TeachScribe: On-Device Multimodal Photo Retrieval

Public Engineering Case Study

Public engineering case study by Alberto Reyes Castro

Evaluated on a Huawei MediaPad M3 Lite 8 (CPN-W09), Android 7 / API 24



Application	TeachScribe
Engineering focus	On-device multimodal photo retrieval
Model	TinyCLIP 39M/16; INT8 image and text encoders
Benchmark artifact	ARM64 Release APK; 169.25 MiB
Later packaging closure	143.01 MiB; September 2026
Evidence period	September 2026
Public case study	October 2026; architecture and de-identified numeric evidence

Performance, architecture, persistence and workflow reference

Original technical charts retained. Application source code, private photos, gallery screenshots, personal records, credentials and internal delivery material excluded.

This is an engineering case study, not an open-source application release.

Executive overview

Production status. The on-device pipeline described in this report was integrated into the TeachScribe app and observation workflow and has shipped within TeachScribe's deployment to more than 50 schools across the UK. That figure is deployment reach; adoption of the feature itself has not been measured yet, and production telemetry will be reported once a representative usage window is available. The audited engineering repository contains a post-delivery, behaviour-preserving review refactor of the delivered source, validated against it, plus fail-closed guards that only affect invalid (non-finite or wrong-dimension) embeddings; that review build is not the deployed snapshot itself. The measurements in this report come from controlled pre-production evaluation: device measurements on one tablet, plus explicitly scoped desktop model-selection, quantisation and FP32–INT8 comparisons.

The engineering problem was to retrieve relevant local photographs from a natural-language observation on constrained Android hardware, while preserving human control over the final decision. The work connects vision–language model evaluation to deployment: tokenisation, quantised inference, native preprocessing, persistent vector storage, responsive ranking and reliable application state.

The resulting prototype uses TinyCLIP 39M/16 with separate INT8 image and text encoders. A TypeScript tokenizer and Kotlin preprocessing module feed ONNX Runtime. Image vectors persist in SQLite, so a cached search avoids repeated image inference. Users review ranked suggestions and explicitly confirm photos or no match. Successful-search history is independent of reviewed observations.

Result	Evidence and scope
441 ms (P95 459 ms)	108 consecutive cached searches across 44 candidates on one Android 7 ARM64 tablet. The timer excludes final rendering and record persistence.
74.77% reduction	Text encoder raw file size: 178,520,196 to 45,042,123 bytes after dynamic INT8 quantisation.
17/20 Hit@1	TinyCLIP 39M/16 on a small curated desktop comparison, with 16 candidate images. This is not a population accuracy estimate.
4.225 s cold example	One search with 43 cached images and one new image, including cold image-model acquisition.
Persistent workflow	Reusable image embeddings, automatic history, independent human reviews and tested local-calendar date semantics.

Contribution

The project contribution is applied ML systems engineering: comparison of five pretrained models; integration of a selected model into a mobile application; measured optimisation across quantisation, image decoding and scheduling; and end-to-end workflow reliability. It is not the invention of TinyCLIP or a claim to have trained a new foundation model.

AI-assisted portfolio production

The public case-study materials were prepared with AI assistance for editing, presentation and media production. The narrated technical video uses an ElevenLabs clone of Alberto Reyes Castro's own voice. Technical claims remain grounded in the audited engineering repository and the de-identified evidence published with this case study.

Engineering judgement

The selected model was neither the smallest nor the fastest. A faster ranking policy was rejected after UI concerns; image-model preloading was rejected because it added approximately 90 MiB idle process memory. These trade-offs matter as much as the headline latency.

Scope note. Four evidence stages remain separate: desktop model selection; historical Android performance measurements; later functional acceptance; and post-delivery checks of this edition. The post-delivery review build changes display/build identity and removes private material. It was source-checked in the audited engineering repository. A release build of the refactored code was installed on the same tablet for a functional A/B against the delivered build; latency, memory, thermal and energy figures were not re-measured and belong to the historical benchmark build.

Contents

Executive overview **1**

1 Evidence, methodology and scope **4**

2 Local retrieval architecture **5**

3 Inference contracts and cache lifecycle **6**

4 User workflow and durable application state **7**

5 Temporal selection and calendar correctness **8**

6 Model selection: five candidates **9**

7 Retrieval quality, selection and failure analysis **10**

8 Quantisation and scheduler optimisation **11**

9 Native preprocessing and rejected preloading **12**

10 Warm and sustained search performance **15**

11 Cold paths and startup milestones **16**

12 Memory and stability **17**

13 Battery-side energy and thermal observations **18**

14 Packaging, runtime identity and validation **19**

15 Temporal acceptance: fixed-target case study **20**

16 Public evidence and distribution **22**

17 Privacy, limitations and next engineering work **23**

18 Public case-study boundary **24**

References **25**

1 Evidence, methodology and scope

This report is a public portfolio derivative of a completed engineering study [7]. It retains relevant numerical results and system contracts while excluding application source code, original images, linked query records, screenshots, raw device dumps and internal delivery correspondence. The published data supports numerical inspection, not an independent replay of the private image evaluation.

Stage	Workload	Interpretation
Desktop selection	Five models; Hard set of 20 queries and 16 images. Two models also evaluated on Expanded and New Images sets.	Curated retrieval comparison on a Windows Intel i7-13620H CPU without CUDA; representative timing runs.
Android benchmark	Physical Huawei MediaPad M3 Lite 8 (CPN-W09), Android 7 / API 24, ARM64. Small candidate pools, repeated cached searches and selected cold cases.	Release-mode deployment measurements; one device, no large-library or cross-device generalisation.
Functional acceptance	Calendar filtering, window widening, cross-day target inclusion, persistence, reopen and rerun.	Case-level acceptance of the later application build; performance study not repeated.
Post-delivery review	Sanitised review build and documentation; behaviour-preserving refactor; post-delivery FP32–INT8 retrieval comparison.	Private source checks and privacy audit; functional rebuild and install for the A/B equivalence check; no new device performance benchmark.

1.1 Metric definitions

For each query q, let G_q be its labelled acceptable images and $R_k(q)$ its first k ranked results. The historical metric labelled Recall@k is more precisely:

$$\text{Hit@}\,k = \frac{1}{|Q|} \sum_{q \in Q} \mathbf{1}\{R_k(q) \cap G_q \neq \varnothing\}. \tag{1}$$

It measures the fraction of queries with at least one correct hit, not the fraction of all relevant images returned. This report consistently uses Hit@k.

1.2 Timing and system measurements

Search phases use a monotonic JavaScript timer. The finish boundary precedes metric/history writes and final UI rendering. Image read, preprocessing and inference are nested within lookup/ranking; session timings are nested within broader encoding phases. These intervals are nested and are therefore not additive. Startup uses process and React Native log milestones, not first-frame or fully-interactive timing.

Memory is Android process PSS and Native Heap PSS from `dumpsys meminfo`. Battery-side power uses device voltage/current samples with fixed screen conditions and USB disconnected. These are practical engineering measurements, not isolated accelerator measurements or calibrated external power-analyser results. Candidate pools grew during development, so comparisons across stages are directional unless matched explicitly.

