NextGen Scientific Publication
World Forum on Cybersecurity, Machine Learning, and Intelligent Systems (WFCMLS-2026)

Pg. No. 32-42, WFCMLS26-104, 2026
https://doi.org/10.63282/3117-5481/WFCMLS26-104
Conference Date: 27th March 2026

Original Article

Personalized Interface Generation via Constrained Contextual Bandits at Billion-User Scale

*** Nilesh Agrawal**
Meta AIDE.

Abstract:

The next billion mobile users are arriving on devices that flagship- market UI frameworks were never designed to serve. Rather than maintaining parallel codebases—as exemplified by Facebook Lite and Instagram Lite—this paper proposes a unified architecture where the UI is dynamically composed by the server and continuously optimized by ML models that learn the right interface for every user-device-context combination. We present AIDE (Adaptive Interface Delivery Engine), a frame- work combining Server-Driven UI (SDUI) for real-time layout com- position with contextual bandit models that personalize UI con- figurations across 40+ emerging-market signals including device capability, measured network quality, battery state, and data-cost sensitivity. The server determines not only what content to show, but how to render it—which component variants to assemble, at what fidelity, in what layout—without requiring client binary up- dates. Evaluation over six-week A/B experiments across 2.4B+ monthly active devices shows statistically significant improvements on Lite- tier devices versus dedicated lite applications: 34% lower crash rates ($p < 0.001$), 41% faster Time-to-Interactive ($p < 0.001$), 22% longer session duration ($p < 0.01$), and 50% broader feature usage—all from a single codebase with under 3% code divergence. We also report an ablation study isolating the ML model's contribution at +18% session duration over rule-based SDUI.

Keywords:

Server-Driven UI, Adaptive Interfaces, ML-Optimized Rendering, Emerging Markets, Contextual Bandits, Engagement Optimization, Device- Tier Adaptation.

1. Introduction

Over 60% of the world's 5.4 billion mobile users are in emerging markets—South and Southeast Asia, Sub-Saharan Africa, Latin America—where the median Android device ships with 3–4 GB RAM, an entry-level chipset, and 32–64 GB storage [16]. These users expect the same rich experiences available on flagship hardware, yet applications designed for flagship devices deliver degraded, often unusable performance on the hardware the majority of the world actually holds.

The industry's response—building separate "lite" applications (Facebook Lite [1], Instagram Lite [2], YouTube Go)—achieves meaningful performance wins but introduces structural costs that worsen with scale: divergent codebases, 6–12 month feature parity lag, and engineering teams split between maintenance and innovation. Section 2 examines these tradeoffs in depth through the FB Lite

and IG Lite case studies.

This paper proposes a fundamentally different approach: a single application whose UI is dynamically composed by the server and continuously optimized by ML models. The client becomes a thin rendering shell; the intelligence lives server-side where it can be updated instantly, A/B tested continuously, and personalized at individual granularity.

1.1. The Emerging Market Signal Landscape

Emerging markets present signals that extend far beyond device specs and should inform UI decisions but rarely do in conventional architectures. Table 1 summarizes the signal taxonomy.

1.2. Contributions

This paper makes the following contributions:

- AIDE (Adaptive Interface Delivery Engine): A production architecture where server-side UI composition replaces client- side conditional rendering, enabling real-time UI adaptation without client deploys.
- ML-Optimized UI Personalization: A contextual bandit system with constraint filtering that learns optimal UI configurations per user-device-context, maximizing engagement while respecting device resource budgets.
- Emerging Market Signal Framework: A taxonomy and weighting system for signals unique to emerging markets, going be- yond device specs to include data-cost sensitivity, battery anxiety, and behavioral patterns.
- Production Validation: Empirical results from six-week A/B experiments across 2.4B+ MAU with statistical significance testing, including ablation isolating ML contribution from structural SDUI benefits.

2. Case Study: Lessons from Facebook Lite and Instagram Lite

2.1. Facebook Lite: The Pioneer

Facebook Lite launched in 2015 as a ground-up rewrite targeting sub-1 GB RAM devices on 2G networks [1]. Its design principles established the template for emerging market optimization: minimal APK size (under 2 MB vs. ~60 MB main app), server-rendered content delivered as lightweight payloads, aggressive data management with images compressed to a fraction of main-app quality, and a simplified interaction model replacing complex gestures with explicit tap targets.

