ENGINEERING MANUAL / 01

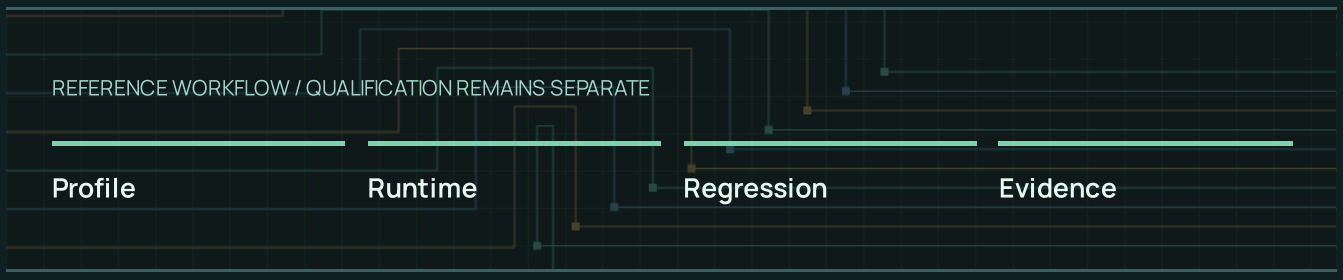
Device Integration & Test Handbook

A practical path from runtime contract to a verified device integration.

1
VM-tested Android port

96
exposed regression inputs

0
qualified physical devices



> Experimental reference. Android VM checks do not establish physical-device support or production quality.

Contents

A field reference for implementation, evaluation and review. Chapter links are clickable in the PDF.

01 Status And Scope 3

02 Verified Android VM Delivery 4

03 Reading Order 4

04 Add A Device Profile 5

05 Common Adapter Contract 5

06 Port The Current Classifier Exactly 6

07 GitHub Build Procedure 7

08 Model Installation And Updates 9

09 Frozen Regression Kit 10

10 Regression Matrix 11

11 Physical-Device Procedure 12

12 Acceptance And Energy 13

13 Bug Reports And Release Review 14

14 Official References 14

Dated evidence and explicit limits are part of the manual. A proposed interface, successful build or matching regression result is not proof of physical-device or production readiness.

Updated October 31, 2026. For integrators, build engineers, testers and release reviewers.

01 Status And Scope

The implemented references are a Node.js/Linux classifier and an experimental native Android APK/AAR, not a universal device runtime. They use pinned Qwen2.5 1.5B q4 plus a reject-only policy. The complete system scored 73/96 (76.0%) with four unsafe acceptances on the synthetic reference. Android matched all 96 exposed reference outputs on an emulator. The >90% target remains unmet. No iOS/watch app, MCU firmware or validated physical-device result exists.

This manual specifies the implementation and qualification path for device-specific adapters. The portable regression kit is an executable scorer, not an inference runtime. The general API/profile examples below remain proposed cross-device contracts; the shipped Node and Android APIs are identified separately.

Device class	Candidate path	Required qualification
Android phone/tablet	Native APK/AAR; x86_64 API 36 emulator verified, ARM64 built only	Actual device/OS compatibility, aggregate memory, quality and physical tests remain open
iPhone/iPad	Native iOS adapter with an approved compatible runtime	Separate signing/distribution, runtime parity and real-device tests
Application-class watch	Platform-specific app and a smaller task model where feasible	Actual OS/SDK support, RAM/storage/thermal budget and foreground lifecycle
Linux embedded board	Headless native/Node adapter on a supported architecture	Runtime binary/kernel support, available memory, storage, power supply and board tests
MCU/RTOS	Deterministic task logic or a purpose-built tiny model	Static arena/flash/stack/operator budget and real-board validation; not the present Qwen model

Do not silently forward work to a phone/server and call it device-local inference. Companion-device processing needs explicit consent and separate connectivity, privacy, latency and energy accounting. No hardware, lab or paid service is assumed available. A device family name is not compatibility evidence.

02 Verified Android VM Delivery

Use the experimental Android release and its `build-manifest.json`, `SHA256SUMS` and VM receipts. Verification 41203062823 tested exact source `ea48bb38df3197d5012f24f7a6a8394e6ee8a11f` with strict dependency checks, three JVM tests, 160 cross-runtime vectors and the full 96-case exposed suite. Integrity, installed permissions, real cancellation/deadline termination, concurrency and foreground UI checks passed.