2 Local retrieval architecture

React Native and Expo own navigation, permissions, input and orchestration [5]. MediaLibrary enumerates accessible assets. The query tokenizer and text encoder produce a 512-dimensional vector. Image vectors come from RAM, persistent SQLite storage or, on a cache miss, the Kotlin/ONNX image path. Cosine similarity ranks candidates:

$$s(\mathbf{t}, \mathbf{i}) = \frac{\mathbf{t}^{\mathsf{T}}\mathbf{i}}{\|\mathbf{t}\|_2 \|\mathbf{i}\|_2}. \tag{2}$$

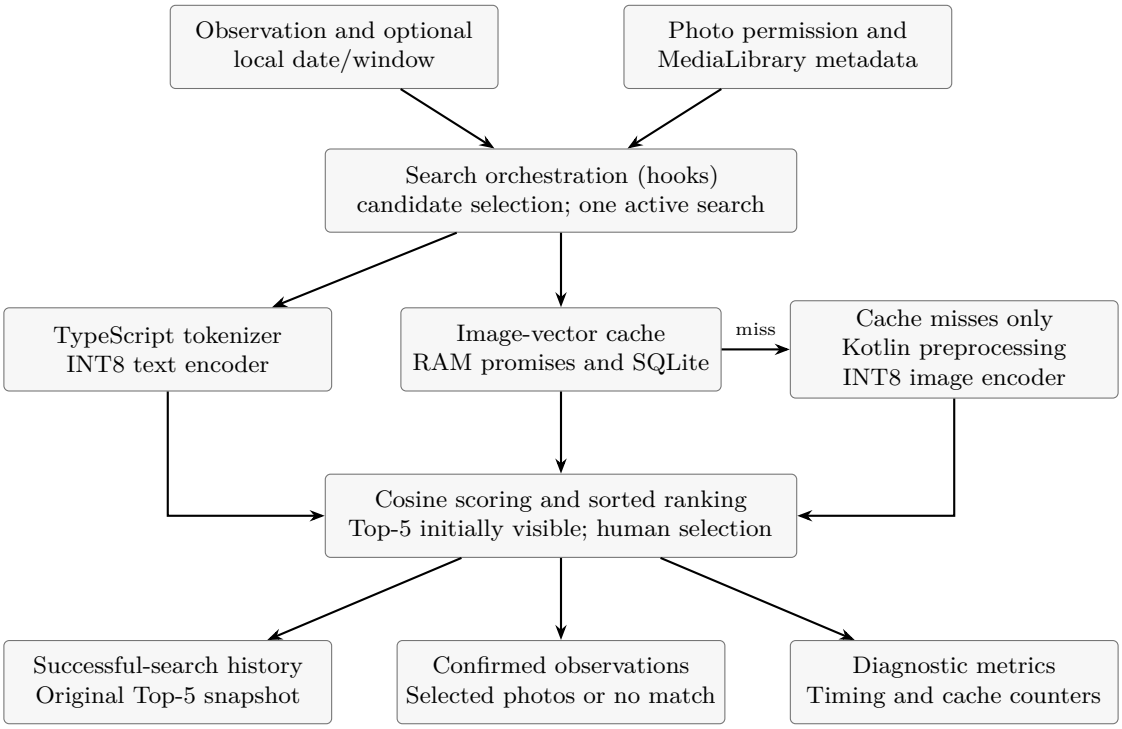


Figure 1: Application responsibilities and data flow. All boxes are local subsystems. Image inference occurs only for missing valid vectors; its result is cached. Diagnostic recording also covers cancelled and failed searches.

Figure 1 shows the local data flow. A search snapshots its query and candidate list. One active search is permitted. Shared lazy promises coalesce session acquisition and image-vector work. Query vectors are recomputed for each search, while valid image vectors are reused. Results are sorted descending and the interface initially presents five suggestions, with explicit human selection or no-match review.

2.1 Cache architecture and rationale

For a fully cached library, the image encoder can remain unloaded. Warm cost is then text encoding, metadata/cache access, scoring, sorting and UI scheduling. The cache is therefore central to the deployment design, rather than an incidental speed improvement. A cold or expanded library has a different cost profile and must be reported separately.

Persistence, tokenisation and calendar parsing were already separate modules at delivery. After delivery, the 3,749-line orchestration screen was split, without behavioural change, into domain hooks, presentational components and React-free search and inference modules (Section 14.1).

3 Inference contracts and cache lifecycle

Boundary	Contract
Text input	`token_ids`, int32 $[1, 77]$; CLIP byte/BPE vocabulary, start/end tokens and padding.
Text output	`text_embedding`, 512 components.
Image input	`pixel_values`, float32 $[1, 3, 224, 224]$.
Image output	`image_embedding`, 512 components.
Native bridge	150,528 Float32 values; 602,112 little-endian bytes. Size/shape are checked and copied to an aligned JavaScript buffer.
Cache vector	512 Float32 values, 2,048 raw bytes before keys, indexes and database overhead.

3.1 Native preprocessing

The decoder inspects dimensions, rejects more than 250 million declared source pixels, and applies power-of-two subsampling towards one million decoded pixels while preserving a viable crop. Decoded images over 20 million pixels are rejected. These are defensive bounds, not measured throughput guarantees at those sizes.

The short side is scaled to 224, followed by a centred square crop and antialiased bicubic interpolation with $a = -0.5$. The two-pass implementation uses precomputed fixed-point coefficients; the horizontal pass retains only required crop columns. Bitmap resources are recycled on success and failure. RGB channels are divided by 255 and normalised with means $(0.48145466, 0.4578275, 0.40821073)$ and standard deviations $(0.26862954, 0.26130258, 0.27577711)$, then written in NCHW order. Tokenisation and preprocessing follow the CLIP/OpenCLIP conventions acknowledged in the source notices [3, 4].

3.2 Cache identity, validation and recovery

A key combines cache/model version, asset ID, modification time, width and height. It is not a content hash. An external edit preserving all key fields may evade invalidation; changing the model or preprocessing requires a version/invalidation decision.

RAM holds promises to prevent duplicate inference for the same key. Missing keys are hydrated from SQLite in batches of at most 400. Reads reject wrong dimensions, byte counts, non-finite components and non-positive/non-finite norms. Invalid rows are deleted. In the audited post-delivery edition, fail-closed guards also reject freshly computed non-finite or wrong-dimension embeddings before caching; the deployed snapshot predates those guards. Inference failure evicts its pending entry; a failed disk write does not discard a successfully computed RAM vector.

Successful gallery refresh prunes stale or inaccessible keys. Permission denial clears visible media and the embedding cache. Process restart discards sessions and RAM, but valid SQLite vectors persist. A failed session creation resets its promise for retry. History/reviews remain independent, so a saved record can outlive access to its referenced image.

4 User workflow and durable application state

The user first grants photo access, enters a visible-content observation and optionally narrows the candidate pool by date. Candidate preview makes the filter effect visible before inference. Search provides ranked suggestions; the user can inspect photos, select suitable results or explicitly confirm no match. A cosine score is not an acceptance probability.

Store	Purpose	Retention
Image embeddings	Versioned vectors and norms for reuse across process restarts.	Pruned against accessible keys.
Automatic history	Original successful query/filter, compared count and Top-5 result metadata.	Latest 100 successful searches.
Reviewed observations	Explicit selected-photo or no-match decision, separate from automatic history.	Until user deletion.
Diagnostics	Search phases, counters, outcome and diagnostic metadata.	Latest 500; development export latest 100.

4.1 Historical evidence versus a new search

Opening a history entry preserves its original rank/score snapshot. Editing prepares form inputs. Rerun executes a new search using the stored query directly, avoiding dependence on asynchronous React state updates. An old result is not silently replaced by a recomputation. A reviewed observation can be reloaded into the workflow, but does not fine-tune or update the model.

Deleting search history does not remove independent reviews. Deleting records does not delete gallery images. Photos are referenced by media metadata rather than copied into the record stores. Revoked access or deleted files can make a previously saved photo unavailable; the viewer reports that condition.

4.2 Persistence contracts

SQLite stores initialise lazily, use WAL and `synchronous=NORMAL`, and bind SQL values. History insertion and retention pruning use one exclusive transaction. Review validation requires a non-empty photo array for a photo decision and an empty array for no match. Lists order by timestamp and then ID to break ties.

Saved photo metadata includes original rank and finite similarity score. The history candidate count is the number successfully compared, which can differ from requested candidates when a file is skipped. No database encryption layer, schema-migration framework or damaged-file recovery guarantee is implemented.

Scope note. Local processing reduces dependence on a hosted inference service; it does not make local databases or diagnostic logs anonymous. Observation text and media references remain sensitive. Device backups and future production security controls require separate review.

