Squiggy: Interactive Nanopore Squiggle Visualization in Positron IDE

Jay Hesselberth[1*], Laura K. White[1], Adam Quach[1]

[1]RNA Bioscience Initiative, University of Colorado School of Medicine, Aurora, CO 80045, USA
[*]Corresponding author. jay.hesselberth@gmail.com

Abstract

Motivation: Oxford Nanopore sequencing technologies generate raw electrical signals (squiggles) that provide rich information beyond base calls, including base modifications, secondary structure, and read quality. However, existing visualization tools either operate as standalone applications disconnected from analytical workflows, or require complex command-line operations that impede exploratory analysis. There is a need for interactive visualization tools that integrate seamlessly with modern data science environments.

Results: We present Squiggy, a Positron IDE extension that enables interactive visualization and exploration of nanopore signal data directly within the data analysis environment. Squiggy leverages the active Python kernel to provide real-time access to POD5 files and BAM alignments, supporting multiple visualization modes including single-read, overlay, stacked, event-aligned, and multi-read aggregate plots. The extension provides advanced features for base modification analysis, motif-based filtering, and multi-sample comparisons, all through an intuitive graphical interface. Built on the Strategy Pattern, Squiggy's architecture facilitates easy extension with new plot types and analysis modes.

Availability and Implementation: Squiggy is implemented as a TypeScript/Python extension for Positron IDE. The Python backend (squiggy-positron) is available on PyPI, and the extension is distributed via GitHub releases and Open VSX Registry. Source code, documentation, and example datasets are freely available at https://github.com/rnabioco/squiggy-positron under the MIT license.

Contact: jay.hesselberth@gmail.com

Supplementary Information: User guide and API documentation are available at https://rnabioco.github.io/squiggy-positron/

Keywords: nanopore sequencing, signal visualization, base modifications, IDE extension, interactive visualization, POD5, Positron

1. Introduction

Oxford Nanopore Technologies (ONT) sequencing platforms generate long-read sequences by measuring ionic current changes as DNA or RNA molecules pass through protein nanopores[1]. Unlike other sequencing technologies that only report base calls, nanopore sequencers provide raw electrical signals (commonly called "squiggles") that contain information beyond the primary sequence, including base modifications, secondary structure, and nucleotide kinetics[2]. Visualization and analysis of these raw signals has become increasingly important for quality control, method development, and investigating biological phenomena such as DNA methylation and RNA modifications.

The ONT community has developed several tools for signal visualization, each addressing different use cases. Pod5Viewer[3] provides a standalone GUI application for inspecting

POD5 files, offering basic plotting and export functionality accessible to users without programming experience. Squigualiser[4] generates publication-quality interactive HTML visualizations with signal pileup displays and support for multiple alignment formats, but operates as a command-line tool that produces static outputs separate from the analysis workflow. Tombo[5] offers sophisticated signal analysis capabilities for modification detection but requires complex command-line operations. While these tools serve important purposes, they operate outside the interactive data analysis environment where scientists perform exploratory research. Modern data science increasingly relies on integrated development environments (IDEs) like Jupyter, RStudio, and Positron[6] that combine code execution, visualization, and documentation in a unified workspace. The lack of signal visualization tools that integrate with these environments creates friction in analytical workflows, requiring users to context-switch between standalone GUI applications, command-line tools for figure generation, and their primary analysis environment.

We developed Squiggy to address this gap by embedding nanopore signal visualization directly into Positron IDE, a next-generation data science IDE from Posit. Squiggy extends Positron's capabilities by adding interactive panels for file management, read exploration, and plot customization, while leveraging the active Python kernel for data access. This architecture enables seamless integration with existing Python-based nanopore analysis workflows while providing an intuitive graphical interface for exploration and visualization.

