.set_edgecolor('black')
        spine.set_linewidth(0.5)

    # Bottom row -> Calibration
    ax_calib = axs[1, idx]
```

```
known_concentrations = known_concentrations_dict[name]
known_Tt = known_Tt_dict[name]

regressor = LinearRegression()
regressor.fit(known_Tt.reshape(-1, 1), np.log(known_concentrations))
r_squared = regressor.score(known_Tt.reshape(-1, 1), np.log(known_concentrations))
r_value = np.sqrt(r_squared)

# Evaluate concentrations for non-severe and severe Tt values
predicted_conc_ns = np.exp(regressor.predict(Tt_values_non_severe.reshape(-1, 1)))
predicted_conc_s  = np.exp(regressor.predict(Tt_values_severe.reshape(-1, 1)))

# Plot regression
Tt_range = np.linspace(0, 22, 150)
ax_calib.plot(Tt_range, np.exp(regressor.predict(Tt_range.reshape(-1, 1))),
            color='#F96771', linewidth=0.5, label='Fit')
ax_calib.scatter(known_Tt, known_concentrations, color='#F96771',
               marker='s', s=7, label='Known')
ax_calib.scatter(Tt_values_non_severe, predicted_conc_ns, facecolors='none',
               edgecolors=ns_color, s=10, label='NS')
ax_calib.scatter(Tt_values_severe, predicted_conc_s, facecolors='none',
               edgecolors=s_color, s=10, label='S')

ax_calib.text(
   1, 1,
   f'$R^2 = {r_squared:.2f}$\n$r = {r_value:.2f}$',
   transform=ax_calib.transAxes,
   fontsize=7,
   verticalalignment='top',
   horizontalalignment='right'
```

```
)

ax_calib.set_yscale('log', base=10)
ax_calib.set_ylim(0.5, 1000)
ax_calib.set_xlim([3, 23])
ax_calib.set_xticks([3, 13,23])

# Set spine thickness
for spine in ax_calib.spines.values():
    spine.set_edgecolor('black')
    spine.set_linewidth(0.5)

# ---------- Adjust labels according to position ----------
# Top row (ΔRFU plots)
if idx == 0:
    # Left column only
    ax_rfu.set_ylabel(r'$\Delta$RFU (A.U.)')
else:
    # Remove y-label on other columns
    ax_rfu.set_ylabel(None)

if idx == 1:
    # Middle column only
    ax_rfu.set_xlabel('Reaction time (min)')
else:
    # Remove x-label from left and right columns
    ax_rfu.set_xlabel(None)

# Bottom row (Calibration plots)
if idx == 0:
    # Left column only
```

```
        ax_calib.set_ylabel(r'C (fM)')
    else:
        ax_calib.set_ylabel(None)

    if idx == 1:
        # Middle column only
        ax_calib.set_xlabel(r'T$_{t}$ (min)')
    else:
        ax_calib.set_xlabel(None)
    # ----------------------------------------------------------

plt.tight_layout()
save_path = Path(r'Please paste the drive link where you want to save\Combined_2x3_Plot.jpg')
plt.savefig(save_path, dpi=600, format='jpg', bbox_inches='tight')
plt.show()
```

Figure 5g code:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib.colors import LinearSegmentedColormap
from matplotlib.gridspec import GridSpec
from pathlib import Path

# Define the file path
file_path = Path(r'Please paste the drive link\Figure 5g.xlsx')  # Adjust file name if needed
df = pd.read_excel(file_path)

# Define the two-color custom colormap
cmap = LinearSegmentedColormap.from_list("custom_cmap", ['#F85663', '#00E266'])

# Set font and figure properties
plt.rcParams.update({
    'font.size': 9,
    'font.family': 'Arial'
})

# Create figure with GridSpec for equal column widths
fig = plt.figure(figsize=(3.5, 2.9))
gs = GridSpec(1, 2, width_ratios=[20, 1], figure=fig)

# Plotting all samples heatmap
ax = fig.add_subplot(gs[0, 0])
cbar_ax = fig.add_subplot(gs[0, 1])

# Transpose and plot heatmap using the custom two-color colormap
```

```
sns.heatmap(df.T, annot=False, cmap=cmap, cbar=True, ax=ax, vmin=0, vmax=200,
            cbar_ax=cbar_ax, mask=df.T.isna(), linewidths=0.2, linecolor='white')

# Customize ticks and labels
ax.set_xticks([0,  39,  79,  119,  153])
ax.set_xticklabels([1,  40, 80,  120, 154])
#ax.text(-100, -.25, 'Sample #', fontsize=9, fontname='Arial', va='center')
ax.xaxis.tick_top()  # Move x-axis ticks to the top
ax.set_yticklabels(['miR-1273', 'miR-296', 'miR-29'], rotation=90)

# Adding Non-severe and Severe labels at the bottom
ax.text(55, 3.5, 'Non-severe', fontsize=9, fontname='Arial', va='center', ha='center')
ax.text(135, 3.5, 'Severe', fontsize=9, fontname='Arial', va='center', ha='center')

cbar_ax.set_ylabel('Conc.(fM)', fontsize=9, fontname='Arial')
#cbar_ax.yaxis.set_label_coords(-1.5, 0.5)

# Adjust layout and save the figure
plt.tight_layout()
output_file_path_combined = file_path.parent / 'heatmap_two_colors.png'
plt.savefig(output_file_path_combined, dpi=600, bbox_inches='tight')
plt.show()
```