5 Temporal selection and calendar correctness

Input	Semantics
Blank date	All accessible loaded photos; window ignored.
`YYYY-MM-DD`	Whole local calendar day: inclusive start, exclusive next midnight. Window is hidden/ignored.
Full local timestamp	`YYYY-MM-DD HH:mm` or `YYYY-MM-DDTHH:mm`; inclusive symmetric before/after window. Blank hours defaults to two.
Zero hours	Exact timestamp equality.
Invalid input	Reject time-only, impossible dates, negative/non-finite windows and overflow.

Whole-day selection uses calendar boundaries, not a fixed 24-hour duration. Regression tests cover leap days and both London daylight-saving transitions, including 23- and 25-hour days. Saved day-only context uses the existing nullable hours field, without a schema migration.

5.1 Why temporal and semantic correctness are separate

A relevant photo excluded by the date filter cannot be recovered by a better ranking model. Conversely, rank 1 in a one-photo pool proves inclusion, not strong semantic discrimination. The application therefore exposes candidate preview and preserves the original date/window in history.

MediaLibrary 18.2.1's upstream Android source was inspected during the original validation: the relevant provider mapping uses `DATE_TAKEN` for `creationTime`, rather than `DATE_ADDED`. A camera asset aligned with its capture naming, while a non-camera asset showed a large discrepancy between provider date fields. The public case study excludes the exact timestamps and identifiers.

This was a read-only comparison of selected assets, not a controlled import experiment. Imported or downloaded media can carry missing or misleading timestamps. No independent EXIF-normalisation feature or historical capture-time guarantee is claimed. Widening or clearing the filter remains a user-controlled recovery path.

5.2 What the automated tests establish

The Node suite contains 33 tests. Five workflow/store tests exercise strict date parsing, local-day membership, timestamp windows, history persistence/retention and independent review/no-match state through a real SQLite adapter; three tokenizer tests cover fixed reference vectors, edge cases and the 77-token context contract; five tests import the production cosine/ranking helpers directly, including rejection of non-finite norms and scores; seven ranking-pipeline tests check scores, event order, yield cadence, skipped and non-finite candidates, cancellation and timing keys; four session tests check reuse, independence, retry after failure and missing model URIs; five embedding-store tests check bit-exact round trips, model-version isolation, corrupt-row deletion, pruning and invalid-vector rejection; and four embedding tests confirm that a finite text embedding is returned unchanged and that NaN, infinite or wrong-dimension vectors are rejected before ranking, caching or persistence. Fifteen Python tests cover tokenizer parity with the TypeScript implementation, the statistics helpers, the A/B comparer (including missing, extra and duplicated searches) and the Android evidence tooling. They run the actual TypeScript implementation under Node where applicable. They do not test Android permissions, rendering, the native bridge, model numerical parity or the entire device workflow.

6 Model selection: five candidates

Five pretrained candidates were compared on the same Hard set of 20 queries and 16 images using a Windows Intel i7-13620H CPU without CUDA. Parameters below are counted totals, not the shorthand in model names. Table 1 and Figure 2 report quality and encoding time. Timings are representative runs, not controlled repeated-run medians.

Model	Params (M)	Hit@1	Hit@5	16 images (s)	Image/text (s)
MobileCLIP-S0	53.792	75%	95%	0.736	.046/.018
TinyCLIP 8M/16	23.446	35%	80%	0.503	.031/.008
TinyCLIP 40M/32	84.206	65%	90%	0.543	.034/.016
SigLIP Base	203.156	70%	95%	3.420	.214/.102
TinyCLIP 39M/16	83.101	85%	90%	1.544	.097/.022

Table 1: Historical desktop comparison. Image/text columns are mean encoding times per item. The corrected 40M configuration is used.

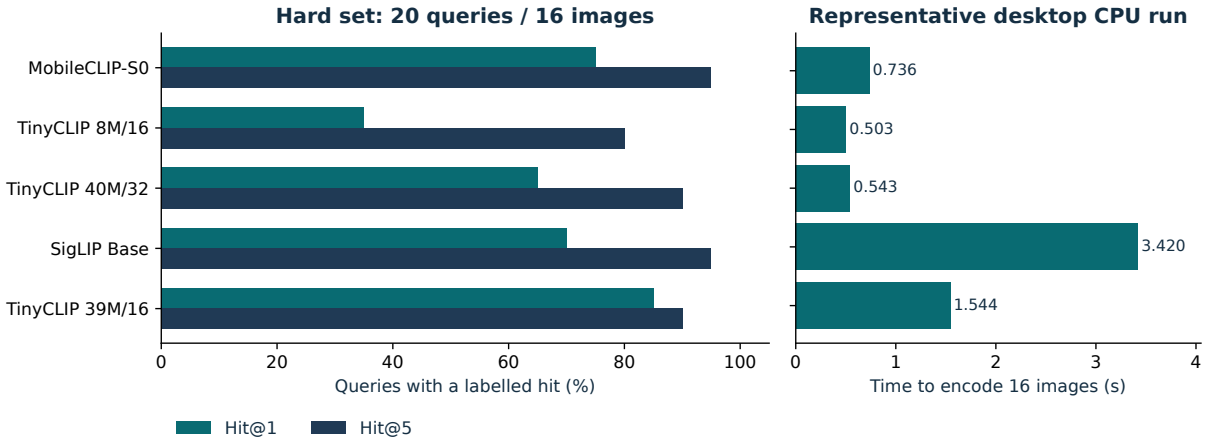


Figure 2: Quality and representative image-encoding time shown separately. The selected model did not dominate every metric.

A saved 39M baseline run took 2.511 s for the 16 images, with 0.157/0.030 s image/text means and the same quality counts. This illustrates run variability. The incorrect earlier 40M configuration is excluded. Mixed-format historical weight sizes are not used as direct APK comparisons.

TinyCLIP 8M/16 was smaller and faster but lost 40 percentage points of Hard Hit@1 against MobileCLIP. SigLIP had the largest parameter count and slowest measured path without a higher Hard Hit@1. TinyCLIP 39M/16 offered the strongest Hard Hit@1 in this comparison, with deployment cost addressed subsequently through caching and optimisation.

7 Retrieval quality, selection and failure analysis

Set / pool	MobileCLIP-S0 Hit@1 / Hit@5	TinyCLIP 39M/16 Hit@1 / Hit@5
Hard: 20 / 16	15/20 (75%) / 19/20 (95%)	17/20 (85%) / 18/20 (90%)
Expanded: 60 / 16	40/60 (66.7%) / 55/60 (91.7%)	44/60 (73.3%) / 55/60 (91.7%)
New Images: 30 / 10	23/30 (76.7%) / 29/30 (96.7%)	24/30 (80%) / 30/30 (100%)

Deeper evaluation exists only for these two candidates. TinyCLIP 39M/16 improved Hit@1 on all three sets, tied Expanded Hit@5, improved New Images Hit@5 and lost five percentage points on Hard Hit@5. It was neither the smallest nor the fastest, and did not win every metric.

7.1 Selection rationale

The chosen checkpoint, `wkcn/TinyCLIP-ViT-39M-16-Text-19M-YFCC15M`, combined useful measured retrieval behaviour with a viable ONNX/native deployment path and published MIT model metadata. The official TinyCLIP implementation supplies attribution to Microsoft and the model research is by Kan Wu et al. [1, 2]. Upstream notices remain separate from rights in this project and evaluation imagery.

Paired per-query outcomes do not separate the two leading models: exact McNemar tests give $p = 0.39$ for Expanded Hit@1 and $p = 1.0$ for the other comparisons. With 20 queries, 17/20 Hit@1 has a 95% Wilson interval of 64–95%. Licensing and the ONNX deployment path were therefore the deciding factors.

