ISSN: 2182-2069 / E-ISSN: 2182-2077

MODI: On-Device Preprocessing of Qualcomm Diagnostic Logs for Cellular Protocol Analysis on Android

Vincent Abella[1], I Wayan Adi Juliawan Pawana[2], Dr. Ilsun You[3], and Hoonyong Park[4*]

[1]Kookmin University, Seoul, South Korea. vincent@kookmin.ac.kr, https://orcid.org/0009-0007-2526-1759

[2]Kookmin University, Seoul, South Korea; Udayana University, Badung, Indonesia. adijuliawan@kookmin.ac.kr, https://orcid.org/0009-0006-3767-027X

[3]Kookmin University, Seoul, South Korea. isyou@kookmin.ac.kr, https://orcid.org/0000-0002-0604-3445

[4*]Autocrypt Co. Ltd., Seoul, South Korea. hypark@autocypt.io, https://orcid.org/0009-0009-2802-4198

Received: October 15, 2025; Accepted: January 20, 2026; Published: February 27, 2026

Abstract

Security analysis of cellular signaling protocols, particularly for false base station detection, requires structured Non-Access Stratum/Radio Resource Control (NAS/RRC) datasets captured from the radio interface. Yet existing approaches either depend on kernel-level access increasingly restricted by modern Android Security-Enhanced Linux (SELinux) policies or require a tethered desktop computer. This study presents MODI (Mobile Observation and Diagnostic Interface), an on-device preprocessing pipeline that converts raw Qualcomm Mobile Data Log (QMDL) binary captures into structured Comma-Separated Values (CSV) files entirely on a rooted Android smartphone. The pipeline implements five sequential stages: binary extraction with High-Level Data Link Control (HDLC) unwrapping, protocol dissection via a cross-compiled Wireshark dissector producing desktop-equivalent output, NAS/RRC message filtering, feature extraction from 352 pre-selected protocol fields, and CSV output with hybrid hexadecimal conversion, yielding 1,408 numeric features per packet. The pipeline maintains Android 11+ compatibility by using a vendor-provided diagnostic binary (diag_mdlog) that operates within the existing SELinux policy without kernel modifications. On-device benchmarks on an Android 13 device show that a 100 MB QMDL file is processed in 61–107 seconds with 40–60% CPU utilization and 200–300 MB peak memory, enabling autonomous field collection of structured NAS/RRC data without desktop infrastructure.

Keywords: QMDL Preprocessing, On-Device Protocol Analysis, Cellular Security, Wireshark Cross-Compilation, NAS/RRC Feature Extraction.

1 Introduction

The continued evolution of cellular technology toward 5G and beyond has introduced increasingly complex signaling protocol stacks (Dangi et al., 2021). Security analysis of these protocols, specifically Non-Access Stratum (NAS) and Radio Resource Control (RRC), requires structured protocol-level data captured from the radio interface (3GPP, 2022; 3GPP, 2024). Researchers studying false base station

Journal of Internet Services and Information Security (JISIS), volume: 16, number: 1 (February-2026), pp. 612-623. DOI: *10.58346/JISIS.2026.I1.034*

*Corresponding author: Autocrypt Co. Ltd., Seoul, South Korea.

attacks and protocol exploits need NAS/RRC datasets to develop and evaluate detection mechanisms (Hussain et al., 2018; Yang et al., 2019; Wani et al., 2024). On commercial devices, this data originates from the Qualcomm diagnostic subsystem as binary Qualcomm Mobile Data Log (QMDL) files. However, converting these files into usable datasets has traditionally required desktop tools or privileged access increasingly restricted on modern Android.

Existing tools have several limitations. Mobile Insight (Li et al., 2016) relies on direct /dev/diag access through a native diagnostic binary, blocked on Android 11+ by Security-Enhanced Linux (SELinux) neverallow rules (Android Open-Source Project, 2024). The Signaling Collection and Analysis Tool (SCAT) (Fgsect, 2020) and QCSuper (P1sec, 2019) require a tethered laptop via USB. Desktop tools (QXDM, Wireshark) cannot operate on mobile devices. To the best of our knowledge, no existing tool provides a complete automated pipeline from QMDL collection to structured CSV output entirely on-device.