2. Implementation

2.1. Architecture

Squiggy employs a hybrid TypeScript/Python architecture that separates user interface logic from data processing and visualization (Figure 1). The TypeScript extension component integrates with Positron's extension API to provide interactive panels and coordinate user actions. The Python backend handles POD5 file reading, BAM file parsing, signal processing, and Bokeh[7] plot generation. Communication between components occurs through Positron's Runtime API, which enables the extension to execute Python code in the active kernel and retrieve results without subprocess overhead.

This design provides several advantages. First, it maintains a single Python environment rather than requiring a separate backend process, simplifying installation and reducing resource usage. Second, all data loaded by Squiggy remains accessible in the kernel's namespace, enabling users to programmatically interact with the same datasets through notebook code. Third, the architecture supports both GUI-driven workflows (through the extension) and code-driven workflows (through the Python API) using the same backend implementation.

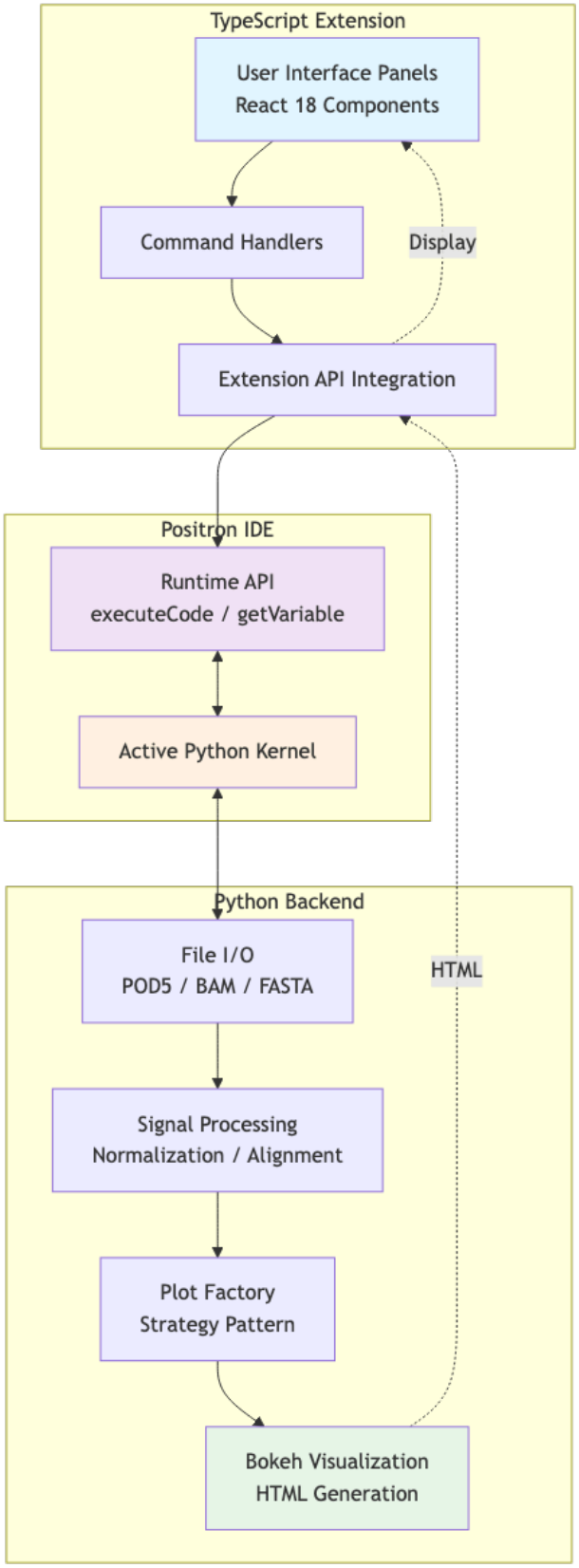

Figure 1: Architecture overview of Squiggy showing the hybrid TypeScript/Python design. The TypeScript extension provides the user interface and integrates with Positron's extension API, while the Python backend handles data processing and Bokeh visualization. Communication occurs through Positron's Runtime API, enabling seamless integration with the active Python kernel.