7.2 Where the evidence stops

Queries were curated with knowledge of the photographs. Small, partly related evaluation sets cannot be pooled into "110 independent queries". With only 20 Hard-set queries, the observed differences are descriptive and do not establish statistical superiority. Top-5 covers five of the sixteen candidate positions in that set, making it a relatively permissive criterion; Hit@1 was more discriminative. New Images means new to this evaluation, not proven absent from model pretraining. No teacher-population trial or external held-out benchmark establishes general classroom accuracy.

Literal descriptions, paraphrases, noisy text and ambiguous cases had different difficulty. In one noisy tablet-related query, the expected photo was only rank 14. An ambiguous formal-event image also limited interpretation; clearer new examples improved that cluster. Abstract statements about learning, participation, outcomes or intentions cannot reliably be inferred from visible appearance alone.

Historical desktop quality counts are not silently attributed to every final quantised Android output; Section 8.1 compares the shipped INT8 encoders directly. Text-encoder quantisation comparisons and physical-device functional examples have their own narrower scope. Differences between retained timing runs are treated as run variability, not merged into an invented benchmark.

Scope note. Human review is part of the system design. A ranked suggestion supports a person's decision; the model does not establish identity, learning outcomes or observation truth.

8 Quantisation and scheduler optimisation

8.1 INT8 text encoder

As shown in Figure 3, dynamic QInt8 quantisation reduced the raw text encoder from 178,520,196 bytes (170.25 MiB) to 45,042,123 bytes (42.96 MiB), a 74.77% reduction. Twenty synthetic token tensors gave mean FP32/INT8 cosine similarity 0.99762, minimum 0.99745 and mean absolute error 0.00244. Synthetic desktop median inference improved from 27.02 to 10.14 ms.

Retrieval with the shipped encoders. After delivery, the shipped INT8 text and image encoders were evaluated end to end (application tokenizer, standard CLIP preprocessing, ONNX Runtime on a desktop CPU) against the historical FP32 results on every labelled query and image.

Set (queries)	FP32 Hit@1	INT8 Hit@1	FP32 Hit@5	INT8 Hit@5
Hard (20)	17	14	18	18
Expanded (60)	44	46	55	54
New Images (30)	24	23	30	30
Total (110)	85	83	103	102

No set shows a statistically significant difference (exact McNemar $p \geq 0.25$ in each), and the shipped model returns the same first photo as FP32 in 97 of 110 queries. The total row is a descriptive sum across three related sets, not a pooled test or accuracy estimate (Section 7.2). All six top-1 losses remain in the top three, and the INT8 model gains four first places. The on-device Kotlin preprocessing path is not part of this comparison.

Twelve real English queries using an ASCII-equivalent tokenizer replica gave mean cosine similarity 0.97795, median 0.97997 and minimum 0.96068. These observations support useful tested semantic behaviour, not numerical identity or universal tokenizer parity.

Before this post-delivery comparison, no matched FP32-to-INT8 retrieval study existed for the image encoder, so the desktop model-selection results, including 17/20 Hard Hit@1, were not on their own evidence of quantised retrieval parity.

The observed device comparison changed median total latency from 604.00 to 539.61 ms and median text inference from 236.64 to 170.65 ms. Candidate pools differed (35 versus 36), so this is directional evidence rather than a strict matched A/B estimate.

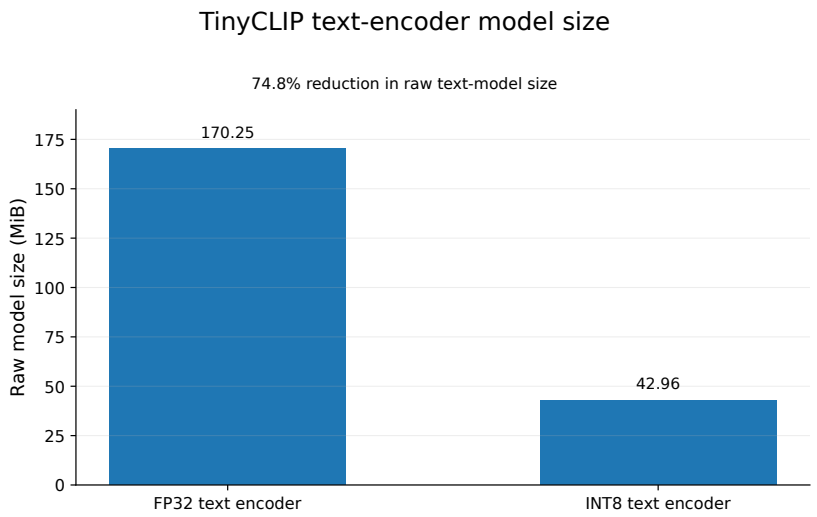


Figure 3: Text-encoder raw file size before and after quantisation. The reduction concerns raw model bytes, not process memory.

8.2 Measure orchestration, not only matrix operations

Figure 4 compares scheduler policies. The original ranking policy yielded roughly every ten candidates and again at the end. At 37 candidates, median ranking was 260.60 ms, including approximately 227.29 ms UI-yield time. Yielding every 20 reduced total median search from 530.92 to 379.42 ms and ranking to 110.00 ms.

An approximately-50-candidate policy reached 303.55 ms total and 43.68 ms ranking, but coincided with visual button-state concerns and was rejected. The retained policy yields every 20 candidates, including an interval boundary at the final candidate. Lower latency alone was insufficient to justify the more aggressive setting.

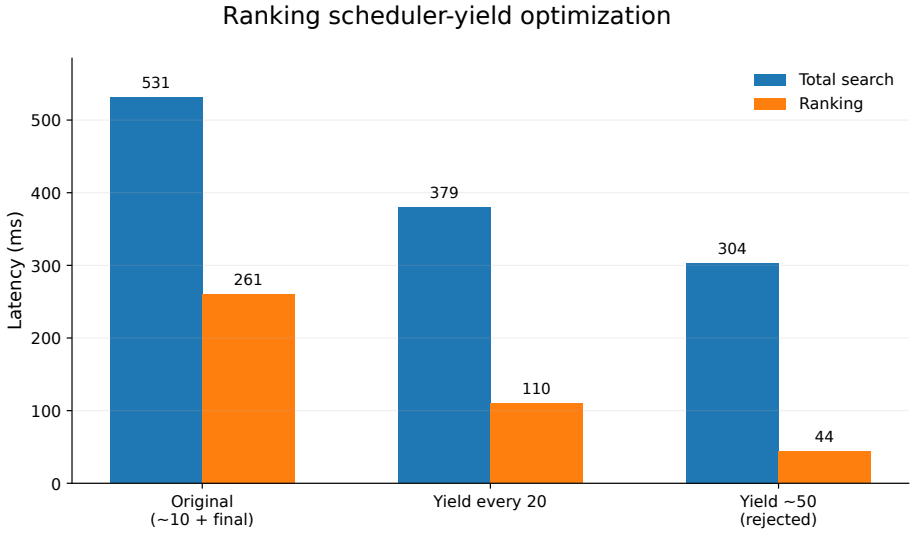


Figure 4: Scheduler experiment with 37 candidates. Ranking time is contained within total time; the bars are not additive. The fastest policy was rejected.

9 Native preprocessing and rejected preloading

The earlier image path decoded, resized, cropped, resampled and normalised the full input at substantial cost. Bounded Android decoding reduces pixels before the expensive bicubic stage, while preserving a valid 224-pixel crop.