This is **96/96 reference parity, not 96/96 correctness**. Correctness remains 73/96; accepted precision is 80.3%, with four unsafe acceptances. The API 36 x86_64 emulator used 8 GiB configured RAM. Warm client-round-trip p50/p95 was 13,081/13,704 ms. Aggregate peak memory and whole-device energy were not measured; no phone budget is waived by the VM configuration.

The package contains an APK and reusable AAR without model weights or signing keys. The installed app has no Internet permission. Obtain the exact model files separately, verify their published hashes and import them explicitly into private app storage. Android can reclaim clearable cached storage while allocating an import. Unknown names, incomplete files and mismatched hashes fail rather than select another model.

The Java `InferenceClient` exposes `load`, `classify`, `cancel` and `close` on the main thread. Model work executes in a separate native process; foreground exit, cancellation or deadline stops it and waits for Binder-death confirmation. The C++ bridge avoids a full vocabulary-logit copy into the Java heap. The detailed native API guide covers installation and lifecycle use.

The reusable AAR's `runtimeSourceSha` is not an integrating application's source revision. Record the host `appSourceSha` independently in device reports. ARM64 was compiled only; iOS, watches, MCU/RTOS and physical Linux boards have not been qualified.

03 Reading Order

1. Define the device profile and adapter contract.
2. Implement the native port and GitHub build.
3. Verify artifacts and install on an authorized device.
4. Run frozen regressions, lifecycle/privacy tests and physical measurements.
5. Review quality, resources and release rights separately.

04 Add A Device Profile

Select from measured capabilities, not CPU core count alone: instruction set, available/peak memory, accelerator operators, storage, OS lifecycle, thermal limits, permitted data and task requirements. An application processor and an MCU require different deployment strategies.

Proposed profile example; `null` and unqualified fields are deliberate blockers:

JSON

```
{
  "schemaVersion": 1,
  "profileId": "android-arm64-reference",
  "status": "unqualified",
  "deviceClass": "phone",
  "os": { "name": "Android", "version": null },
  "architecture": "arm64-v8a",
  "runtime": { "adapter": "onnxruntime-mobile", "version": null, "backend": "cpu" },
  "tasks": [{ "id": "service-classification/v1", "modelKey": "qwen-1.5b", "qualified": false }],
  "budgets": {
    "aggregateMemoryBytes": 1610612736,
    "warmP95Ms": 5000,
    "coldP95Ms": 10000,
    "requestDeadlineMs": 20000,
    "loadDeadlineMs": 180000
  },
  "network": { "inference": "deny", "installation": "explicit-consent" },
  "evidence": { "compatibilityReceipt": null, "physicalRunReceipt": null }
}
```

These numbers preserve the original phone research budget. The current cloud process used about 5.42 GiB; its native fit in 1.5 GiB is unresolved. A profile cannot waive that gate. Watch/board/MCU budgets must be independently preregistered, not copied from the phone as proof of suitability.

Add a device by recording its profile, pinning the runtime/toolchain, choosing permitted artifacts, implementing the adapter, running contract tests and attaching physical receipts. Until qualification, return `unavailable` with a reason. Support is per device/OS/model/backend combination. Do not select a larger model automatically when limits fail.

05 Common Adapter Contract

The generic native interface below is a design requirement. The working Node API remains `createServiceClassifier`, `classify` and `close`; the Android client exposes `load`, `classify`, `cancel` and `close`. Do not assume either exports the generic profile API verbatim.

Operation	Required behavior
`capabilities()`	Pinned runtime/backend, supported task IDs, limits and qualification state; no hidden cloud fallback
`load(profile, artifactManifest)`	Check ABI/operators, rights, hashes and memory before allocation; ready only after tokenizer/inference smoke checks
`classify(requestId, text, deadline)`	One bounded foreground request; suggestion, review-required result or explicit failure; no automatic action
`cancel(requestId)`	Stop computation, await termination/release and reject late results; UI cancellation alone is insufficient
`unload()`	Close tensors, sessions, cache and worker resources; verify memory accounting rather than assuming release

Use states `unloaded -> loading -> ready -> running -> ready`, with explicit failed/cancelled/closed transitions. Reject concurrent work rather than accumulating a queue. Cache/KV state is scoped to the model, task, adapter, session and data permission. No always-on coordinator, wake lock or background service just to retain readiness.