2.2. File Format Support

Squiggy reads nanopore signal data from POD5 files[8], the current standard format for ONT data storage that supersedes the legacy FAST5 format. POD5 files employ efficient compression and indexing strategies that enable rapid random access to individual read signals. The extension uses the official pod5-file-format Python library to access signal data, read metadata, and calibration parameters.

For base-calling information and genomic alignments, Squiggy supports BAM files generated by ONT basecallers (Guppy, Dorado[9]) and aligners (minimap2[10]). BAM files must be indexed (.bai) for efficient region-based queries. Squiggy extracts alignment information, base calls, quality scores, and the move table that maps signal samples to base positions. When BAM files contain MM/ML tags (following the SAM specification for base modifications[11]), Squiggy automatically enables modification visualization and filtering capabilities.

FASTA reference sequences can optionally be loaded to support reference-based visualization modes and motif searching across genomic contexts.

2.3. Visualization Strategies

Squiggy implements seven distinct visualization modes using the Strategy Pattern, where each plot type is implemented as a separate strategy class that conforms to a common interface. The PlotFactory creates the appropriate strategy based on user selections, and each strategy generates a Bokeh HTML plot customized for its specific use case.

Single-read plots display the raw or normalized signal from an individual read, optionally overlaid with base calls when alignment information is available. This mode serves as the foundation for quality inspection and method validation.

Overlay plots superimpose signals from multiple reads on shared axes using alpha blending to reveal consensus patterns and variation. This mode effectively displays 2-20 reads for pattern comparison.

Stacked plots arrange multiple reads vertically with configurable spacing, similar to Squigualiser's visualization style. Stacking prevents signal overlap and works well for visualizing larger read sets (up to 100 reads).

Event-aligned plots align signals to reference base positions using the move table from basecaller output. This alignment enables direct comparison of signal characteristics at specific genomic positions and forms the basis for modification detection algorithms[11].

Aggregate plots compute summary statistics (mean, standard deviation, percentiles) across many reads aligned to the same reference region. Multiple visualization panels display base modification heatmaps, base composition pileups, dwell time statistics, mean signal with confidence bands, and quality score distributions. This multi-track layout provides comprehensive views of population-level patterns.

Delta comparison plots compute signal differences between two samples at each reference position, highlighting regions with divergent electrical signatures that may indicate differential modification states or structural variants.

Signal overlay comparison plots directly superimpose mean signals from multiple samples on the same axes, facilitating visual comparison of signal characteristics across experimental conditions.

2.4. Signal Processing

Squiggy provides four normalization methods to standardize signals for comparison. Z-score normalization transforms signals to zero mean and unit variance, the most common approach for cross-read comparison. Median normalization adjusts signals relative to the median value, providing robustness to outliers. Median Absolute Deviation (MAD) normalization scales signals by the MAD, offering robust standardization for datasets with extreme values. Users can also visualize raw, unnormalized signals for quality control purposes.

For efficiency with high-sampling-rate data, Squiggy implements optional downsampling that reduces point density while preserving signal shape. The implementation uses Lttb (Largest-Triangle-Three-Buckets) algorithm to identify the most representative subset of points.

2.5. Base Modification Analysis

When BAM files contain base modification information in MM/ML tags, Squiggy provides interactive filtering and visualization capabilities. The Modifications Explorer panel displays all modification types present in the data (e.g., 5mC, 6mA, 5hmC) with user-adjustable probability thresholds. In event-aligned mode, bases are color-coded by modification type and probability. In aggregate mode, modification heatmaps show the frequency and confidence of modifications at each reference position across the read population.

The modification rendering system uses a modular design (ModificationTrackBuilder) that separates data extraction from visualization logic, facilitating addition of new modification types and visualization schemes.