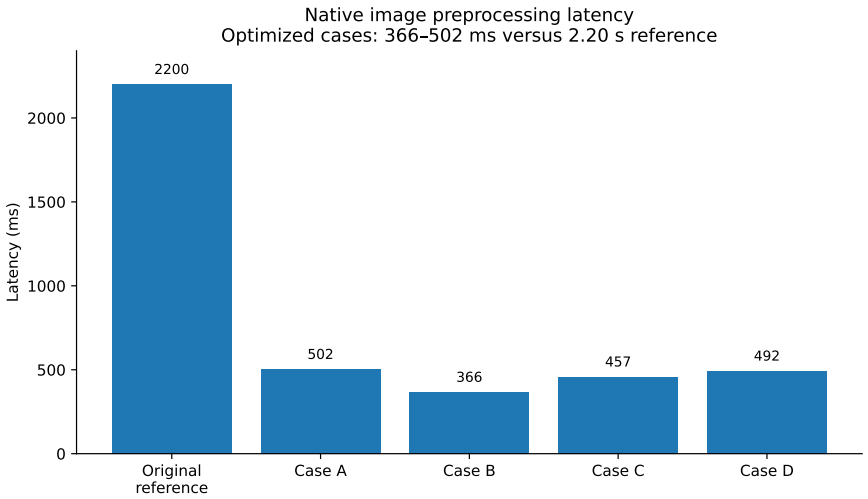


Figure 5: Native preprocessing latency for the reference and subsequent examples. These are different images, not paired same-image A/B measurements.

In Figure 5, the earlier reference was 2,199.73 ms. Optimised observations were 502.42, 366.17, 457.46 and 492.50 ms. Because images differ, a universal causal speedup ratio is not justified. The later new-image example completed preprocessing in 492.50 ms and ranked the intended photo first.

9.1 Image-model preload experiment

Proactive image-session loading shifted approximately 1.5–1.7 seconds of cold model creation out of the later search path. It also increased idle process PSS from approximately 76.3 to 166.6 MiB. The roughly 90 MiB idle cost was rejected on the constrained target; lazy session acquisition was retained. Figure 6 summarises the memory trade-off.

This decision separates occasional cold-path delay from persistent memory pressure. Fully cached searches may avoid the image session altogether, making always-on preloading a poor default for the tested usage.

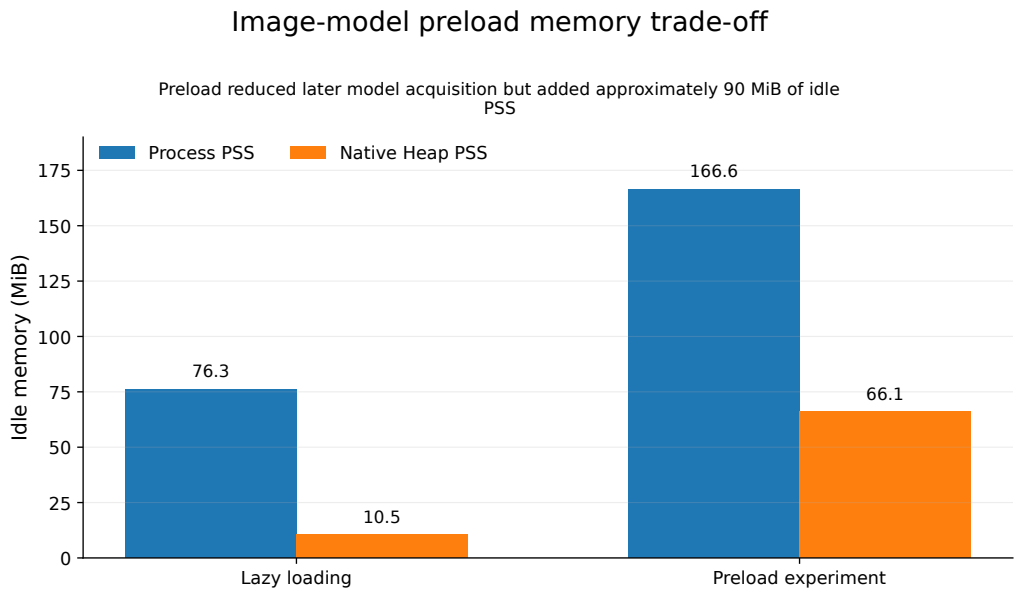


Figure 6: Idle memory with and without proactive image-session loading. Native Heap PSS is contained within whole-process PSS. The approximately 90 MiB idle increase motivated retaining lazy loading.

9.2 Compatibility boundary

The tested JPEG/PNG path does not establish general EXIF-rotation parity, HEIC support or cloud-only asset handling. Defensive decode bounds limit obvious resource blow-ups but are not a comprehensive hostile-image security evaluation. Native changes require a new bridge/parity and device-validation cycle.

10 Warm and sustained search performance

Table 2 reports five historical ARM64 Release searches over 43 cached candidates:

Run	Total (ms)	Text inference	Ranking	UI yield
1	435.16	153.79	177.70	136.69
2	436.51	170.96	164.07	125.09
3	470.21	193.00	166.68	127.77
4	424.93	163.71	163.17	125.07
5	437.70	168.27	159.47	122.38

Table 2: Five-run warm reference: 436.51 ms median total. Phase values are nested.

The separate sustained run in Figure 7 completed 108/108 searches over 44 cached candidates. All recorded completed outcomes and zero newly computed image vectors. The median was 441.07 ms, mean 440.62 ms, P95 459.28 ms and range 402.39–486.65 ms. Median text inference was 165.57 ms and median ranking 169.60 ms.

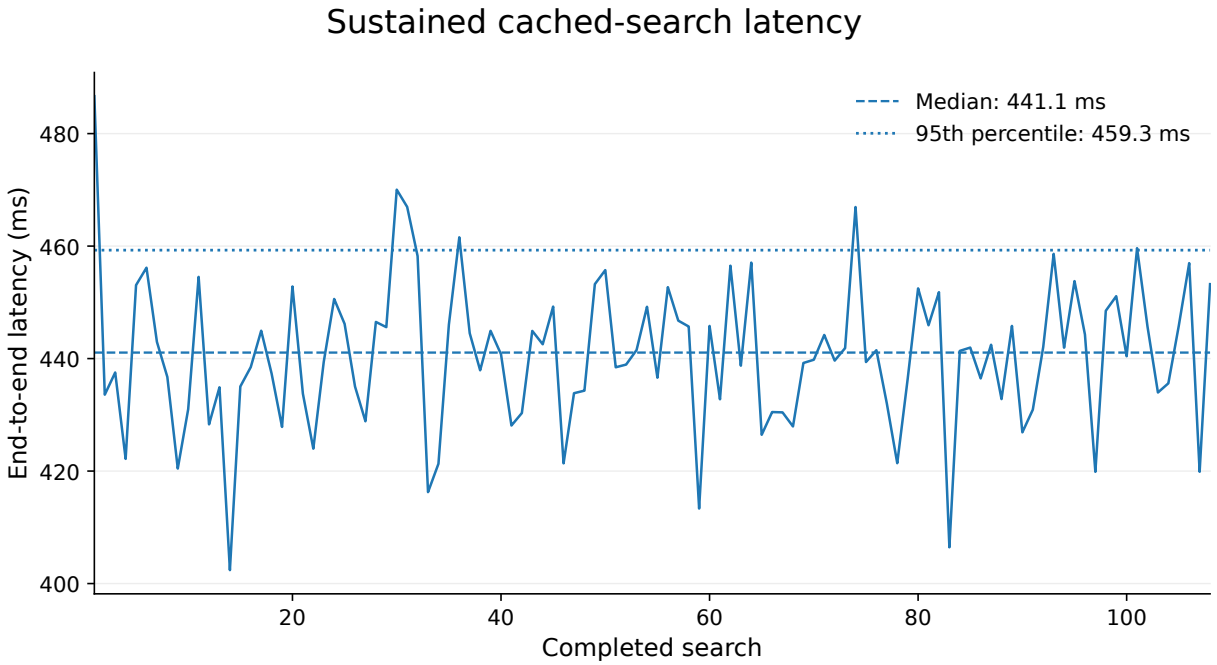


Figure 7: Instrumented latency across 108 consecutive cached searches. Its historical "end-to-end" axis refers to the instrumented pipeline, excluding final rendering and record persistence. The committed data retains only run order and duration.

The first 20 searches had median 437.03 ms and the last 20 median 444.99 ms, a 1.82% change. This demonstrates bounded variation in the tested sequence, not a general service-level guarantee. It does not test concurrent users, thousands of photos or repeated cold library indexing.