Table 1. The Emerging Market Signal Landscape—Signals That Should Dynamically Influence UI Composition

Signal Category	Specific Signals	UI Implication
Device Hardware	RAM (2–4 GB), SoC tier, GPU class, storage free, screen	Component complexity, animation fidelity, image cache density size, view recycler pool depth
Thermal State	CPU temperature, throttle level, sustained load capacity	Dynamic downgrade of GPU-heavy components mid-session; disable blur, reduce layers
Network Quality	Measured throughput (not reported type), latency, jitter, stability class	Image resolution, video autoplay, prefetch depth, API batching strategy
Battery Context	Level, drain rate, charging state, projected session remaining	Sync frequency, animation FPS, media upload deferral
Data-Cost Sensitivity	Prepaid vs. postpaid, data balance, carrier price/GB, Wi-Fi availability	Auto-download settings, image quality default, video loading behavior
Usage Patterns	Session length, scroll velocity, tap-vs-swipe ratio, dwell time, feature frequency	Layout density, content mix, feature visibility, shortcut placement
Locale Context	Language, text direction, script complexity, cultural content norms	Typography sizing, layout direction, card dimensions, tap target sizes

Key Insight. Users in emerging markets accept substantially different UI if the core value proposition is preserved. Users did not miss

parallax scrolling, blur effects, or 60fps animations—they noticed when the app crashed, when images failed to load, and when navigation was confusing. This insight is foundational: *optimize for reliability and clarity first, visual richness second.*

2.2. Instagram Lite: The Iteration

Instagram Lite's 2021 relaunch reflected evolved thinking [2]. Rather than a stripped-down feature set, IG Lite launched with Reels, Stories, Shopping, and DMs—recognizing that feature parity drives retention. It used a native shell with server-rendered content surfaces, modular on-demand loading, and an adaptive media pipeline adjusting quality based on network and device signals.

2.3. Structural Limits of the Lite Approach

Table 2 summarizes the achievements and structural limitations of the lite app approach alongside SDUI+ML alternatives.

3. Architecture: The Aide Framework

AIDE inverts the traditional mobile architecture. In a conventional app, the client contains UI logic—it knows how to lay out a feed card, render a profile, animate a transition. In AIDE, the client is a thin rendering engine that receives a declarative UI specification from the server. The server composes this specification dynamically based on the full signal context of the requesting device and user.

3.1. The SDUI Rendering Pipeline

The system operates through four stages on every screen request:

Stage 1: Signal Collection (Client → Server). The client transmits a device context payload (~200 bytes, piggybacked on existing API requests) containing 40+ signals: hardware capabilities (RAM, SoC, GPU, storage), runtime state (thermal level, available memory, frame rate), network quality (measured throughput, latency, stability), battery context (level, drain rate, charging), and behavioral signals (scroll velocity, session duration, feature usage).

Stage 2: Context Scoring (Server). The server computes a multi- dimensional Device Context Score (DCS) producing independent scores across five dimensions: compute budget (0–100), render bud- get (0–100), network budget (0–100), power budget (0–100), and data-cost budget (0–100). Each dimension is a weighted, min-max normalized aggregation of 3–7 raw signals, with weights tuned via offiine regression against observed performance regressions (crash rate, jank frequency, session abandonment). Hard caps override the weighted score when any single signal is critical—e.g., avail- able RAM below 512 MB forces compute budget to zero regardless of other signals. A device with strong compute but poor network receives a different UI than one with weak compute but excellent network.

Stage 3: UI Composition (Server, ML-Informed). The composition engine assembles a screen specification from a component variant library, selecting variants based on DCS scores and the ML personalization model's ranking (Section 4). Output is a declarative UI tree—component types, properties, layout constraints, data bindings—transmitted as a compact binary payload (2–4 KB per screen).

Stage 4: Client Rendering. The rendering engine interprets the UI tree and instantiates platform-native components (Android Views/iOS UIKit). The client maintains a component registry mapping server types to native implementations. Each component accepts quality parameters from the UI tree without needing to know why those values were chosen.

3.2. Component Variant System

Every UI component exists in multiple variants designed for different resource budgets. The server selects variants based on DCS scores. This is the mechanism through which a single codebase serves every device tier. Table 3 presents the component variant matrix.

3.3. Declarative UI Tree Specification

The server transmits UI specifications as binary-encoded component trees using a custom protocol buffer schema optimized for UI trees, achieving 60–70% smaller payloads versus JSON-based SDUI implementations. Each node specifies: component type and variant, layout constraints, quality parameters (animation duration, image resolution, blur radius), data bindings, interaction handlers mapped to server-defined actions, and visibility conditions (e.g., "show only if network budget > 60"). The client caches specifications aggressively, only fetching updates when the server indicates a change.