This study presents MODI (Mobile Observation and Diagnostic Interface), an on-device preprocessing pipeline that converts raw QMDL captures into structured CSV files with NAS/RRC protocol attributes on a rooted Android smartphone. The pipeline uses Qualcomm's vendor-provided diag_mdlog binary to avoid the /dev/diag restriction and a cross-compiled Wireshark dissector (ws_dissector) for protocol analysis producing output equivalent to desktop Wireshark. The contributions are:

- A five-stage on-device preprocessing pipeline converting QMDL binary data into structured NAS/RRC CSV datasets without external hardware or desktop software
- An Android 11+ compatible diagnostic collection strategy using diag_mdlog within the vendor SELinux domain
- On-device benchmarks characterizing latency, CPU, memory, and storage across 20–200 MB QMDL files on production hardware

2 Background

2.1 Qualcomm Diagnostic Logging

Qualcomm-based devices include a diagnostic subsystem that captures modem-level telemetry such as NAS/RRC signaling, radio measurements, and protocol state transitions (3GPP, 2024, 3GPP, 2024). This data is serialized using High-Level Data Link Control (HDLC) framing into binary QMDL files, with 0x7E delimiters, byte-stuffing, and Cyclic Redundancy Check (CRC)-16/X.25 checksums. Timestamps use a 64-bit format (epoch 1980-01-06, upper 48 bits at 1.25 ms resolution).

Two access mechanisms exist. Direct /dev/diag access allows applications to issue ioctl commands for real-time diagnostic streaming; MobileInsight (Li et al., 2016) uses this via a native binary (diag_revealer). diag_mdlog, a Qualcomm vendor binary, runs in a privileged SELinux domain with pre-granted diagnostic access and produces standard QMDL files, functioning on Android 11+ where /dev/diag is blocked.

2.2 Android 11+ SELinux Restrictions

Android 11 has made the use of SELinux neverallow rules that block access to diag_device by untrusted_app processes regardless of root privileges. The diagnostic binary provided by MobileInsight does not work on Android 11 and above with errors of avc: denied; to overcome the error, it is necessary

to rebuild the boot image, which is not feasible on all types of devices. Android 12+ also does not allow running binaries in app data directories, and instead requires explicit SELinux exec_type environments.

2.3 NAS/RRC Protocol Signaling

RRC protocol is used to handle radio bearer setup and mobility between User Equipment (UE) and the radio access network (3GPP, 2022; 3GPP, 2024) whereas NAS is used to do authentication and session management between UE and core network (3GPP, 2024, 3GPP, 2024). For Long-Term Evolution (LTE), the relevant protocols are LTE-RRC (TS 36.331) and NAS Evolved Packet System (NAS-EPS, TS 24.301); for 5G New Radio (NR), they are NR-RRC (TS 38.331) and NAS 5G System (NAS-5GS, TS 24.501). False base station attacks exploit these protocols (Shaik et al., 2016, Hussain et al., 2018), making structured field-level datasets essential for detection research.

2.4 GSMTAP Encapsulation

The GSM Tap Protocol (GSMTAP) provides standardized encapsulation for cellular protocol messages, encoding protocol type, channel type, and direction. Qualcomm-specific protocol identifiers can be mapped to GSMTAP type/subtype fields, enabling protocol dissection through standard tools such as Wireshark.

3 Related Work

Mobile Diagnostic Tools. MobileInsight (Li et al., 2016) accessed */dev/diag* via a native diagnostic binary for real-time analysis, but was incompatible with Android 11+ SELinux policies (Section 2.2) and required per-kernel compilation. The SCAT framework (Fgsect, 2020) parsed Qualcomm, Samsung Shannon, and HiSilicon diagnostics via USB, requiring a tethered laptop. The proposed pipeline adapts SCAT's parsing logic, replacing USB I/O with file-based QMDL processing. Similarly, QCSuper (P1sec, 2019) required a host computer. PHOENIX framework (Echeverria et al., 2021) offered device centric monitoring; however, diagnostic settings were device-specific.

Datasets and protocol Analysis. Desktop tools (QXDM, Wireshark, tshark) were also able to do full dissection but were unable to run on mobile equipment. Hussain et al., 2018 reported adversarial testing of critical LTE protocol vulnerabilities, and based on desktop-captured signaling traces, whereas (Mubasshir et al., 2025) created fake base station detection traces on desktop-only QXDM workflows. Both demonstrated the need for structured NAS/RRC data but depended entirely on desktop infrastructure for data collection.

On-Device Security Systems. The SMDFbs framework (Park et al., 2023) implemented specification -based false base station detection but operated on network-side simulators rather than on the UE. FBS-Radar (Li et al., 2017) and SeaGlass (Ney et al., 2017) detected fake base stations through crowdsourced or vehicle-mounted sensor data, requiring external collection infrastructure. Adjacent domains faced similar data dependency challenges: misbehavior detection for Internet of Things (IoT) devices (Choudhary et al., 2020) and deep learning-based intrusion detection for 5G/6G roaming (Pawana et al., 2025; Won et al., 2026) each relied on structured signaling datasets produced by external toolchains. Surveys on machine learning-based detection for cellular networks (Boualouache & Engel, 2023) further highlighted that these approaches typically assumed the availability of preprocessed structured data. All of these systems depended on an upstream data pipeline but did not address the QMDL-to-CSV preprocessing step that the proposed pipeline provides.