Figure 5h-j code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
from scipy.interpolate import interp1d
from pathlib import Path

# Set global font family and size
plt.rcParams.update({'font.family': 'Arial', 'font.size': 14})

# File paths and parameters
file_path = Path(r'Please paste the drive link')
file_name_114 = 'Figure 5_114 samples.xlsx'  # Dataset from first code
file_name_40 = 'Figure 5_40 samples.xlsx'  # Dataset from second code
miRNA_names = ['1273', '296', '29']

# Known concentrations and Tt values for Figure 5_114 samples
known_concentrations_dict_114 = {
    '1273': np.array([100, 20, 10, 5, 1]),
    '296': np.array([100, 20, 10, 5, 1]),
    '29': np.array([100, 20, 10, 5, 1])
}
known_Tt_dict_114 = {
    '1273': np.array([9, 13, 14, 16, 18]),
    '296': np.array([4, 7.00, 10, 12.5, 13]),
    '29': np.array([10, 14, 17, 18.5, 20])
}

# Known concentrations and Tt values for Figure 5_40 samples
```

```
known_concentrations_dict_40 = {
    '1273': np.array([100, 20, 10, 8, 5, 2, 1]),
    '296': np.array([100, 20, 10, 8, 5, 2, 1]),
    '29': np.array([100, 20, 10, 8, 5, 2, 1])
}
known_Tt_dict_40 = {
    '1273': np.array([10.5, 12.5, 13.125, 14.25, 14.75, 16.625, 20]),
    '296': np.array([5, 7.00, 7.75, 8.5, 9.625, 11.75, 13]),
    '29': np.array([10.5, 12.75, 13.5, 15.75, 17, 19.25, 20.5])
}

# Function to find Tt values where RFU crosses a threshold
def find_Tt_values(samples, time, threshold):
    Tt_values = []
    for i in range(samples.shape[1]):
        sample = samples.iloc[:, i]
        cross_points = np.where(sample > threshold)[0]
        if len(cross_points) > 0:
            Tt_value = time.iloc[cross_points[0]]
            Tt_values.append(Tt_value)
    return np.array(Tt_values)

# Function to process Figure 5_114 samples data for each miRNA
def process_figure5_114_miRNA(name):
    # Load portable data
    data_portable = pd.read_excel(file_path / file_name_114, sheet_name=name,
engine='openpyxl')
    time_portable = data_portable.iloc[:, 0]
    ns_samples_portable = data_portable.iloc[:, 11:21].apply(pd.to_numeric, errors='coerce')
    s_samples_portable = data_portable.iloc[:, 21:31].apply(pd.to_numeric, errors='coerce')
    Tt_values_ns_portable = find_Tt_values(ns_samples_portable, time_portable, threshold=100)
```

```
    Tt_values_s_portable = find_Tt_values(s_samples_portable, time_portable, threshold=100)

