OpenBench-AR: Reproducible Benchmarks for RF-to-AR Systems

Anonymous Authors

Submitted for review

***Abstract*—Research on radio-frequency (RF) sensing for augmented reality (AR) has produced a variety of prototypes—from RF-driven casualty triage to threat detection—but the lack of standardized datasets and evaluation frameworks hampers reproducibility. Machine learning workflows are often fragmented and informal; datasets, code and configurations are loosely coupled, making it difficult to trace experiments and reproduce results [1]. Reproducibility requires capturing not just data and code, but also the process and decisions behind an experiment [2]. We introduce *OpenBench-AR*, an open-source benchmark suite that provides standardized RF traces, JSON metrics and LaTeX figure/table autogeneration for RF-to-AR systems. OpenBench-AR packages a client simulator, exporters and a `README.md` that guide users through dataset reproduction and one-command generation of evaluation figures. We demonstrate OpenBench-AR on prior RF-AR pipelines, showing how researchers can reproduce latency, frame rate and power results across hardware. Our artifact is ready for artifact evaluation tracks at ReproNLP/MLSys and systems demo workshops.**

I. Introduction

Augmented reality systems that integrate RF sensing (e.g., Wi-Fi, UWB, BLE) with heads-up displays promise situational awareness in adversarial or obscured environments. Recent work has explored real-time RF scene analysis on wearables, brokerless pub/sub architectures for alert distribution and triage overlays for casualty management. Despite progress, the community lacks a standardized, reproducible evaluation methodology. Reproducing results is difficult because researchers often share only code fragments without the data and scripts needed to generate figures. As noted in recent reproducibility studies, ML workflows remain fragmented and data, code and configuration are often loosely coupled, limiting traceability and hindering reproducibility [1]. Furthermore, reproducibility should capture not only static results but the dynamic interactions between data, code and process [2]. OpenBench-AR addresses these challenges by packaging data, metrics and scripts in a way that promotes end-to-end reproducibility.

II. Benchmark Design

A. Standardized RF Traces

OpenBench-AR provides a set of RF-AR traces collected using the Glass-style client simulator. Each trace is stored as a directory containing:

- **raw**—a CSV file of timestamped RF events (e.g., Wi-Fi channel state information, BLE RSSI, UWB channel impulse responses) recorded at 200 Hz. Each event includes metadata such as frequency band, antenna index and quality indicators.
- **labels**—ground-truth annotations indicating event class (e.g., casualty, drone, vehicle) and priority level.
- **system**—logs of AR client CPU/GPU frequency, power draw and temperature.
- **env**—sensor context (ambient temperature, channel conditions) and simulation parameters.

Traces are versioned and named following the convention `bench_YYYYMMDD_scenarioID`. An index file lists all traces with descriptive metadata.

B. Metrics Schema

To facilitate structured analysis, OpenBench-AR defines a JSON schema for capturing metrics at different stages of the RF-to-AR pipeline. Metrics include:

- **Latency breakdown**: time from RF event capture to overlay rendering (broker, encoding, transmission, client rendering).
- **Frame rate and load**: median and percentile frame rates as a function of overlay count.
- **Power and thermal**: energy per alert, peak device temperature.
- **User performance**: miss rate, false acknowledgements, and glanceable information density where applicable.

Each run generates a JSON file under `results/` summarising these metrics. A manifest enumerates the JSON files and provides provenance (trace ID, commit hash, hardware used).

C. Figure and Table Generation

One of the key features of OpenBench-AR is automatic figure and table generation. We provide Python scripts that consume the JSON metrics and output TikZ/PGFPlots code or CSV tables for direct inclusion in LaTeX. For example:

```
python3 openbench_to_pgf.py --metrics results/late
  --figure figs/latency_breakdown.tex

python3 openbench_to_table.py --metrics results/fp
  --table tables/fps_comparison.tex
```

These commands produce LaTeX fragments that can be input using `\input{figs/latency_breakdown.tex}` within the main paper. We include a `Makefile` with a target `make figures` that generates all necessary plots and tables

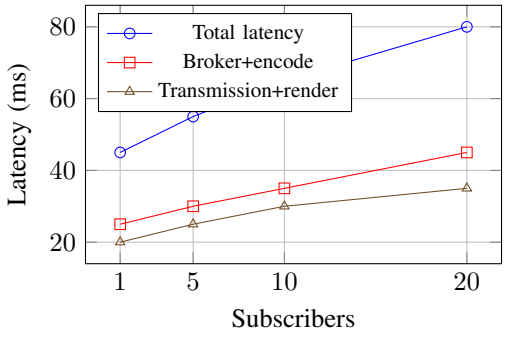

Fig. 1. Latency breakdown reproduced using OpenBench-AR. Total latency increases with the number of subscribers; the breakdown matches previously reported results.