2.6. User Interface

Squiggy's interface consists of multiple coordinated panels in Positron's sidebar (Figure 2). The Setup panel guides initial installation and provides quick access to test datasets. The Samples panel manages multi-sample comparisons, allowing users to load multiple POD5/BAM file pairs and toggle their visibility. The Read Explorer panel displays a virtualized table of all reads in the current POD5 file, grouped by reference sequence when BAM alignments are available. Real-time search filtering enables rapid location of specific reads or genomic regions.

The Plotting panel provides controls for selecting visualization modes, normalization methods, sample selection, and plot-specific parameters. A separate Modifications Explorer panel appears when modification data is available, providing filtering controls organized by modification type. The Motif Explorer panel supports sequence-based filtering for aggregate plots.

All panels employ React 18 for UI rendering, providing type-safe component architecture and efficient updates via virtual DOM diffing. The Read Explorer uses react-window for virtualization, enabling smooth rendering of tables with hundreds of thousands of reads.

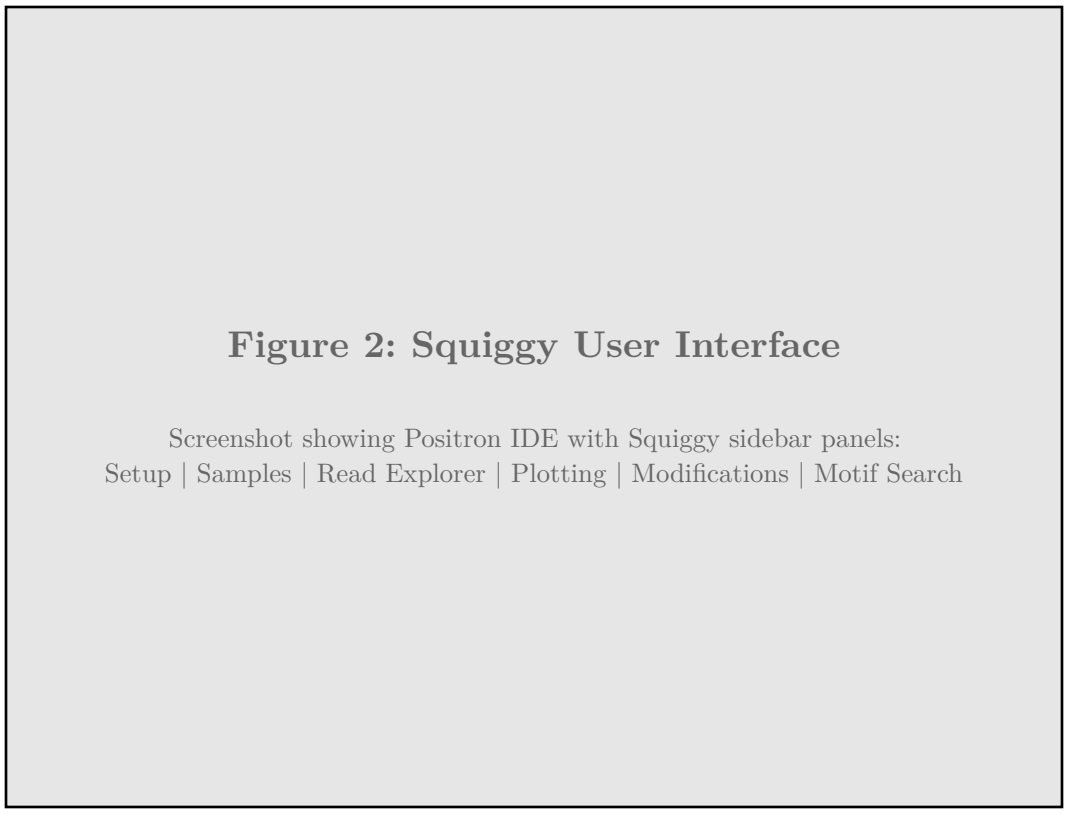

Figure 2: Squiggy user interface integrated into Positron IDE. The sidebar contains multiple coordinated panels: Setup (installation and test data), Samples (multi-sample management), Read Explorer (virtualized read table with search), Plotting (visualization controls), Modifications Explorer (base modification filtering), and Motif Explorer (sequence-based filtering). All panels use React 18 for efficient rendering and real-time interactivity.