Scope note. "441 ms" refers to the instrumented cached pipeline on a particular device and library. It excludes final UI rendering and record persistence. It is not a universal tap-to-result latency measurement.

11 Cold paths and startup milestones

A single new-image case used 44 candidates: 43 cached vectors and one newly computed vector. The intended new photograph ranked first. The total instrumented duration was 4,225.41 ms.

Measurement	ms	Measurement	ms
Text encoding	452.32	Cold image-model acquisition	1,527.81
Text session	217.40	Ranking	2,027.31
Text inference	228.75	Image read	208.34
Index loading	214.43	Preprocessing	492.50
Image inference	726.81	Embedding persistence	8.89
UI yield	372.78	Total pipeline	4,225.41

These are nested diagnostic intervals, not a decomposition to sum. One successful cold example establishes pipeline execution and target retrieval in that case; it does not estimate a cold-path median or tail distribution.

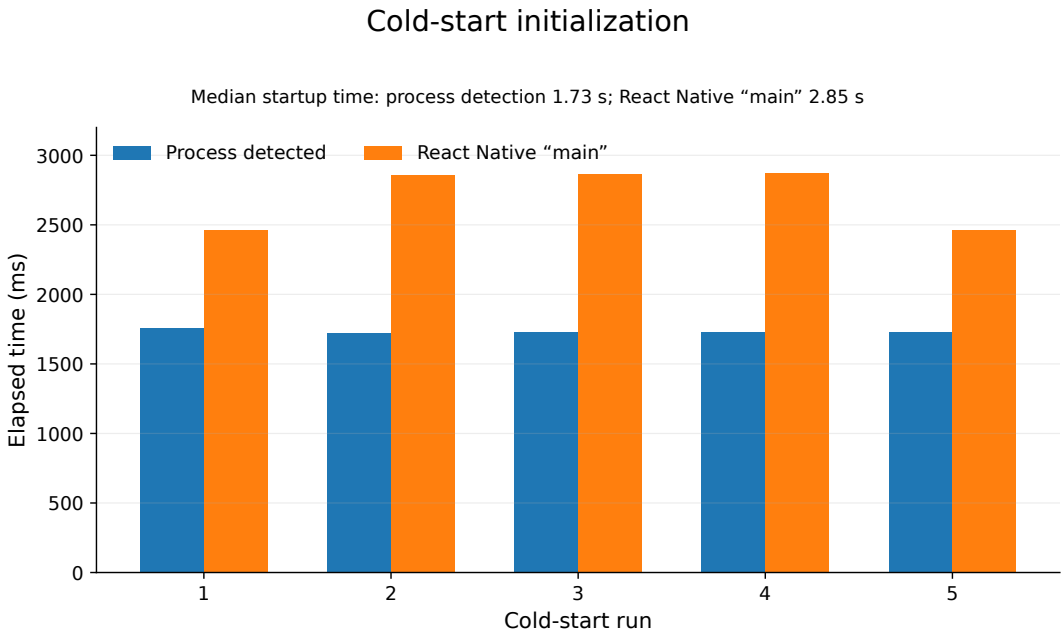


Figure 8: Cold-start milestones across five launches. The bars are milestones from launch, not incremental phases or fully-interactive startup.

The median process-detection time was 1.726 s; the median React Native `main` time was 2.855 s. Figure 8 retains its historical display rounding (1.73 s and 2.85 s). Neither is first-frame or fully-interactive startup. Cold app launch and cold image-model acquisition are distinct events.

12 Memory and stability

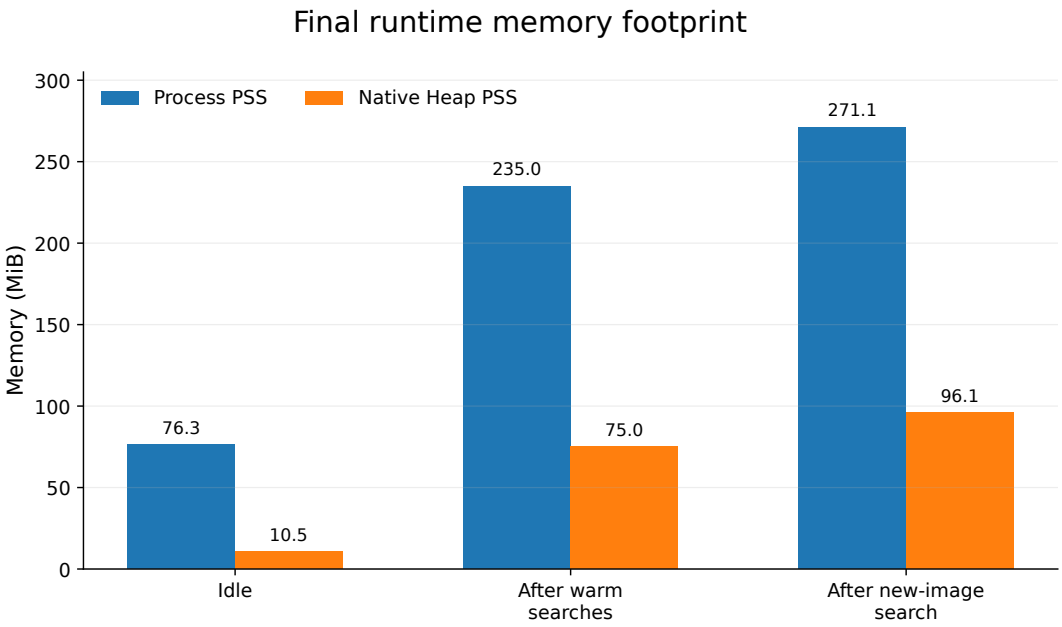


Figure 9: Runtime memory footprint across application states. Native Heap PSS is part of whole-process PSS, not extra memory to add. “Final” refers to the historical benchmark stage.

State	Process PSS (MiB)	Native Heap PSS (MiB)
Idle	76.26	10.49
After five warm searches	235.01	75.00
After new-image search	271.13	96.06

Figure 9 reports process PSS for the application as a whole, which varies with runtime state. It is not a direct measurement of model file size, nor a fixed per-query allocation. SQLite, JavaScript, native buffers, model sessions and the OS memory accounting all contribute.

Post-soak samples remained in the same general range. The historical log review found no application fatal exception, ANR or native fatal signal during the stress work. Combined with 108 completed sustained searches, that provides useful bounded stability evidence. It is not a formal memory-leak proof or evidence of reliability over weeks of production usage.

12.1 Operational design implications

Persisted vectors reduce repeat inference but do not eliminate the need to enumerate accessible assets. Lazy image-session loading avoids unnecessary idle memory in a cached workload. Versioned cache identity and validation prevent several stale/corrupt-row failures, but metadata-based keys and unencrypted stores retain explicit limitations.

Large libraries, low-storage conditions, process death during native work and broader file-format coverage are appropriate future validation targets. They are not completed results of this study.

13 Battery-side energy and thermal observations

The energy run used fixed brightness, an illuminated screen, Wi-Fi enabled, foreground application execution and USB disconnected. Device voltage/current readings estimated $P(t) = V(t)I(t)$ and energy by integration over time.

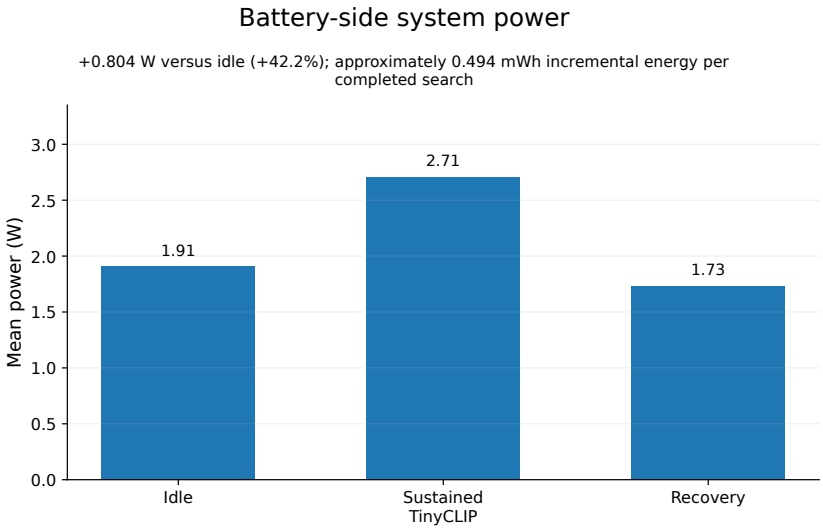


Figure 10: Whole-device battery-side power by workload phase. Incremental energy is an idle-subtracted estimate, not isolated inference energy.