Use a bounded C/C++ ABI for MCU/native libraries and thin Kotlin/Swift/Linux wrappers. The schema is shared; allocator and runtime implementation are not. Prefer caller-owned buffers and explicit byte lengths on an MCU. A sensor task needs a separate typed schema/corpus, not the current text-service labels. If the app supplies a task ID, bypass routing. Add a coordinator only after a measured benefit over simpler baselines.

06 Port The Current Classifier Exactly

1. Pin the evaluated ONNX revision, all external weight files, tokenizer, template, examples and policy hashes from the kit. Do not download a floating `main` revision.
2. Match JavaScript string/trim semantics and the 2,000 UTF-16-code-unit limit, then the pinned tokenizer's 512-request-token limit. Test Unicode, emoji, whitespace and JSON escapes; do not silently truncate.
3. Reproduce the reject-only policy, including NFKC normalization. It may return `review` but never a supported category. Record `modelExecuted: false` for this path. These heuristics are not a complete injection defense. The current Node worker loads its model even for policy rejections; an earlier native rejection changes resource behavior and must be reported as a separate scheduling variant.

4. Construct the exact system message, 16 examples and JSON ticket with the pinned chat template. Compare token IDs and prompt bytes on golden fixtures. Another JSON serializer's escaping can change tokenization. Enforce 2,048 total prompt tokens.

5. Inspect actual graph input/output names, shapes, types and optional cache inputs. Do not guess tensor names. Initialize the matching tokenizer and runtime; a raw ONNX session does not automatically implement chat formatting or generation.

6. For one-token classification, run the prefill required by the actual graph, inspect final-position logits, preserve the original scores for four validated distinct A/B/C/D tokens, mask other tokens and select greedily. Map them to maintenance/delivery/inspection/review. Expected answers never enter the prompt or runtime.

7. Preserve request ID, raw choice, canonical output, policy/model path, model revision, timings, errors and human-review requirement. Confidence remains `null`; logits are not calibrated confidence.

8. Compare all exposed parity cases with the cloud reference. Changed quantization/tokenizer/backend behavior is a separately fingerprinted experimental arm, not an automatically equivalent port.

A Java-only adapter can use `OrtEnvironment` / `OrtSession`, but copying the complete logits tensor into the Java heap is expensive. The supplied port instead uses the Android ONNX C++ API behind a small JNI bridge, inspects graph names/shapes/dtypes and returns only four choice scores. It uses pinned Android libraries, not a desktop binary. Start with CPU compatibility; GPU/NNAPI/NPU requires separate operator/fallback and memory evidence.

Keep independent validated control logic in charge of safety-critical electronics. This classifier must not directly actuate machinery, change medical treatment or authorize financial/security actions.

07 GitHub Build Procedure

Existing Workflow

The workflow retains `unit`, `publish`, `live` and `ai` modes and adds `android-prepare`, `android` and `android-media`. `android-prepare` generates dependency verification/locks and builds the APK/AAR without device inference. `android` replays pins strictly and executes the native emulator suite. `android-media` captures the exact already-verified APK without rebuilding it or repeating the 96-case corpus. Native jobs have read-only repository permissions and do not publish the website.

Use `unit` for report-contract checks, `publish` with `experiment=false` for SDK/manual/test-kit generation, and `live` with the exact publication source SHA for hosted acceptance. `ai` remains the frozen Node evaluator and is unnecessary for manual/package work.

All installation, builds, automated checks, rendering and report processing for this project run on GitHub. Physical inference must run on the target device. A legitimate fork must deliberately adapt its own repository-specific cloud guard; never spoof GitHub environment variables locally.

Native Android Implementation

The native project is implemented under `native/android`, with an application and reusable runtime module. These are its reproduction and qualification steps, not physical-device acceptance.