2.7. Extensibility

The Strategy Pattern architecture makes Squiggy readily extensible. Adding a new plot type requires implementing a single PlotStrategy class with a `create_plot()` method that returns Bokeh HTML. The new strategy automatically integrates with the existing PlotFactory, UI controls, and export functionality without modifications to other components.

Squiggy's rendering components (ThemeManager, BaseAnnotationRenderer, ModificationTrackBuilder) are designed for reuse across strategies, reducing code duplication and ensuring consistent visual styling. The theme system supports both light and dark color schemes that adapt to Positron's UI theme.

3. Results

3.1. Performance

We evaluated Squiggy's performance using a test dataset of 180 nanopore reads from yeast tRNA sequencing (POD5: 8.2 MB, BAM: 156 KB). File loading times are dominated by POD5 indexing, which completes in under 1 second on a standard laptop (2023 MacBook Pro, M2 processor). Read table rendering uses virtualization to display only visible rows, enabling smooth scrolling through read lists of any size.

Plot generation time scales with the number of reads and selected visualization mode. Single-read plots render in 100-200 ms. Overlay plots with 10 reads render in 200-400 ms.

Aggregate plots sampling 100 reads complete in 1-2 seconds, including signal extraction, alignment, statistical computation, and Bokeh rendering. The Bokeh-based plots provide interactive pan, zoom, and hover functionality while maintaining smooth frame rates.

Memory usage remains modest due to lazy loading. POD5 files are memory-mapped rather than loaded entirely into RAM, and signals are extracted on-demand when generating plots. A session with 10,000 reads loaded consumes approximately 50-100 MB of Python process memory.

3.2. Integration with Positron

Squiggy's integration with Positron's Python kernel enables seamless transitions between GUI and programmatic workflows. Data loaded through the extension appears in the Variables pane (as a `SquiggyKernel` object) and can be accessed directly in both the Python console and Jupyter notebooks:

```python
import squiggy
from squiggy.io import squiggy_kernel

# Session state populated by extension GUI
# Available in both console and notebook
print(squiggy_kernel.read_ids[:5])
# ['read_001', 'read_002', 'read_003', 'read_004', 'read_005']

# Access POD5 reader directly
print(squiggy_kernel.reader)
# <Reader at /path/to/file.pod5>

# View BAM alignment info
print(squiggy_kernel.bam_info)
# {'references': ['chr1', 'chr2'], 'num_reads': 1234}

# Generate custom plot programmatically in notebook
html = squiggy.plot_read(
    'read_001',
    plot_mode='EVENTALIGN',
    normalization='ZNORM'
)
```

This bidirectional integration supports exploratory workflows where users switch fluidly between GUI-based exploration and code-based analysis. Crucially, the same kernel state is shared across the extension UI, the Python console, and Jupyter notebooks within Positron, ensuring consistency and eliminating data duplication. Users can load files through the GUI, inspect them interactively in the console, perform statistical analysis in a notebook, and generate custom visualizations using the Python API—all working with the same underlying data objects.

3.3. Multi-Sample Comparison

Squiggy supports multi-sample comparison workflows through its Samples panel. Users can load multiple POD5/BAM file pairs representing different experimental conditions, toggle sample visibility using eye icons, and generate comparative visualizations. The delta comparison mode computes per-position signal differences between two samples, highlighting regions with divergent electrical signatures. The signal overlay mode superimposes mean signals from multiple samples with distinct colors, facilitating visual comparison of signal characteristics.

For multi-sample aggregate plots, users can choose between overlay style (all samples on one plot) or multi-track style (separate 5-panel tracks for each sample). This flexibility accommodates different analytical goals and presentation preferences.