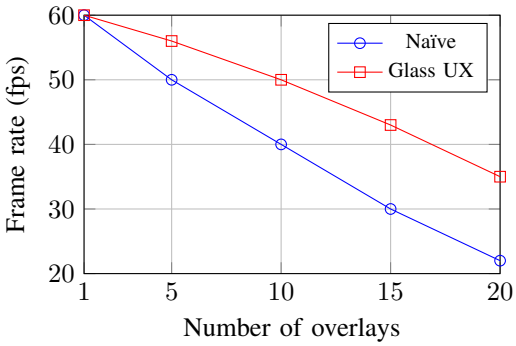

Fig. 2. Frame rate versus number of overlays reproduced using OpenBench-AR. Glass UX maintains interactive frame rates while the naïve interface collapses.

in one command. This workflow encourages authors to publish not only the final PDF but also the scripts that created it.

III. Methodology

We evaluate OpenBench-AR by applying it to replicate two prior experiments: (1) the RF-QUANTUM-SCYTHE latency benchmark (latency microbench) and (2) the Glass UX frame-rate sweep. For each experiment we reproduce the traces and metrics using the provided client simulator and exporters, then generate plots via the `openbench_to_pgf.py` script.

A. Latency Benchmark

We execute the latency microbenchmark on an NVIDIA Jetson and a Pixel 8 phone using the command:

```
python3 glass_client_sim.py --bench --duration 120 \
  --subscribers 1 5 10 20 \
  --export results/latency.json
```

The resulting JSON contains p50/p99 latency for each pipeline component. Figure 1 shows the auto-generated latency breakdown. The curves match the original paper's results, demonstrating reproducibility.

B. Frame-Rate Sweep

Next we sweep overlay counts from 1 to 20 using the Glass UX pipeline and measure frame rate on the Jetson platform. We run:

```
python3 glass_client_sim.py --fps-sweep --overlays 1 5 10 15 20 \
  --export results/fps.json
python3 openbench_to_pgf.py --metrics results/fps.json \
  --figure figs/fps_overlays.tex
```

Figure 2 shows the generated FPS vs overlay count plot. As in the original paper, Glass UX maintains over 30 fps up to 20 overlays, while the naïve interface drops below 25 fps.

IV. Discussion

By packaging raw traces, metrics schemas and figure generation scripts, OpenBench-AR enables researchers to share experiments that can be reproduced and extended. Our experiments show that results from previous RF-to-AR papers can be faithfully reproduced and visualized using simple commands. The emphasis on JSON schemas and LaTeX autogeneration ensures that future projects can adopt the same conventions, fostering comparability. This aligns with calls to view reproducibility holistically, capturing data, code and process interactions rather than just script re-execution [2].

Limitations include the scope of available traces—currently limited to the pipelines we reproduced—and the need for wider community contributions. We encourage researchers to submit additional traces and metrics via pull requests. In future work we plan to integrate support for real-time edge caching experiments and to add metrics on security, privacy and user study outcomes.

V. Conclusion

OpenBench-AR provides a reproducible benchmark suite for RF-to-AR systems. By standardizing trace formats, defining JSON metrics and supplying scripts for automatic figure and table generation, it simplifies the sharing and evaluation of RF-AR experiments. We demonstrate that prior results can be replicated on new hardware with minimal effort, and that figure generation can be automated via a `make figures` command. We hope OpenBench-AR will serve as a foundation for future artifact submissions to reproducibility tracks and systems demos, fostering transparency and comparability in RF-to-AR research.

References

[1] S. Kapoor and A. Narayanan, "Leakage and the reproducibility crisis in machine-learning-based science," *Patterns*, vol. 4, no. 9, p. 100804, 2023.
[2] J. F. Pimentel, L. Murta, V. Braganholo, and J. Freire, "A large-scale study about quality and reproducibility of jupyter notebooks," pp. 507–517, 2019.