4 Proposed System

4.1 Design Requirements

Five verifiable requirements guide the MODI design:

- **(R1)** On-device operation: Complete QMDL-to-CSV pipeline executes on-device without tethered laptop or desktop software.
- **(R2)** Android 11+ compatibility: Operates within SELinux mandatory access control without kernel modifications or boot image rebuilding.
- **(R3)** Desktop-equivalent dissection: On-device dissector produces output equivalent to desktop Wireshark for NAS/RRC fields.
- **(R4)** Fully automated execution: No manual intervention between QMDL collection and CSV output.
- **(R5)** Reproducible output: Identical QMDL input produces identical CSV output across runs.

4.2 System Architecture

A three-layer architecture is provided in Figure 1, consisting of an Android application layer to coordinate all the activities, a Python processing layer (Chaquopy 15.0.1, Python 3.8), and a native binary layer containing the cross-compiled ws_dissector.

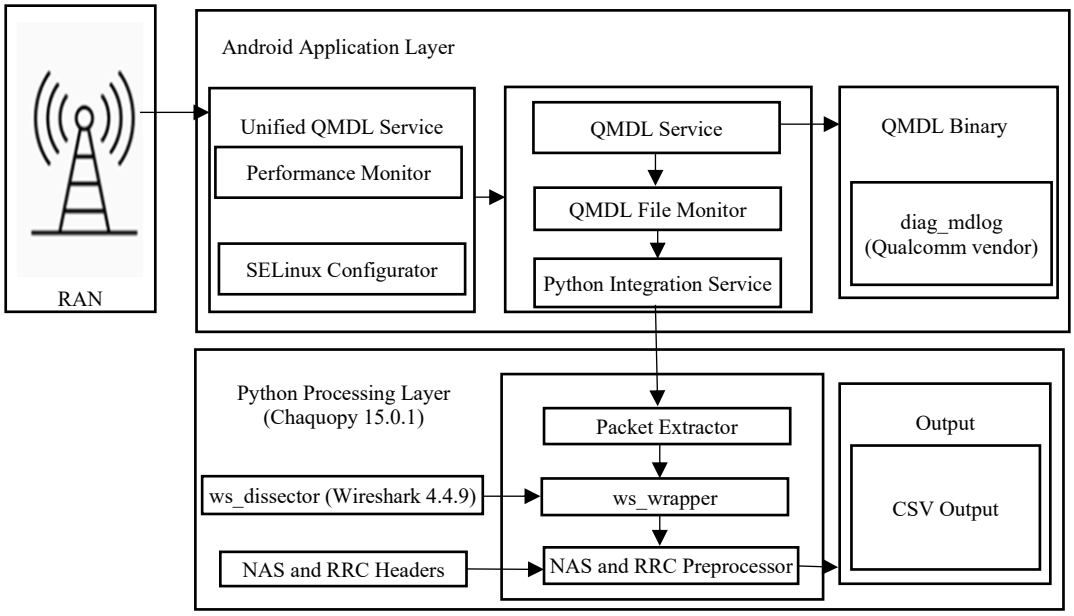

Figure 1: Architecture of MODI system with three layers android application services to coordinate and QMDL collection, python processing modules embedded through chaquopy protocol extraction and feature engineering, native binaries diagnostic logging (diag-mdlog) and protocol dissection (ws-dissector)

Four services are coordinated by the application layer: the UnifiedQmdlService is in charge of the collection-to-analysis workflow; the QmdlService is used to control the diag_mdlog process through the su -c command and the QmdlFileMonitor initiates analysis when the QMDL files reach a customizable

threshold (the default is 100 MB); the Python Integration Service is used to interface Android with the embedded Python environment.

4.3 Diag_mdlog vs. /Dev/Diag: Design Rationale

To meet R2, the pipeline calls diag_mdlog, instead of direct /dev/diag access. Android 11 or later has SELinux rules that are required to prevent access to applications. Policies placed by the vendor on devices based on Qualcomm include:

neverallow untrusted_app diag_device:chr_file { read write open ioctl };

These rules cannot be overridden without rebuilding the boot image. In contrast, diag_mdlog runs in the vendor SELinux domain with pre-granted access: *su -c "diag_mdlog -f <config> -s <size_mb> -n <count>"*. The trade-off is file-based (non-streaming) output, mitigated by QmdlFileMonitor's polling mechanism.

4.4 Five-Stage Preprocessing Pipeline

Figure 2 presents the five-stage preprocessing pipeline that converts raw QMDL binary data into structured CSV output.