3.4. Session Management

To support reproducible research, Squiggy implements session save/restore functionality. Sessions capture the current state including loaded files, selected samples, plot parameters, modification filters, and motif search queries. Sessions persist across Positron restarts and can be exported as JSON files for sharing with collaborators or archiving with published analyses.

The demo session feature provides a quick-start capability that loads example datasets and configures recommended visualization settings, serving both as a tutorial for new users and a validation tool for developers.

4. Discussion

Squiggy addresses a gap in the nanopore analysis ecosystem by providing interactive signal visualization integrated directly into a data science IDE. Existing tools occupy different niches in the visualization landscape. Pod5Viewer[3] offers a user-friendly standalone application for file inspection, making raw data accessible to users without programming skills, but lacks integration with analysis workflows and provides limited visualization customization. Squigualiser[4] excels at generating publication-quality signal pileup visualizations with sophisticated alignment handling and multi-dataset comparison, but operates as a command-line tool that generates static HTML outputs separate from the analysis environment. Tombo[5] provides powerful signal analysis for modification detection but requires complex command-line operations and expertise. Earlier tools like poRe[12] provided R-based analysis but are now dated. None of these tools integrate with interactive computational environments where modern data science occurs.

Squiggy's approach of embedding visualization within Positron IDE reduces context switching and enables exploratory workflows where users rapidly iterate between data loading, filtering, visualization, and programmatic analysis. Unlike standalone GUIs, Squiggy maintains direct access to the Python kernel, allowing users to programmatically manipulate loaded data and generate custom visualizations through code. Unlike command-line tools that generate static outputs, Squiggy provides real-time interactive exploration with immediate visual feedback as users adjust parameters. This integration fundamentally changes the workflow from "analyze → export → visualize separately → return to analysis" to a unified "explore → visualize → analyze → iterate" cycle within a single environment.

The dual-interface design (GUI extension plus Python API) provides flexibility for different user preferences and use cases. Researchers can use the GUI for interactive exploration during method development, then transition to the Python API for automated analysis pipelines. The shared backend implementation ensures consistency between interface modes and enables users to extend functionality through custom Python code while still leveraging the GUI for routine tasks.

Squiggy's Strategy Pattern architecture facilitates community extension. The modular design separates plot generation logic from UI and data access concerns, allowing contributors to add new visualization modes by implementing a single strategy class. We anticipate community contributions could extend Squiggy with specialized plot types for RNA secondary structure analysis, ultra-long read visualization, or real-time sequencing monitoring.

Feature	Pod5Viewer	Squigualiser	Tombo	Squiggy
IDE Integration	No	No	No	Yes
Console/Notebook Access	No	No	No	Yes
Interactive GUI	Yes	No	No	Yes
Signal Pileups	No	Yes	No	Yes
Multi-Sample Comparison	No	Yes	Yes	Yes
Base Modification Viz	No	Yes	Yes	Yes
Programming Required	No	Yes	Yes	Optional
Output Format	Plots/Export	HTML	Analysis	Interactive

Table 1: Comparison of nanopore signal visualization tools. Squiggy uniquely combines IDE integration with interactive exploration and programmatic access.

Table 1 summarizes key differences between Squiggy and existing visualization tools. Squiggy uniquely combines IDE integration, interactive exploration, and programmatic access, positioning it as a complementary tool to existing options. Users might employ Pod5Viewer for initial file inspection, Squigualiser for generating publication figures, and Squiggy for exploratory analysis within their computational workflow.

Current limitations include the requirement for Positron IDE, which may not suit users committed to other environments like JupyterLab or RStudio. However, the Python backend (squiggy-positron package) can be used independently in Jupyter notebooks, providing basic plotting functionality outside Positron. Additionally, Squiggy currently supports only POD5 files (not legacy FAST5), though the POD5 format is now standard for ONT data.