    # Load benchtop data
    sheet_name_benchtop = f"{name} (100)"
    data_benchtop = pd.read_excel(file_path / file_name_114, sheet_name=sheet_name_benchtop,
engine='openpyxl')
    time_benchtop = data_benchtop.iloc[:, 0]
    ns_samples_benchtop = data_benchtop.iloc[:, 5:76].apply(pd.to_numeric, errors='coerce')
    s_samples_benchtop = data_benchtop.iloc[:, 76:99].apply(pd.to_numeric, errors='coerce')
    Tt_values_ns_benchtop = find_Tt_values(ns_samples_benchtop, time_benchtop,
threshold=5000)
    Tt_values_s_benchtop = find_Tt_values(s_samples_benchtop, time_benchtop, threshold=5000)

    # Combine Tt values
    Tt_values_ns = np.concatenate([Tt_values_ns_portable, Tt_values_ns_benchtop])
    Tt_values_s = np.concatenate([Tt_values_s_portable, Tt_values_s_benchtop])

    # Interpolate to obtain concentrations
    known_concentrations = known_concentrations_dict_114[name]
    known_Tt = known_Tt_dict_114[name]
    interpolator = interp1d(known_Tt, known_concentrations, fill_value="extrapolate")

    concentrations_ns = interpolator(Tt_values_ns)
    concentrations_s = interpolator(Tt_values_s)

    return concentrations_ns, concentrations_s

# Function to process Figure 5_40 samples data for each miRNA
def process_figure5_40_miRNA(name):
    sheet_name = name
    known_concentrations = known_concentrations_dict_40[name]
```

```
    known_Tt = known_Tt_dict_40[name]

    # Import the data from the Excel sheet
    data = pd.read_excel(file_path / file_name_40, sheet_name=sheet_name, engine='openpyxl')

    # Extract columns
    time = data.iloc[:, 0]
    ns_samples = data.iloc[:, 7:37].apply(pd.to_numeric, errors='coerce')
    s_samples = data.iloc[:, 37:47].apply(pd.to_numeric, errors='coerce')

    # Extract Tt values
    Tt_values_ns = find_Tt_values(ns_samples, time, threshold=100)
    Tt_values_s = find_Tt_values(s_samples, time, threshold=100)

    # Interpolate to obtain concentrations
    interpolator = interp1d(known_Tt, known_concentrations, fill_value="extrapolate")
    concentrations_ns = interpolator(Tt_values_ns)
    concentrations_s = interpolator(Tt_values_s)

    return concentrations_ns, concentrations_s

# Initialize dictionaries to store combined concentrations
concentration_data_ns = {}
concentration_data_s = {}

for name in miRNA_names:
    # Process Figure 5_114 samples data
    concentrations_ns_114, concentrations_s_114 = process_figure5_114_miRNA(name)

    # Process Figure 5_40 samples data
    concentrations_ns_40, concentrations_s_40 = process_figure5_40_miRNA(name)
```

```
    # Combine concentrations from both datasets
    concentrations_ns = np.concatenate([concentrations_ns_114, concentrations_ns_40])
    concentrations_s = np.concatenate([concentrations_s_114, concentrations_s_40])

    # Store the combined concentrations
    concentration_data_ns[name] = concentrations_ns
    concentration_data_s[name] = concentrations_s

# Now, we can proceed to plot the bar plots using the combined data
import matplotlib.pyplot as plt

# Define function to label significance levels
def significance_label(p_value):
    if p_value < 0.001:
        return '***'
    elif p_value < 0.01:
        return '**'
    elif p_value < 0.05:
        return '*'
    else:
        return 'ns'

# Prepare the figure and axes
fig, axs = plt.subplots(1, 3, figsize=(3.5, 2), sharey=True)
plt.rcParams.update({'font.size': 9})

for i, name in enumerate(miRNA_names):
    # Retrieve concentrations
    concentrations_non_severe = concentration_data_ns[name]
    concentrations_severe = concentration_data_s[name]
```

```
    # Calculate T-test between non-severe and severe samples
    t_stat, p_value = stats.ttest_ind(concentrations_non_severe, concentrations_severe, equal_var=False)
    sig_label = significance_label(p_value)

    # Plotting the bar plot for non-severe and severe samples
    ax = axs[i]
    bar_width = 0.6
    positions = np.array([1, 2])
    means = [np.mean(concentrations_non_severe), np.mean(concentrations_severe)]
    errors = [np.std(concentrations_non_severe), np.std(concentrations_severe)]