Table 2. Achievements and Structural Limits of the Lite App Approach, with SDUI+ML Alternatives

Achievement	Limitation	SDUI + ML Solution
Proved emerging markets need distinct UIs	Binary choice: lite or main, nothing in between	Continuous spectrum; every user gets their optimal UI
Demonstrated server-side content rendering	Content was adaptive but layout was static, hardcoded in client	Both content AND layout composed server-side
Achieved tiny APK sizes (<2 MB)	Tiny APK meant fewer capabilities; no AR, limited camera	Moderate APK (~15–20 MB) with on-demand module loading
Reduced data consumption significantly	One-size-fits-all savings; no per-user preference learning	ML learns per-user data-quality tradeoff preferences
Reached 200M+ MAU in target markets	Created organizational silos; competing roadmaps	Single team, single codebase, single roadmap

Table 3. Component Variant Matrix—How Every Major UI Component Adapts across Device Capability Tiers

Component	Ultra (85–100)	Standard (40–64)	Lite (20–39)	Minimal (0–19)
Feed Card	Rich card: inline video, engagement overlay, reaction animations, 3-line preview	Standard card: image + text, tap-to-play video, 2-line preview	Compact card: thumbnail + headline, no video	Text-first: headline only, image on tap
Stories Tray	Animated ring, auto-advance preview, 72px avatars	Static ring, no preview, 56px avatars	Compact list, 40px avatars, no ring	Text-only: "3 new stories from..."
Reels Player	Full-screen 1080p, real-time effects, comments overlay	Full-screen 720p, basic effects, tap for comments	Inline 480p, no effects, separate comments	Thumbnail grid, tap to stream at 360p
Navigation	5-tab bar + gesture nav + FAB	5-tab bar + swipe nav	4-tab bar (merge Explore into search)	3-tab bar + hamburger for rest
Profile Page	Cover parallax, post grid with hover, highlights carousel	Static cover, post grid, highlights row	No cover, compact post list, highlights as link	Bio + stats + chronological list, no images default
Comments	Threaded, inline replies, reaction picker, GIF keyboard	Threaded, inline replies, emoji reactions	Flat list, tap to reply, text reactions	Paginated flat list, basic text input
Image Viewer	Pinch-zoom, swipe gallery, EXIF overlay, share sheet	Pinch-zoom, swipe gallery, share button	Single image, zoom button, share	Progressive JPEG, no zoom, save/share

3.4. Client Rendering Engine

The client's rendering engine diffs incoming UI trees against the currently rendered tree (analogous to React's virtual DOM diffing on native component trees), manages component lifecycle with tier-adjusted pool sizes, enforces per-frame GPU time and memory budgets derived from DCS scores (triggering automatic quality reduction if budgets are exceeded for two consecutive measurement windows), and reports performance telemetry on every rendered frame, creating a closed feedback loop informing future composition decisions.

4. Ml-Optimized Ui Personalization

The SDUI architecture enables dynamic composition, but requires a decision system to select which variant of each component to use.

This section describes the ML model that transforms AIDE from rule-based adaptation to learned, personalized optimization.

4.1. Problem Formulation

We formulate UI personalization as a contextual multi-armed bandit problem. For each screen request, the system chooses a UI con- figuration (the "arm") maximizing a reward signal (engagement) given context (device signals + user history). The challenge is three-fold: the action space is combinatorial (8 components × 4 variants = 65,536 configurations), the context is high-dimensional (40+ de-vice signals, 20+ behavioral features, temporal features), and the reward is delayed and composite (session duration, interactions, return probability, jank events).

4.2. Model Architecture

The model operates in two stages:

Stage 1: Constraint Filtering. A rule-based pre-filter eliminates configurations exceeding device resource budgets, reducing the action space to 50–200 feasible configurations per screen. Configurations requiring GPU render time, memory, or bandwidth exceeding DCS-derived limits are excluded.

Stage 2: Contextual Ranking. A neural contextual bandit scores each feasible configuration by predicted engagement reward. The model uses a two-tower architecture: a context tower encoding device-and-user context into a 128-dimensional embedding, and a configuration tower encoding the UI specification into a 64- dimensional embedding. The dot product of embeddings produces an engagement score.

4.3. Training and Inference Details

The model is trained on logged interaction data using a pairwise hinge loss that maximizes the score gap between configurations that led to higher engagement versus those that did not, within the same context. Key technical details include:

- Reward attribution: Rewards are attributed at the session level. All UI configurations served during a session share the session's composite reward, discounted by recency.
- Counterfactual correction: We use inverse propensity scoring (IPS) [18] to correct for the selection bias inherent in logged data.
- Feedback loop mitigation: Periodic policy resets (10% of traffic receives uniformly random feasible configurations for one week per quarter) and an exploration floor ensuring every feasible configuration receives at least 0.5% selection probability.
- Model update cadence: Retrained hourly on the trailing 7 days of interaction data. Model inference latency P99 is under 5ms.