1. Separate application UI, task contract and runtime adapter into owned packages/modules. Include unit and instrumentation tests. Commit the Gradle wrapper with its distribution SHA-256 and dependency verification/locks.
2. The current baseline uses JDK 17, Gradle 9.6.0, AGP 9.4.1, SDK 36, build tools 36.0.0, NDK 28.2.13676358, CMake 3.22.1 and ONNX Runtime Android 1.31.1. Java and C++ are used; no Kotlin implementation is required. API 37 was unavailable in the SDK registry used by the attempted setup. Rolling upgrade suggestions remain visible as informational lint findings; code, privacy and resource diagnostics remain blocking.
3. Add a manual-dispatch native workflow on an authorized standard GitHub-hosted Linux runner. Build/test with read-only permissions. Separate publication/signing jobs; do not expose credentials to inference or untrusted PR code. Do not enable a paid runner or schedule.
4. Execute the real module tasks on that runner, for example:

```sh
./gradlew --no-daemon :runtime:testDebugUnitTest :runtime:lintDebug :app:lintDebug
./gradlew --no-daemon :runtime:assembleDebug :app:assembleDebug
./gradlew --no-daemon :app:assembleDebugAndroidTest
```

5. Run `connectedDebugAndroidTest` only with a configured, authorized emulator/device attached to the runner. The supplied VM script rejects physical, offline or ambiguous ADB targets and retains the test APKs until private reports/screenshots are collected. A hosted job cannot directly access a phone connected to a separate workstation. Emulators test plumbing, not physical energy/thermal/RAM behavior. A remote device lab requires separately authorized access.
6. After required gates pass, publish the exact APK, reports, dependency inventory, build manifest, SHA-256 and signer fingerprint. Verify the signed artifact on the runner:

```sh
apksigner verify --verbose --print-certs app-debug.apk
sha256sum app-debug.apk
```

Use the actual output paths. An APK is installable; an AAB is not directly installable. Ephemeral debug certificates can change between runners, preventing in-place upgrades; they are not production signing. Preserve test data before an authorized reinstall. Never uninstall/wipe automatically. Release signing needs approved secret management; never commit keystores/passwords or paste them into chat.

Other Device Builds

Target	Procedure	Required artifact/evidence
Linux board	Pin sysroot/toolchain/runtime; cross-compile on GitHub if needed; package adapter and model manifest; test matching architecture	Versioned executable/package, dependency hashes and board-side lifecycle/memory results
Watch	Create the platform app/extension, reuse schema, choose a watch-qualified task/model and enforce OS lifecycle rules	Signed platform artifact and physical-watch results independent of the paired phone
MCU/RTOS	Select supported operations and bounded sensor model; convert on GitHub; compile constant weights and a measured tensor arena; test fixed-shape I/O/watchdog	ELF/BIN/HEX, map file, flash/RAM/stack budgets and real-board results; no Node/Qwen claim
iOS	Pin runtime/SDK; implement Swift/native bridge; establish approved signing/distribution and remote build access	Separately signed device artifact and iPhone qualification; no Apple membership or macOS runner assumed approved

LiteRT for Microcontrollers is a candidate for tiny tasks, not a conversion button for this Qwen model. Its limited operations, manual memory management and lack of on-device training must fit the task. Reuse an established runtime instead of implementing a new inference engine.

08 Model Installation And Updates

1. Separate installation networking from inference. Obtain explicit consent and show model source, license, size and storage requirements. Never bundle credentials/private data.
2. Download fixed reviewed artifacts into staging. Check every byte count/hash and preserve external ONNX data files. Reject interrupted/partial files before activation.
3. Run compatibility and known smoke checks, then atomically activate. Keep the prior verified model until activation succeeds.

4. On failure, return a documented error and retain the prior version. No silent model/backend substitution or cloud fallback. Model/template/policy changes require a new protocol.
5. Provide explicit model removal without deleting unrelated reports/app data. Keep inputs in private storage and out of routine logs.