Future development directions include implementing additional visualization modes for specific applications (e.g., direct RNA sequencing with splice junction highlighting), enhancing modification analysis with statistical testing between conditions, and adding export capabilities for downstream analysis tools. We also plan to explore integration with other Positron extensions to enable coordinated multi-omics visualization.

The nanopore sequencing field continues to evolve rapidly, with new basecalling algorithms, modification detection methods, and applications emerging regularly. Squiggy's extensible architecture and integration with a modern IDE environment position it to adapt to these developments while maintaining an accessible interface for researchers.

5. Availability and Implementation

Squiggy is implemented as an open-source extension for Positron IDE. The TypeScript extension component (UI and kernel integration) and Python backend (data processing and visualization) are developed together in a single repository.

Software Requirements:

- Positron IDE version 2025.6.0 or later
- Python 3.12 or later
- Python packages: squiggy-positron (via PyPI), including dependencies: pod5 ≥0.3.0, bokeh ≥3.1.1, numpy ≥1.20.0, pysam ≥0.20.0

Installation:

Python package:

```
uv pip install squiggy-positron
```

Extension: Available via GitHub releases (.vsix files) or Open VSX Registry.

Source Code: https://github.com/rnabioco/squiggy-positron (MIT License)

Documentation: https://rnabioco.github.io/squiggy-positron/

Test Data: Example POD5 and BAM files for yeast tRNA sequencing included with Python package

Programming Languages: TypeScript (extension), Python (backend)

License: MIT License

6. Acknowledgments

We thank the Positron development team at Posit for creating the IDE and extension API that enabled this work. We thank Oxford Nanopore Technologies for developing the POD5 format and associated libraries. We acknowledge the developers of Squigualiser[4] for inspiration on visualization design, and the Remora[13] team for modified base calling tools. This work was supported by the RNA Bioscience Initiative at the University of Colorado School of Medicine.

7. Author Contributions

J.H. conceived the project and led software design and implementation. L.K.W. and A.Q. contributed to software development, testing, validation, documentation, and manuscript preparation. All authors reviewed and approved the final manuscript.

References

1. Jain, M., Olsen, H. E., Paten, B. & Akeson, M. The Oxford Nanopore MinION: delivery of nanopore sequencing to the genomics community. *Genome Biology* **17,** 239 (2016).
2. Rang, F. J., Kloosterman, W. P. & Ridder, J. de. From squiggle to basepair: computational approaches for improving nanopore sequencing read accuracy. *Genome Biology* **19,** 90 (2018).
3. Dietrich, V., Alagna, N., Helm, M., Gerber, S. & Butto, T. Pod5Viewer: a GUI for inspecting raw nanopore sequencing data. *Bioinformatics* **40,** (2024).
4. Samarakoon, H. *et al.* Interactive visualization of nanopore sequencing signal data with Squigualiser. *Bioinformatics* **40,** (2024).
5. Stoiber, M. *et al.* De novo identification of DNA modifications enabled by genome-guided nanopore signal processing. *bioRxiv* (2016) doi:10.1101/094672.
6. Posit Team. Positron: A next-generation data science IDE. (2024).
7. Bokeh Development Team. Bokeh: Python library for interactive visualization. (2024).
8. Oxford Nanopore Technologies. POD5 File Format. (2023).
9. ONT Research Team. Dorado: Oxford Nanopore's production basecaller. (2023).
10. Li, H. Minimap2: pairwise alignment for nucleotide sequences. *Bioinformatics* **34,** 3094–3100 (2018).
11. Simpson, J. T. *et al.* Detecting DNA cytosine methylation using nanopore sequencing. *Nature Methods* **14,** 407–410 (2017).

12. Watson, M. *et al.* poRe: an R package for the visualization and analysis of nanopore sequencing data. *Bioinformatics* **31,** 114–115 (2015).

13. Nanopore Technologies Research Team. Remora: Modified base calling for Oxford Nanopore sequencing. (2023).