Phase	Mean power (W)	Energy (mWh)	Mean current (A)
Idle	1.905	46.58	0.438
Sustained search	2.709	179.86	0.650
Recovery	1.732	42.83	0.414

The sustained phase in Figure 10 added approximately 0.804 W relative to idle (42.2%). Over 239 seconds, this gives approximately 53.4 mWh incremental energy: about 0.5 mWh per completed search under the tested idle-baseline method (108 searches). This whole-device estimate is sensitive to the baseline period and is not an isolated model-energy measurement or a universal per-query cost.

Android battery accounting estimated approximately 14.9 mAh for the app, versus approximately 14.1 mAh from the differential fuel-gauge calculation, a roughly 5.6% difference. This supports order of magnitude, not calibrated measurement accuracy.

During the controlled run, CPU frequency ceilings remained at 1.401 and 1.094 GHz. Figure 11 shows SoC readings moving from low/mid 30s into the mid/high 30s; a separate longer run reached 41 °C. No evidence of throttling was observed under these conditions. Different ambient temperature, charging state or longer/heavier loads could produce different behaviour.

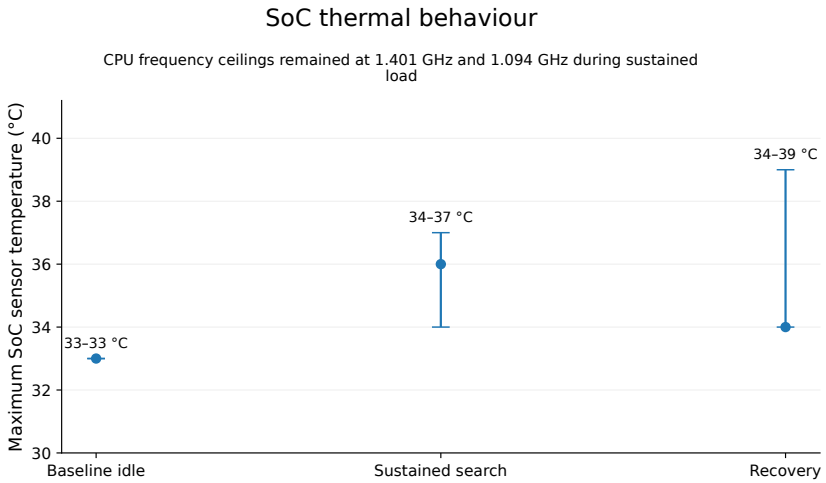


Figure 11: SoC temperatures by phase. Vertical extents show observed minimum–maximum readings: idle 33–33, search 34–37 and recovery 34–39 °C, not confidence intervals. Dots at 33, 36 and 34 °C are fixed display reference points with no independent statistical meaning.

14 Packaging, runtime identity and validation

The source stack is Expo SDK 54, React Native 0.81.5 and React 19.1. The JavaScript ONNX binding is 1.24.3; a committed patch pins native Android ONNX Runtime 1.29.0 and extensions 0.13.0, and removes obsolete module metadata. The JavaScript binding and native runtime are versioned separately [5, 6].

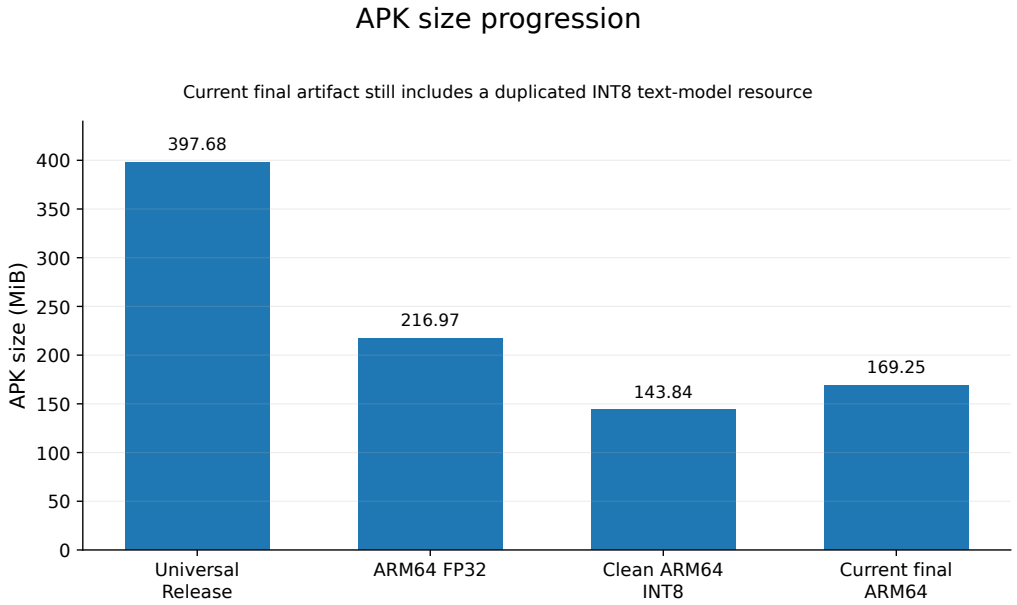


Figure 12: APK size progression. Its "Current final ARM64" label identifies the historical 169.25 MiB benchmark artifact, not the later 143.01 MiB packaging closure described below. A release build of the refactored code was later installed only for the functional A/B; its size and performance were not re-measured.

Figure 12 includes the historical benchmark APK: 177,467,535 bytes (169.25 MiB). Inspection found a duplicate INT8 text-model resource, explaining approximately 25.4 MiB of compressed overhead. Removing the duplicate resource produced the later packaging closure of 149,953,577 bytes (143.01 MiB), with exactly one copy of each required model and ARM64 libraries. That is packaging evidence, not a new performance result.

The evaluation build used development-key signing; it was not a production store release. The review build used a generic demonstration application ID and display identity, with no original APK or signing material included. It was not a byte-identical repackaging of the validated APK.

14.1 Source validation and behaviour-preserving refactor

A clean `npm ci` completed, including the committed patch. TypeScript checking, ESLint with zero permitted warnings and all 33 Node and 15 Python regression tests passed on the audited engineering tree. The declaration inventory was regenerated from that source during validation. One device check complements them: a release build of the refactored code was installed on the same tablet and returned the same Top-5 results and similarity scores as the delivered build for every compared search in the A/B comparison. The latency, memory, thermal and energy results in this report belong to the historical benchmark build and were not re-measured after the refactor.

After delivery, the application was refactored without behavioural change. The 3,749-line screen became domain hooks, presentational components and dependency-injected modules. A compiler-API check confirms that 143 original units survive verbatim with no added control flow, Prettier's AST-equivalence check covers formatting, and one validation command runs 33 Node and 15 Python tests. On the tablet, an A/B comparison of both builds returned identical Top-5 results and similarity scores. After the A/B, deliberate fail-closed guards were added: NaN or infinite embeddings and similarity scores, and image embeddings without 512 dimensions, are rejected instead of reaching the ranking or the cache, while valid finite values follow the previous code path unchanged.

The private validation checks require no model files. Model fingerprints are retained in the audited engineering record, while this public case study does not distribute exported model binaries or provide a download endpoint for them.