4.4. What the ML Model Learns

The model discovers optimization patterns that would be difficult to encode as manual rules. Table 4 presents six representative patterns with engagement impact and confidence intervals.

4.5. Reward Function Design

Table 5 presents the composite reward function. The jank penalty prevents the model from choosing highest-fidelity UI that maxi- mizes engagement when it works but crashes on constrained de- vices. The data efficiency term ensures the model learns data con- servation on metered connections.

4.6. Exploration Strategy

The model uses Thompson Sampling with two domain-specific modifications: (1) *Conservative exploration on constrained tiers:* Ex- ploration probability is reduced by 50% for Lite and Minimal devices, because the cost of an over-ambitious UI (crash, severe jank) exceeds the opportunity cost of a slightly suboptimal but safe configuration.(2) *Feature launch boost:* New feature component variants receive a temporary exploration bonus to accelerate learning, preventing cold-start delay.

5. Emerging Market Signal Intelligence

5.1. Data-Cost Awareness

In markets where mobile data costs $2–5/GB and average monthly income is $200–400, every megabyte has real economic cost. The system implements data-cost intelligence through: a carrier-level pricing database mapping carriers across 60+ markets to coarse-grained cost buckets, prepaid cycle awareness tracking estimated data balance, Wi-Fi opportunity optimization for pre-fetching, and user-controlled data modes (Data Saver, Balanced, Full Quality) setting a floor on the data-cost budget.

5.2. Battery Anxiety Mitigation

Battery anxiety is documented in emerging markets where many users charge once daily at home. Production telemetry shows users begin modifying app usage at ~30% battery—well before any technical limitation. The system addresses this through session duration projection, battery-friendly defaults below 30% (reduced animation FPS 60→30, disabled video autoplay), and a charging-state reward serving highest-fidelity UI when charging.

5.3. Network Reality vs. Network Reporting

A critical finding: reported network type is unreliable for UI decisions. Table 6 presents the gap between reported network type and measured throughput. AIDE measures actual delivery performance through opportunistic probing and uses measured quality rather than reported type for UI composition.

5.4. Device Thermal Intelligence

Devices with passive cooling thermally throttle aggressively under sustained load. A "Standard"-tier device can spend 40% of a 30- minute session in a throttled state where effective capability drops to "Lite" or below. The system monitors thermal state continuously and applies asymmetric response: downgrade within one rendering cycle (5 seconds), but recovery is gradual over three cycles (15 seconds) to avoid re-triggering thermal pressure.

6. Production Implementation

6.1. Prior Art: Collaboration Platform Optimization

The rendering techniques underlying AIDE were first validated on a real-time collaboration platform serving users in India and Brazil: 30% GPU workload reduction through shader complexity tiering, compositing layer merging, and blur replacement; and 20% CPU utilization improvement through deferred layout computation, batched DOM reconciliation, and reduced background sync frequency [8]. These hardcoded optimizations become configurable, ML-driven adaptation parameters in AIDE's component variant system.

6.2. Component Library

The production system maintains 47 component types averaging 3.8 variants each, organized into three client-side tiers:

- Core (12 types): Always in client binary. Text, image, button, container, list, grid, input, icon, avatar, badge, divider, spacer. Covers 85% of screen composition needs.
- Extended (23 types): Loaded on first use. Video player, camera, maps, carousel, charts, rich editor, feature-specific components.
- Premium (12 types): Only loaded on devices with compute budget > 60. AR overlays, real-time filters, advanced animation containers.

Table 4: ML-Discovered UI Optimization Patterns with Engagement Impact and 95% Confidence Intervals

Discovered Pattern	Context	Impact
Text-first feed on data reset day	Prepaid users in India, first session after midnight data reset	+31% session duration (±4.2%, 95% CI)
Compact stories for fast scrollers	Users with >3 swipes/sec on any device tier	+14% story views (±2.8%, 95% CI)
3-tab nav for feature-light users	Users accessing <3 features/session on Lite devices	+8% session retention (±1.9%, 95% CI)
480p Reels on Wi-Fi + low battery	Battery <25%, Wi-Fi connected, Standard-tier	+19% Reels consumption (±3.6%, 95% CI)
Full-fidelity UI when charging	Any tier, actively charging, stable network	+26% engagement (±3.1%, 95% CI)
Expanded comments in evening sessions	Southeast Asia, sessions 8–11 PM	+22% comment interactions (±4.5%, 95% CI)