    # Create bar plot
    ax.bar(positions, means, yerr=errors, width=bar_width, color=['#00E266', '#F85663'], edgecolor='none', capsize=3)

    # Scatter points on top of the bar plot
    for j, y in enumerate([concentrations_non_severe, concentrations_severe]):
        x = np.random.normal(positions[j], 0.04, size=len(y))
        ax.scatter(x, y, color='black', zorder=2, s=3)

    # Set labels and axis limits
    ax.set_xticks(positions)

    # Only the middle subplot gets x-axis labels rotated 45 degrees
    if i == 1:
        ax.set_xticklabels(['Non-severe', 'Severe'], fontsize=9, fontname='Arial', rotation=30)
    else:
        ax.set_xticklabels([])  # Remove x-axis labels for other subplots
```

```
    ax.set_yticks([0, 100, 200])  # Add custom y-ticks
    ax.tick_params(axis='y', labelsize=9)  # Set y-tick label font size to 9
    for label in ax.get_yticklabels():  # Ensure Arial font for y-tick labels
        label.set_fontname('Arial')
    ax.set_ylim(0, 250)
    ax.set_title(f'miRNA-{name}', fontsize=9, fontname='Arial')

    # Add significance label
    max_value = max(np.max(concentrations_non_severe), np.max(concentrations_severe))
    y, h, col = max_value + 10, 5, 'k'
    ax.plot([positions[0], positions[0], positions[1], positions[1]], [y, y + h, y + h, y], lw=0.5, c=col)
    ax.text((positions[0] + positions[1]) * 0.5, y + h, sig_label, ha='center', va='bottom', color=col,
fontsize=9, fontweight='bold')
    print(f"P-value for miRNA-{name}: {p_value}")

    # Compute and display means
    mean_non_severe = np.mean(concentrations_non_severe)
    mean_severe = np.mean(concentrations_severe)
    print(f"Mean concentration for Non-severe samples (miRNA-{name}): {mean_non_severe}")
    print(f"Mean concentration for Severe samples (miRNA-{name}): {mean_severe}")

    # Adjust spine thickness
    for spine in ax.spines.values():
        spine.set_linewidth(0.5)

# Set a shared y-axis label for all subplots with reduced font size
fig.text(0.04, 0.5, 'C (fM)', va='center', rotation='vertical', fontsize=9)

# Adjust layout and save the figure
plt.tight_layout(rect=[0.05, 0, 1, 1])  # Adjust the left margin to accommodate y-axis label
output_file_path_combined = file_path / 'barplot.png'
```