15 Temporal acceptance: fixed-target case study

The timestamp-window cases in Table 3 used a fixed intended target in an unchanged 44-photo library and the London timezone. $D0$ denotes an observation timestamp on the target's calendar day; $D-3$ denotes the same local clock time three days earlier. The target was captured slightly after the $D0$ observation time. Exact dates, media IDs and filenames are removed here. Whole-day cases are described separately below.

Observation timestamp	± hours	Compared	Target included?	Rank / score
$D0$	2	1	Yes	1 / 0.2745
$D0$	24	3	Yes	1 / 0.2745
$D0$	168	14	Yes	1 / 0.2745
$D-3$	2	1	No: outside window	Not ranked
$D-3$	168	14	Yes	1 / 0.2745

Table 3: One fixed target across temporal windows. These are functional cases, not independent accuracy trials.

The target remained first as the candidate pool grew from one to three to fourteen. Its margin over the runner-up narrowed from 0.1084 in the 24-hour window to 0.0270 in the 168-hour window. Wider selection recovers candidates but can also add closer distractors.

The narrow cross-day pool contained a different, semantically valid bottle photograph. Its rank 1 did not mean the predesignated later target had been retrieved. The correct conclusion is target exclusion by date, not a general semantic no-bottle failure. The wider cross-day window recovered the intended target.

Whole-day selection on $D-3$ previewed ten photos. A $D0$ whole-day search compared three; reopening and rerunning preserved the whole-day interpretation and count. This supports the date-only persistence behaviour in the later application build.

15.1 What this demonstrates

The mechanism can accommodate an observation and related photograph taken on different days. It does not constitute a new classroom/art dataset, guarantee that widening always improves retrieval, or establish general reliability of imported timestamps. A one-photo pool proves inclusion only. Human review remains necessary even when the fixed target is ranked first.

16 Public evidence and distribution

16.1 What is independently inspectable

The public case-study repository publishes this report, technical diagrams and de-identified numeric evidence for model selection, retrieval quality, sustained and warm latency, startup milestones, temporal acceptance and scoped engineering measurements. The application source code, native-module implementation, model binaries, APKs and private image datasets are intentionally not distributed.

The retained evidence supports inspection of the reported numbers and their scope. It does not make the original private image evaluation independently replayable, because the underlying images and linked per-query records are not redistributed.

Expected encoder	Bytes
Text INT8	45,042,123
SHA-256: `e403265d2b39a135603a9f3d15995769faa6e39665f9445cdaa477c3c9a10c29`	
Image INT8	41,325,670
SHA-256: `37a67c6d03d13474f33484ba2bc68df7976a5f0450ea38fbe62d483dbb76996c`	

These hashes identify the evaluated encoder binaries and are not credentials. Model files themselves are intentionally omitted. Historical export tooling was not fully revision-pinned, so this case study does not claim bitwise recreation from a clean upstream environment; a fresh export would require explicit numerical-parity, contract and cache-invalidation validation.

16.2 Validation record

The audited engineering implementation passed strict TypeScript checking, zero-warning linting, 33 Node regression tests and 15 Python tests. A mechanical refactor proof preserved 143 original units with zero mismatches. The delivered and refactored builds were also compared on-device, with the same Top-5 order and similarity scores for every compared search. The historical 108-search performance benchmark was not rerun after the refactor, so its latency, memory, thermal and energy numbers remain attached to the historical benchmark build.

16.3 Rights and reuse

TeachScribe Ltd owns the proprietary application code described by this case study. The implementation is not published here. The engineering work was designed and developed by Alberto Reyes Castro and is presented as a portfolio case study. Publication of the case-study material does not grant a licence to use, copy, reconstruct or redistribute TeachScribe's unpublished application source or other proprietary product materials. Third-party components and pretrained models remain under their own licences.

17 Privacy, limitations and next engineering work

The public case study excludes application source code, original photos, gallery screenshots, thumbnails, raw metadata, linked query records, device identifiers, local paths, internal briefs, signing material and the unused photo-derived reference vector. The report uses neutral metadata and reviewed numeric observations.

17.1 Limits that materially affect interpretation

1. **Evaluation scale.** One older device, small candidate pools and curated queries do not establish external validity, large-library performance or classroom outcomes.
2. **Different evidence stages.** Desktop selection, quantisation parity, historical device performance and later functional acceptance are distinct. The refactored build has no new device benchmark; its equivalence was checked with the documented on-device A/B protocol.
3. **Measured boundaries.** Pipeline timing is not final-frame latency; startup milestones are not interactivity; PSS is whole-process; energy is an idle-subtracted device estimate.
4. **Data reliability.** Media timestamps and metadata-based cache identity can be misleading. Saved references cannot restore removed files.
5. **Scope of this repository.** It supplies no production signing, formal security review, encryption layer, migration framework, broad device/format matrix or current vulnerability-free claim; production integration, signing and distribution belong to TeachScribe's app.

17.2 Prioritised future work, not completed claims

First, establish a consented held-out retrieval set with clear relevance criteria and uncertainty reporting. Next, test fresh model exports and cache-version transitions under controlled conditions. Extend profiling to larger libraries and multiple devices, including genuine tap-to-visible timing and repeated cold paths. Review log minimisation, backup policy, dependency security and failure recovery as part of ongoing production hardening. Once production telemetry is available, measure feature adoption and reviewer outcomes, and tune the per-candidate scoring loop for throughput.

17.3 Engineering outcomes

The completed work spans multimodal evaluation, quantised on-device deployment, Kotlin/TypeScript integration, persistent embedding caches, instrumentation and human-reviewed retrieval workflows.

18 Public case-study boundary

This public derivative is intentionally detailed without being a source release. It preserves product context, architecture, model-selection methodology, tensor contracts, cache and persistence semantics, quantified optimisation trade-offs, validation outcomes, limitations, technical charts and de-identified numeric evidence.

It intentionally excludes the React Native/TypeScript application source, Kotlin native-module source, implementation tests and build/provisioning scripts, model binaries, APKs, private images and screenshots, real media identifiers and filenames, exact capture dates, signing material and internal delivery correspondence.

The resulting boundary follows a simple rule: describe and evidence the engineering work, but do not publish reusable proprietary implementation code. This keeps the technical case study inspectable for engineering review without presenting the commercial application as open source.

References

[1] K. Wu et al. *TinyCLIP: CLIP Distillation via Affinity Mimicking and Weight Inheritance*. ICCV, 2023, pp. 21970–21980. Official implementation and licence: `https://github.com/microsoft/Cream/tree/main/TinyCLIP`.

[2] TinyCLIP model card: `wkcn/TinyCLIP-ViT-39M-16-Text-19M-YFCC15M`. `https://huggingface.co/wkcn/TinyCLIP-ViT-39M-16-Text-19M-YFCC15M`.

[3] OpenAI. *CLIP: official implementation and licence*. `https://github.com/openai/CLIP`. Tokenizer-related notice preserved in the repository.

[4] OpenCLIP contributors. *OpenCLIP implementation and licence*. `https://github.com/mlfoundations/open_clip`. Tokenisation and preprocessing conventions are acknowledged separately from project authorship.

[5] Expo. *SDK 54 source and documentation snapshot*. `https://github.com/expo/expo/tree/sdk-54`. The project retains its evaluated dependency pins; later documentation/API versions are not substituted.

[6] Microsoft. *ONNX Runtime*. `https://github.com/microsoft/onnxruntime`. The JavaScript package and Android native versions are identified separately in this report and the committed patch.

[7] TeachScribe engineering evidence, September 2026. De-identified numerical extracts are published with the public case study. Original technical charts and the architecture schematic are retained. The underlying private images, raw device artifacts, application source and internal delivery documentation are intentionally not redistributed. No new benchmark is implied.

Scope note. Author: Alberto Reyes Castro. GitHub: `github.com/albertoreyescastro`. This case study presents applied engineering work with pretrained models and measured prototype evidence. It makes no claim of organisational endorsement, production certification or universal retrieval accuracy.