09 Frozen Regression Kit

The cloud build generates `downloads/pocketroute-device-test-kit-0.1.0.zip`, with no model weights or inference. Use matching files from one SHA-256-verified publication:

File in kit	Purpose
`protocol.json`	Model, policy, prompt, examples, limits and evaluated artifact fingerprints
`requests.json`	The 96 exposed synthetic input cases, without expected answers; only this side goes to the device
`reference.json`	Expected labels and cloud predictions, outside the inference input
`report-template.json`	Deliberately incomplete output; `not-run` /null timings fail validation until attempts exist
`validate-device-report.mjs`, `device-report.mjs`, `contract.mjs`	Dependency-free Node scorer; no runtime installation/download
`review-policy.mjs`	Reference reject-only policy for native conformance
`MANUAL.pdf`, `LICENSE`	Designed integration/test handbook and original-source license

Do not substitute newer expected answers or drop difficult cases. Both existing corpora are exposed: regression/parity only, not a fresh accuracy claim. This text-service kit cannot validate an MCU sensor task; that requires a separately versioned suite and scorer.

Test Reports

Fill the template with actual app source SHA, device class/model/OS/architecture, runtime/version/backend, execution kind (`physical-device`, `simulator` or `cloud-fixture`) and phase (`cold` or `warm`). No serial numbers, account IDs, customer text or tokens. Each original case ID needs one outcome, output, total elapsed milliseconds and peak-memory bytes or `null` when unmeasured. Timeout/cancel/error/unavailable attempts use `output: null`; never delete the row.

From the extracted kit, on GitHub or another separately authorized test environment:

```sh
node validate-device-report.mjs reference.json report.json parity
node validate-device-report.mjs reference.json report.json non-regression
```

For this workspace, score on GitHub, not the Mac. Add the authorized synthetic-only report to a dedicated cloud job or manually dispatched validation step; the existing workflow has no report-upload input. Never publicly upload private traces or customer reports.

Exit `0` means the selected parity/non-regression comparison passed, `1` means a recorded behavioral regression, and `2` means malformed/incomplete evidence. Exact parity checks every output. Non-regression checks accuracy counts, accepted precision, supported coverage, unsafe acceptances and invalid/failed attempts. The output also reports quality separately. **Matching the failing 76% reference is not release acceptance.** The scorer cannot attest actual hardware execution and never sets `productionReady` to true.

Keep cold and warm runs separate. Preserve every repeat and preregister aggregation instead of choosing the best run. The scorer reports descriptive p50/p95 and peak memory; it does not certify energy, independent hardware evidence, thermal acceptance or population accuracy.

10 Regression Matrix

ID	Test	Required observation
C01	Empty/whitespace, 2,001 UTF-16 units, 513+ tokens, Unicode/JSON escapes	Matching validation, no silent truncation
C02	Golden prompt bytes, token IDs, A/B/C/D IDs	Exact pinned semantics; no answers in input
C03	Invalid/NaN logits, ties, wrong-but-valid labels, no output	Fail closed where required; wrong labels remain errors
C04	Override/secret/code/weather queries plus legitimate printer/weather-damage cases	Policy never assigns a supported category; false rejections counted

ID	Test	Required observation
C05	Every frozen request in each declared phase	All IDs present; differences/errors retained; parity distinct from quality
L01	Busy request, active cancellation, immediate retry	No unbounded queue/stale result; actual computation stops
L02	Background/resume, close/reopen, OS process termination	No unauthorized background inference; explicit restart
L03	Low memory/storage, unsupported ABI/operator, sustained heat	Controlled error/unavailable, no hidden backend switch
S01	Corrupt/wrong-hash/truncated model; interrupted update	Rejected before activation, prior verified version retained
S02	Offline inference, restored connectivity, exports/logs	No unauthorized egress; airplane mode is not a complete network audit
P01	Cold/warm, sparse/burst load, 15-minute condition	Load cost, p50/p95, aggregate memory, failures and thermal state recorded
R01	Missing/duplicate IDs, changed hashes, placeholders/null values	Incomplete evidence rejected; unknown is not zero

Every fixed defect gets a regression fixture. Fixing an exposed example is development, not new independent quality evidence. Native crashes, tokenizer differences, privacy violations and unsupported operators block that profile even if aggregate accuracy improves.