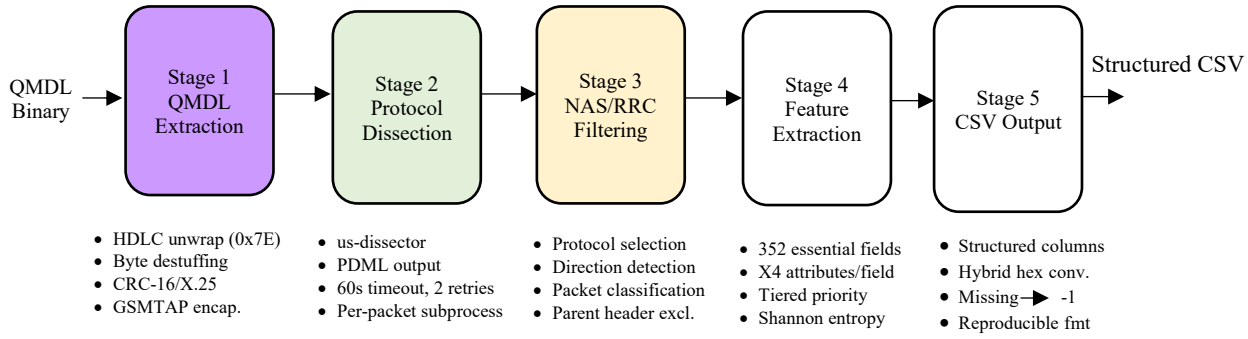

Figure 2: Five-stage preprocessing pipeline. each stage operates sequentially: QMDL binary extraction (stage 1), protocol dissection via embedded ws_dissector (stage 2), NAS/RRC message filtering and classification (stage 3), feature extraction with tiered field priority (stage 4), and structured csv output with hybrid hexadecimal conversion (stage 5)

4.4.1 Stage 1: QMDL Binary Extraction

The extractor of packet contents in QMDL files - The packet_extractor.py module supports raw, gzip and bzip2 files, unwrapping of HDLC frames (0x7E delimiters), destuffing of the bytes and the validation of the CRC-16/X.25. The QualcommParser forwards packets to technology-dependent parsers (LTE, NR, Universal Mobile Telecommunications System (UMTS), Global System for Mobile Communications (GSM)) and wraps each in a GSMTAP envelope, using Qualcomm protocol identifiers as standardized type/subtype values. The Qualcomm 64-bit format is decoded and transcended into the ISO 8601 format using timestamps.

The ws_wrapper.py module calls ws_dissector which is cross-compiled against Wireshark 4.4.9 using Android Native Development Kit (NDK) 27.3 on arm64-v8a. It removes the GSMTAP header, accepts protocol type and payload through the stdin, and accepts Packet Detail Markup Language (PDML) Extensible Markup Language (XML) through the stdout. It extracts the binary out of the assets of the app when it runs and sets the SELinux execution type context of Android 11 and higher. The

packets are split by packet splitting each with a 60-second time out and 2 retries; a failure is registered and the pipeline is not stopped.

4.4.2 Stage 3: NAS/RRC Message Filtering

NasrrCPreprocessor is used to filter the PDML output keeping packets with nas-eps, nas-5gs, lte-rrc or nr-rrc protocol fields. Direction (uplink/downlink) is determined using fields in the header of RRC messages (e.g. DL_DCCH_Message, UL_CCCH_Message), fallback is through the use of GSMTAP header to a standalone NAS packet. Downstream analysis is done by the message type of each packet.

4.4.3 Stage 4: Feature Extraction

The functionality (nas_rrc_headers_headers.py) defines 352 predefined fields that are structured to form two protocol segments that have priority levels. Important attack indicators, including authentication tokens and cause codes, type of security header and key set identifiers, UE capability flags, encryption and integrity algorithm support (EPS Encryption Algorithm (EEA)/ EPS Integrity Algorithm (EIA)/ 5G-EA/ 5G-IA), and bearer parameters are present in the NAS-EPS section (157 fields in 5 levels). RRC section (195 fields in 6 levels) includes LTE cell identity and signal measurements, NR/5G cell parameters, LTE configuration and timers, system information and reselection parameters and UE capability containers, and NR advanced configuration. The fields generate 4 attributes (show, value, size, pos) and give 1,408 columns of feature per packet.

4.4.4 Stage 5: CSV Output with Hybrid Hexadecimal Conversion

A hybrid conversion strategy yields numeric CSV: short hex values (<10 chars) are turned into integers, and long hex values (≥10 chars) are turned into Shannon entropy $(H = -\Sigma_i\, p_i\, log_2(p_i))$, and missing values to −1. The resulting all-numeric format facilitates direct ingestion by downstream analysis tools.

5 Experimental Setup

The QMDL files used for benchmarking were collected from two environments: (1) a laboratory testbed comprising Open5GS, srsRAN, and a LimeSDR Software-Defined Radio (SDR), and (2) commercial LTE/5G networks in South Korea. Table 1 summarizes both environments. The UE device (OnePlus 9 Pro) ran Android 13 (API 33), satisfying the Android 11+ compatibility requirement (R2). The laboratory testbed generated LTE signaling traffic, while the commercial network captures provided real-world NAS/RRC signaling sessions across multiple operators.