```
plt.savefig(output_file_path_combined, dpi=600, bbox_inches='tight')
plt.show()
```

Figure 5k code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc
from sklearn.linear_model import LogisticRegression
from pathlib import Path

# ---------------------------------
# Global Plot Settings: Arial, Size 9
# ---------------------------------
plt.rcParams.update({'font.family': 'Arial', 'font.size': 9})

# ------------------------------------------------
# File paths
# ------------------------------------------------
file_path = Path(r'Please paste the drive link')
output_path_combined = file_path / "Combined_Plot.png"

# ------------------------------------------------
# Define miRNA Combinations with Subscripts
# ------------------------------------------------
combinations = {
    "miR$_{1273}$":      ["miR-1273"],
    "miR$_{296}$":       ["miR-296"],
    "miR$_{29}$":        ["miR-29"],
    "miR$_{1273+296}$":  ["miR-1273", "miR-296"],
    "miR$_{1273+29}$":   ["miR-1273", "miR-29"],
    "miR$_{296+29}$":    ["miR-296",  "miR-29"],
    "All":               ["miR-1273", "miR-296", "miR-29"]
```

```
}

# Set order of combinations in plots
combos_in_order = list(combinations.keys())

# Define color palette
color_cycle = [
    "#4DBBD5",
    "#E84A98",
    "#F96771",
    "#2297E6",
    "#79D87D",
    "#4A5CB2",
    "#7570B3"
]

# ----------------------------------------
# Load pre-processed DataFrame `df_combined`
# ----------------------------------------
df_combined = pd.read_excel(file_path / "Figure 5 k-l.xlsx")

# Ensure correct labels for logistic regression
y = df_combined["Label"].values
auc_dict = {}  # Store AUC values

# ----------------------------------------
# Create Combined Plot with Two Subplots
# ----------------------------------------
fig, axes = plt.subplots(1, 2, figsize=(3.5, 1.8))  # Total size: 3.5 x 1.8 inches

# ------------------------------
```

```
# Subplot 1: ROC Curves
# ------------------------------
ax1 = axes[0]

for i, combo_key in enumerate(combos_in_order):
    feature_cols = combinations[combo_key]
    X = df_combined[feature_cols].values

    # Fit logistic regression
    model = LogisticRegression()
    model.fit(X, y)

    # Compute ROC
    y_scores = model.predict_proba(X)[:, 1]
    fpr, tpr, _ = roc_curve(y, y_scores)
    auc_dict[combo_key] = auc(fpr, tpr)

    # Line style and width
    if combo_key == "All":
        ls = "-"   # solid line
        lw = 1.2   # thicker line
    else:
        ls = "--"  # dashed line
        lw = 0.7   # thinner line

    # Plot
    ax1.plot(
        fpr, tpr,
        linestyle=ls,
        linewidth=lw,
        color=color_cycle[i],
```

```
        label=f'{combo_key}'
    )

# Add diagonal reference line
ax1.plot([0, 1], [0, 1], '--', color='gray', linewidth=0.5)  # thinner line

# Formatting for ROC plot
ax1.set_xlabel('1 - Specificity', fontsize=9)
ax1.set_ylabel('Sensitivity', fontsize=9)
ax1.set_xlim([-0.1, 1])
ax1.set_xticks([0, 0.5, 1])
ax1.set_ylim([0, 1.1])  # Adjusted y-axis limit

# ------------------------------
# Subplot 2: AUC Bar Plot
# ------------------------------
ax2 = axes[1]

combo_list = combos_in_order
auc_values = [auc_dict[c] for c in combo_list]

# Use same colors for bars
bar_colors = [color_cycle[i] for i in range(len(combo_list))]

ax2.bar(combo_list, auc_values, color=bar_colors)
ax2.set_ylim(0.75, 1.00)  # Fix Y-axis range
ax2.set_ylabel("AUC", fontsize=9)
ax2.set_xticks(range(len(combo_list)))
ax2.set_xticklabels(combo_list, rotation=90, fontsize=7, ha='center')  # Rotate labels and align center
```

```
# No x-axis title for bar plot
ax2.set_xlabel("")

# ------------------------------
# Final Adjustments
# ------------------------------
plt.tight_layout()

# Save the combined plot
plt.savefig(output_path_combined, dpi=600, bbox_inches='tight')
print('Combined plot saved at:', output_path_combined)
plt.show()
```

Section 5. References

(1) Higgins, M.; Ravenhall, M.; Ward, D.; Phelan, J.; Ibrahim, A.; Forrest, M. S.; Clark, T. G.; Campino, S. PrimedRPA: Primer Design for Recombinase Polymerase Amplification Assays. *Bioinformatics* **2019**, *35* (4), 682–684. https://doi.org/10.1093/bioinformatics/bty701.

(2) Politza, A. J.; Liu, T.; Kshirsagar, A.; Dong, M.; Ahamed, M. A.; Guan, W. A Portable Centrifuge for Universal Nucleic Acid Extraction at the Point-of-Care. Rochester, NY April 3, 2024. https://doi.org/10.2139/ssrn.4781228.

(3) Wee, E. J. H.; Trau, M. Simple Isothermal Strategy for Multiplexed, Rapid, Sensitive, and Accurate miRNA Detection. *ACS Sens.* **2016**, *1* (6), 670–675. https://doi.org/10.1021/acssensors.6b00105.

(4) Wang, P.; Ma, C.; Zhang, X.; Chen, L.; Yi, L.; Liu, X.; Lu, Q.; Cao, Y.; Gao, S. A Ligation/Recombinase Polymerase Amplification Assay for Rapid Detection of SARS-CoV−2. *Front. Cell. Infect. Microbiol.* **2021**, *11*, 680728. https://doi.org/10.3389/fcimb.2021.680728.