11 Physical-Device Procedure

1. Record an owned/authorized device's model, SoC, RAM, free storage, OS and battery/thermal state. No serial/account identifiers or purchases assumed.
2. Download the identified GitHub-built test artifact; verify its published hash and signer through a trusted method. Use normal authorized installation, never root/bypass warnings. Undo temporary installer/debugging permissions after testing.
3. Explicitly approve the pinned model download on Wi-Fi. Verify readiness, hashes and memory. Stop on incompatibility rather than choosing an unrecorded fallback.

4. Enable airplane mode and explicitly disable Wi-Fi/mobile data. Restart and run a synthetic request. Offline success establishes an offline path, not proof of no egress when connectivity returns; inspect runtime policy/traces separately.

5. Run the frozen suite, then cancellation, concurrency, restart and lifecycle tests. Export only authorized synthetic evidence for GitHub processing.

6. Measure fresh-process/model-load cold runs with filesystem-cache state disclosed, and warm runs with the model resident. Record total completion time, all failures, peak process plus non-overlapping accelerator allocations and pressure/thermal state.

7. Test a 15-minute sustained condition and representative sparse/burst sessions. Do not maintain background inference merely to finish a test. Repeat on a second SoC before the relevant broad portability claim.

Opening the public dashboard/Notes app on a phone tests UI and rule-based Notes functionality only. It does not execute this SLM. Android-emulator and GitHub-CPU results are not physical-phone or watch measurements.

12 Acceptance And Energy

- Quality target: strictly above 90% on a new declared untouched confirmation set, accepted precision at least 95%, supported coverage at least 80%, zero unsafe acceptances. Synthetic success alone is not customer/population validation.
- Original phone memory budget: at most 1.5 GiB aggregate resident app/router/workers plus non-overlapping accelerator allocations, shared buffers counted once with the method documented; OS pressure limits also apply.
- Classification/extraction p95: at most 5 seconds warm/10 seconds cold. Declare separate hard request deadlines before testing.
- No OOM/crash in the defined repeated suite, bounded cancellation and no unauthorized actions/egress. Missing instrumentation is blocked/inconclusive, not passing.
- Routing energy claim: lower 95% confidence bound of validated whole-device session-energy savings per accepted correct task at least 20% against the best eligible simpler baseline, with quality/coverage intact. Include loading, switching, errors and declared idle intervals. Zero accepted tasks is undefined, never a free efficiency win.

Pixel 6 and later supported Pixel devices expose ODPM rails. These are system-wide, not app-isolated and not automatically complete whole-device energy. Rails may omit/overlap components. Rail-only diagnostics cannot pass the whole-device gate. Other devices need validated counters or calibrated authorized measurement. Battery percentage, CPU utilization, emulators and an ordinary USB charging meter alone do not prove energy savings. No measurement equipment has been obtained.

The existing cloud memory result cannot waive a mobile budget. Watch/MCU targets need independent task-appropriate budgets and methods. iOS requires a separately approved compatible harness, signing/distribution and physical device; no Apple membership, TestFlight access or paid subscription is assumed.

13 Bug Reports And Release Review

Record profile ID, app/model/runtime/protocol hashes, non-identifying OS/SoC details, synthetic case ID, expected/observed behavior, reproduction steps, cold/warm state, first failing version, frequency, sanitized logs, severity and linked receipt. Preserve failures and attach fix/retest evidence. Do not post customer prompts in public issues.

Sign off contract compatibility, complete regressions, new independent quality evidence, physical performance/lifecycle/privacy evidence, licenses and signing/update rollback separately. A build pass, valid signature, emulator pass and exact cloud parity each answer different questions; none alone proves readiness.

14 Official References

- ONNX Runtime Mobile: https://onnxruntime.ai/docs/get-started/with-mobile.html
- Java session/tensor API: https://onnxruntime.ai/docs/get-started/with-java.html
- Android command-line builds: https://developer.android.com/build/building-cmdline
- APK signature verification: https://developer.android.com/tools/apksigner
- LiteRT Micro requirements/limits: https://developers.google.com/edge/litert/microcontrollers/overview
- Android power profiling: https://developer.android.com/studio/profile/power-profiler
- Macrobenchmark energy scope: https://developer.android.com/topic/performance/benchmarking/macrobenchmark-metrics#power

Sources reviewed October 30, 2026. Exact target compatibility still requires a recorded test.