Table 1: Experimental environments for QMDL collection and on-device benchmarking

Laboratory Testbed		**On-Device Benchmark**	
Component	**Specification**	**Component**	**Specification**
Core Network	Open5GS	Device	OnePlus 9 Pro
RAN	srsRAN	Chipset	Snapdragon 888
SDR Hardware	LimeSDR	RAM	8 GB
UE Device	OnePlus 9 Pro	Android	13 (API 33)
Android	13 (API 33)	Python	3.8 (Chaquopy 15.0.1)

Pipeline output format. The rows of the CSV generated by the pipeline have 1,408 numeric feature columns, which were obtained based on 352 pre-selected NAS/RRC protocol fields (4 attributes: show, value, size, pos). Some metadata fields are processed like timestamp, direction, packet type and message info to be filtered and classified but not added to the final CSV output. The hybrid hexadecimal

conversion (Stage 5) makes sure that all the feature columns will be numeric, which results in structured output that can be further analyzed using downstream tools. With a 100 MB QMDL file (the default QmdlFileMonitor threshold), the pipeline produces a pipeline that has an average of 191 NAS/RRC packets, 15 MB PDML, 0.8 MB CSV and 116 MB of storage on average. Table 2 shows storage requirements per pipeline stage across representative QMDL file sizes.

Table 2: Storage requirements per pipeline stage

QMDL Size	PDML XML	CSV Output	Total Storage	Approx. Packets
20 MB	~3 MB	~0.2 MB	~23 MB	~38
50 MB	~7 MB	~0.4 MB	~57 MB	~95
100 MB	~15 MB	~0.8 MB	~116 MB	~191
200 MB	~30 MB	~1.6 MB	~231 MB	~383

6 Experiments

The experimental results compared the on-device preprocessing of MODI, which quantified the end-to-end latency, per-stage time, CPU usage, memory usage, and scalability of varying files of QMDL. The main evaluation question was whether the pipeline would be able to transform QMDL binary captures to structured NAS/RRC CSV output under a realistic time and resource constraint on Android 11 and above hardware.

All benchmarks ran on a OnePlus 9 Pro (Snapdragon 888, 8 GB RAM, Android 13, API 33) running MODI v3.0, satisfying the minimum Android 11+ compatibility requirement (R2). The Performance Monitor service sampled CPU utilization via /proc/self/stat and memory via Debug.MemoryInfo at 1-second intervals. Four QMDL file sizes (20, 50, 100, 200 MB) were each processed three times to capture run-to-run variability; reported ranges reflect the minimum and maximum values observed across trials.

6.1 Preprocessing Latency

Table 3 presents per-stage latency across QMDL file sizes. Protocol dissection (Stage 2) dominated at 77–85% of total time due to per-packet ws_dissector subprocess invocations. Stages 3–5 collectively consumed a small fraction of total processing time, decreasing from approximately 12% for 20 MB files to under 7% for 200 MB files, confirming that binary extraction and protocol dissection constituted the computational bottleneck.

Table 3: On-Device processing latency by QMDL file size

QMDL Size	Extraction (s)	Dissection (s)	Filter + Feature (s)	CSV Write (s)	Total (s)
20 MB	2–3	10–20	<1	1–2	13–25
50 MB	4–6	25–45	1–2	2–3	31–54
100 MB	8–12	50–90	2–3	3–5	61–107
200 MB	15–25	100–180	3–5	5–8	120–213

6.2 Resource Utilization

Table 4 presents CPU and memory usage using a 100 MB QMDL file as reference. Peak single-core CPU utilization of 40–60% left sufficient headroom for concurrent device operation (e.g., foreground applications and system services). Peak RAM of 200–300 MB remained within the capacity of Android 11+ devices with 4 GB or more.

Table 4: On-Device resource utilization (100 MB QMDL reference)

Metric	Stage 1–2	Stage 3–5	Overall
Peak CPU (single core)	50–60%	30–40%	40–60%
Peak RAM (MB)	250–300	150–200	200–300
Native heap (MB)	100–150	20–30	100–150
Java heap (MB)	80–100	60–80	80–100

6.3 Scalability Analysis

Figure 3 illustrates the relationship between QMDL file size and processing time. The observed upper-bound processing rate of approximately 1.1 s/MB was consistent across all tested sizes (20–200 MB), indicating approximately linear scalability with respect to input size within the tested range. This predictable linear relationship enables threshold-based configuration of QmdlFileMonitor for automated collection cycles and allows operators to estimate processing time for other QMDL sizes within the tested range on Android 11+ devices with comparable hardware.

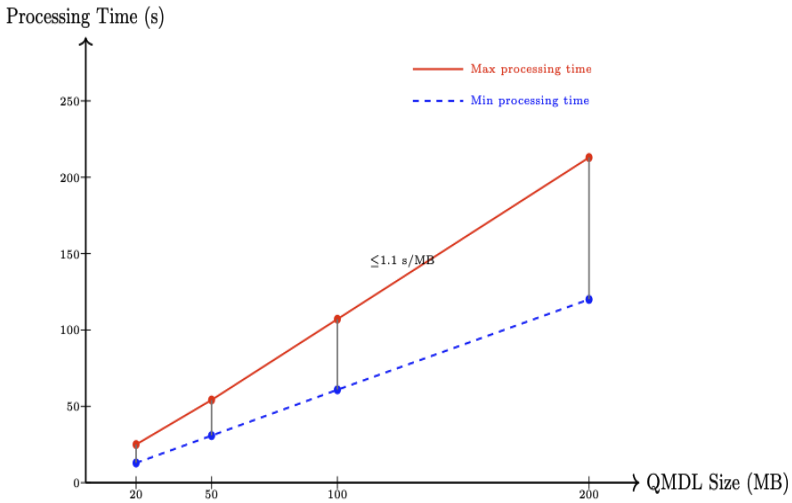

Figure 3: Processing time vs. QMDL file size showing approximately linear scalability. Vertical bars indicate the observed range across three trials. The upper-bound processing rate was approximately 1.1 seconds per MB of QMDL input

6.4 Requirement Verification

Table 5 summarizes the verification of each design requirement against experimental evidence.

Table 5: Requirement verification summary

Req.	Requirement	Verification Evidence	Status
R1	On-device operation	Complete pipeline executed on Android 13; no USB tether or desktop software required	Satisfied
R2	Android 11+ compat.	diag_mdlog in vendor SELinux domain; tested on Android 13 (API 33)	Satisfied
R3	Desktop-equiv. dissection	Cross-compiled Wireshark 4.4.9 PDML output equivalent to desktop tshark -T pdml	Satisfied
R4	Automated pipeline	End-to-end execution via UnifiedQmdlService; no manual intervention	Satisfied
R5	Reproducible output	Identical QMDL input → identical CSV output across runs	Satisfied

7 Discussion

Table 6 compares MODI with existing cellular diagnostic tools across key capabilities. Among the surveyed tools, MODI is the only one that combines on-device operation, Android 11+ SELinux compatibility, and a fully automated QMDL-to-CSV pipeline that produces structured NAS/RRC output without requiring a tethered laptop.

Table 6: Comparison with existing cellular diagnostic tools

Capability	MODI	MobileInsight	SCAT	QCSuper	QXDM
On-device operation	Yes	Yes	No	No	No
Android 11+ compat.	Yes	No	N/A	No	N/A
No tethered laptop	Yes	Yes	No	No	No
Automated pipeline	Yes	Partial	No	No	No
Desktop-equiv. dissection	Yes	Custom	Custom	Partial	Yes
Open source	Yes	Yes	Yes	Yes	No
QMDL → CSV	Yes	No	No	No	No

Preprocessing performance. The Stage 2 bottleneck (77–85% of total time) resulted from per-packet ws_dissector subprocess invocations; batch processing or a persistent daemon mode with stdin/stdout streaming could substantially reduce this overhead. The approximately linear scalability (~1.1 s/MB) enables predictable processing times across QMDL file sizes, and the moderate resource footprint (40–60% CPU, 200–300 MB peak RAM) leaves sufficient headroom for concurrent device operation on Android 11+ hardware.

Design trade-offs. The diag_mdlog method compromised real time streaming support in exchange of Android 11+ support without making any changes to the kernel or rebuilding the boot image. It is a pragmatic decision as the such SELinux neverallow rules cannot be broken in production firmware. Using cross-compiled Wireshark rather than a custom parser ensures desktop-equivalent output fidelity for NAS/RRC dissection and supports protocol updates through Wireshark version upgrades without modifying application logic. The file-based (non-streaming) output model is mitigated by QmdlFileMonitor's threshold-triggered processing, which automates the collection-to-CSV workflow.

Limitations. Several constraints affect the current system:

1. Vendor and chipset dependency: The diag_mdlog binary depends on vendor firmware availability and is specific to Qualcomm chipsets; generalization to Samsung Shannon or MediaTek platforms requires alternative diagnostic interfaces such as SCAT's multi-vendor parsing infrastructure.
2. Root access privilege: Diagnostic collection through diag_mdlog needs root privileges, so that it can be deployed only to rooted Android 11 devices.
3. Stage 2 overhead: The per-packet subprocess model for ws_dissector introduces I/O overhead that dominates overall latency (77–85% of total processing time).
4. Memory limitations: The devices with less than 4GB RAM may be limited by peak memory of 200300 MB during preprocessing.
5. Device generalization: Benchmarks ran on a single device model (OnePlus 9 Pro, Android 13); while the pipeline targets Android 11+ broadly, processing performance may vary across different Qualcomm chipsets, Original Equipment Manufacturer (OEM) firmware variants, and Android versions within the 11+ range.

8 Conclusion

Security analysis of cellular signaling protocols depends on structured NAS/RRC data, yet the primary tool that previously provided this data on mobile devices, MobileInsight, has been rendered incompatible by Android 11+ SELinux restrictions, while the remaining alternatives require tethered desktop infrastructure. This study addressed that gap by presenting MODI, an on-device preprocessing pipeline that converts raw QMDL diagnostic logs into structured NAS/RRC CSV output entirely on an Android 11+ smartphone.

Using diag_mdlog for SELinux-compatible diagnostic collection and cross-compiled Wireshark 4.4.9 for desktop-equivalent protocol dissection, the pipeline processed 100 MB QMDL files in 61–107 s with 40–60% CPU and 200–300 MB peak RAM on an Android 13 device, while targeting compatibility with the broader Android 11+ ecosystem. The resulting CSV output captures 1,408 numeric features per packet across NAS-EPS, LTE-RRC, and NR-RRC protocol fields, producing structured output equivalent to desktop Wireshark without requiring a tethered laptop or external infrastructure. All five design requirements were verified experimentally.

By removing the dependency on desktop tools and circumventing the SELinux restrictions that broke prior on-device solutions such as MobileInsight (Li et al., 2016), the proposed pipeline enables researchers and security practitioners to collect and preprocess cellular protocol data autonomously in the field. This capability represents a necessary step toward field-deployable cellular security monitoring, where structured NAS/RRC data must be produced on-device before any detection logic can be applied. Existing detection approaches, including network-side specification-based systems (Park et al., 2023) and desktop-trained machine learning classifiers (Mubasshir et al., 2025), demonstrate the types of analysis that structured NAS/RRC data supports; adapting such approaches for on-device execution over MODI's output remains an open direction.

Future work targets batch dissection to reduce the Stage 2 bottleneck, chipset generalization via SCAT's (Fgsect, 2020) multi-vendor parsing infrastructure, benchmarking across diverse Android 11+ device models, and integration with downstream analysis frameworks that consume the structured CSV output for on-device false base station detection.

Acknowledgments

This work was supported by the Institute for Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. RS-2024-00437252, Development of anti-sniffing technology for mobile communication and AirGap environments).

References

[1] 3GPP. (2022). Radio Resource Control (RRC) protocol specification. Technical Specification TS 38.331, Release 17. 3rd Generation Partnership Project (3GPP).

[2] 3GPP. (2024). Non-Access-Stratum (NAS) protocol for 5G System (5GS). Technical Specification TS 24.501, Release 18, v18.4.0. 3rd Generation Partnership Project (3GPP).

[3] 3GPP. (2024). Non-Access-Stratum (NAS) protocol for Evolved Packet System (EPS). Technical Specification TS 24.301, Release 18, v18.4.0. 3rd Generation Partnership Project (3GPP).

[4] 3GPP. (2024). Radio Resource Control (RRC) protocol specification (LTE). Technical Specification TS 36.331, Release 18. 3rd Generation Partnership Project (3GPP).

[5] Android Open-Source Project. (2024). Security-Enhanced Linux in Android.

[6] Boualouache, A., & Engel, T. (2023). A survey on machine learning-based misbehavior detection systems for 5g and beyond vehicular networks. *IEEE Communications Surveys & Tutorials*, *25*(2), 1128-1172. https://doi.org/10.1109/COMST.2023.3236448

[7] Choudhary, G., Astillo, P. V., You, I., Yim, K., Chen, R., & Cho, J. H. (2020). Lightweight misbehavior detection management of embedded IoT devices in medical cyber physical systems. *IEEE Transactions on Network and Service Management*, *17*(4), 2496-2510. https://doi.org/10.1109/TNSM.2020.3007535

[8] Dangi, R., Lalwani, P., Choudhary, G., You, I., & Pau, G. (2021). Study and investigation on 5G technology: A systematic review. *Sensors*, *22*(1), 1-32. https://doi.org/10.3390/s22010026

[9] Echeverria, M., Ahmed, Z., Wang, B., Arif, M. F., Hussain, S. R., & Chowdhury, O. (2021). Phoenix: Device-centric cellular network protocol monitoring using runtime verification. https://doi.org/10.48550/arXiv.2101.00328

[10] Fgsect (2020). SCAT: Signaling Collection and Analysis Tool. https://github.com/fgsect/scat

[11] Hussain, S., Chowdhury, O., Mehnaz, S., & Bertino, E. (2018). LTEInspector: A systematic approach for adversarial testing of 4G LTE. In *Network and Distributed Systems Security (NDSS) Symposium 2018*. http://dx.doi.org/10.14722/ndss.2018.23313

[12] Li, Y., Peng, C., Yuan, Z., Li, J., Deng, H., & Wang, T. (2016). Mobileinsight: Extracting and analyzing cellular network information on smartphones. In *Proceedings of the 22nd Annual International Conference on Mobile Computing and Networking* (pp. 202-215). https://doi.org/10.1145/2973750.2973751

[13] Li, Z., Wang, W., Wilson, C., Chen, J., Qian, C., Jung, T., ... & Liu, Y. (2017). FBS-Radar: Uncovering Fake Base Stations at Scale in the Wild. In *NDSS*. 1-17.

[14] Mubasshir, K. S., Karim, I., & Bertino, E. (2025, August). Gotta detect'em all: Fake base station and multi-step attack detection in cellular networks. In *Proceedings of the 34th USENIX Security Symposium*. 419-423.

[15] Ney, P., Smith, I., Cadamuro, G., & Kohno, T. (2017). SeaGlass: Enabling city-wide IMSI-Catcher detection. *Proceedings on Privacy Enhancing Technologies, 2017*(3), 39-56. https://doi.org/10.1515/popets-2017-0027

[16] P1sec (2019). QCSuper: Tool for capturing raw 2G/3G/4G radio frames from Qualcomm-based phones. https://github.com/P1sec/QCSuper

[17] Park, H., Astillo, P. V. B., Ko, Y., Park, Y., Kim, T., & You, I. (2023). Smdfbs: Specification-based misbehavior detection for false base stations. *Sensors*, *23*(23), 1-15. https://doi.org/10.3390/s23239504

[18] Pawana, I. W. A. J., Abella, V., Lastre, J. K., Ko, Y., & You, I. (2025). Enhancing Roaming Security in Cloud-Native 5G Core Network through Deep Learning-Based Intrusion Detection System. *Computer Modeling in Engineering & Sciences*, *145*(2), 2733. https://doi.org/10.32604/cmes.2025.072611

[19] Shaik, A., Borgaonkar, R., Asokan, N., Niemi, V., & Seifert, J.-P. (2016). Practical attacks against privacy and availability in 4G/LTE mobile communication systems. In Proc. Network and Distributed System Security Symposium (NDSS), San Diego, CA, USA. https://doi.org/10.48550/arXiv.1510.07563

[20] Wani, M. S., Rademacher, M., Horstmann, T., & Kretschmer, M. (2024). Security vulnerabilities in 5G non-stand-alone networks: A systematic analysis and attack taxonomy. *Journal of Cybersecurity and Privacy*, *4*(1), 23-40. https://doi.org/10.3390/jcp4010002

[21] Won, T., Kwon, H., Ko, Y., Lastre, J. K., & You, I. (2026). Towards 6G Roaming Security: Experimental Analysis of SUCI-Based DoS, Cost, and NF Stress. *Applied Sciences*, *16*(1), 1-29. https://doi.org/10.3390/app16010508

[22] Yang, H., Bae, S., Son, M., Kim, H., Kim, S. M., & Kim, Y. (2019). Hiding in plain signal: Physical signal overshadowing attack on {LTE}. In *28th USENIX Security Symposium (USENIX Security 19)* (pp. 55-72).

Authors Biography

Vincent Abella received the Bachelor of Science degree in Computer Engineering from the University of San Carlos, Cebu City, Philippines, in 2024. He is currently working as a Master's graduate student at the Department of Cybersecurity, Kookmin University, Seoul, South Korea. His research interests include web API security, 5G/6G networks, and specification-based detection systems.

I Wayan Adi Juliawan Pawana received the S.Kom. degree in computer science from Udayana University, Bali, Indonesia, in 2014 and the M.T. degree in electrical engineering from Udayana University, Bali, Indonesia, in 2021. He is currently working as a Ph.D. graduate student at the Department of Cyber Security, Kookmin University, Seoul, South Korea. His research interests include 5G/6G networks, internet security, AI security, and cloud computing.

Dr. Ilsun You received the MS and PhD degrees from Dankook University, Seoul, Korea, in 1997 and 2002, respectively. He received the second PhD degree from Kyushu University in 2012. Now, he is a full professor at Department of Information Security, Cryptology, and Mathematics, Kookmin University. His research interests include 5/6G security, security for wireless networks & mobile internet, IoT/CPS security, and security protocol design/formal verification. He is included in Stanford-Elsevier's list of the world's top 2% scientists from 2020 to present while achieving 60 H-index based on the google scholar. He is a Fellow of the IET and a Senior member of the IEEE.

Hoonyong Park received the B.S. degree in Information Security Engineering from Soonchunhyang University, Asan, South Korea, in 2019, and the Ph.D. degree through an integrated M.S.-Ph.D. program from the same university in 2023. He is currently a Researcher with Autocrypt Co., Ltd. His research interests include security for V2X communications, software-defined vehicles (SDV), and 5G networks.