Table 5. Composite Reward Function for ML-Driven UI Optimization

Component	Weight	Definition
Session duration	0.30	Log-normalized time spent in app
Meaningful interactions	0.25	Comments, shares, saves, profile visits
Return probability	0.20	Predicted 24-hour return probability
Jank penalty	−0.15	Dropped frames >16ms, ANR, OOM
Data efficiency	0.10	Engagement-per-MB ratio

Table 6. Gap between Reported Network Type and Measured Throughput in Emerging Markets. P10 = 10th Percentile

Reported Type	Median	P10	% <1 Mbps
4G/LTE (India)	8.2 Mbps	0.6 Mbps	23%
4G/LTE (Nigeria)	5.1 Mbps	0.3 Mbps	34%
3G (Indonesia)	2.4 Mbps	0.2 Mbps	41%
Wi-Fi (Brazil)	15.3 Mbps	1.1 Mbps	18%

6.3. Server Infrastructure

The UI composition server is deployed at CDN edge PoPs. Composition latency P99 is under 15ms, with component template cache hit rate of 97%. ML model inference adds under 5ms at the edge. The system supports 50 concurrent A/B experiments across screen types and user segments without client releases. On cache miss or regional failover, the client falls back to a locally cached UI tree from the previous successful response, applying rule-based DCS adaptation to the cached specification.

7. Evaluation

7.1. Experimental Methodology

All results are from controlled A/B experiments with the following methodology:

- Duration: Six-week experiments per comparison, with a two- week warmup period excluded from analysis to allow ML model convergence.
- Randomization: User-level randomization stratified by market, device tier, and account age. Treatment and control groups were validated for covariate balance using chi-squared tests across 12 pre-treatment variables (all $p > 0.1$).
- Statistical testing: Two-sided bootstrap confidence intervals (10,000 resamples) with Bonferroni correction for multiple comparisons. Median deltas reported.
- Sample sizes: Each experimental cell contained a minimum of 2M users ($\beta > 0.95$ for detecting 2% relative improvements).
- Novelty effect control: Week-over-week metric trends monitored; all reported metrics showed stable or improving trajectories through week six.

7.2. AIDE vs. Main App vs. Lite App

Table 7 presents the three-way comparison on Lite-tier devices.

The feature usage breadth metric is the most significant finding. The Lite App limited users to 1.6 features per session because many features were absent or degraded beyond usefulness. AIDE achieved 2.4 features per session by adapting features to the device rather than removing them, driving the 33% improvement in meaningful interactions and 11% improvement in 7-day retention.

7.3. Regional Impact

Table 8 presents regional impact versus the main app baseline.

7.4. ML Model Ablation

To isolate the ML model's contribution from structural SDUI benefits, we ran a controlled ablation comparing three configurations over four weeks on a held-out 5% population slice. Table 9 presents the results.

SDUI alone (with rule-based selection) already outperforms both baselines, but the ML model adds a further 18% session duration improvement. Behavioral signals (usage patterns, scroll velocity, feature preferences) contribute approximately 40% of the ML model's incremental value, validating the importance of modeling the user, not just the device.

8. Engineering Efficiency and Organizational Impact

8.1. Codebase Unification

Table 10 presents the engineering efficiency gains from codebase unification.

Table 7. Three-Way Comparison on Lite-Tier Devices (Six-Week A/B, Bootstrap Cis, Bonferroni-Corrected). *Data Consumption Higher than lite but delivers substantially more Engagement per MB. **Target Range: 50–65% Sustained Utilization

Metric	Main App	Lite App	AIDE	Δ vs. Lite	*p*-value
Crash rate	4.2%	1.8%	1.2%	−34%	< 0.001
TTI (cold start)	6.8s	3.2s	2.8s	−13%	< 0.001
Session duration	8.2 min	11.4 min	13.9 min	+22%	< 0.01
Sessions per day	3.1	4.6	5.2	+13%	< 0.01
Feature usage breadth	2.1/session	1.6/session	2.4/session	+50%	< 0.001
Meaningful interactions	1.4/session	2.1/session	2.8/session	+33%	< 0.001
Data per session	18.2 MB	4.1 MB	6.8 MB	+66%*	–
7-day retention	52%	61%	68%	+11%	< 0.01
GPU utilization	89%	45%	58%	Optimal**	–

Table 8. Regional Impact vs. Main App Baseline. All Deltas Statistically Significant ($p < 0.01$, Bonferroni-Corrected)

Region	TTI	Crash	Sess.	DAU	Eng/MB
India	−43%	−38%	+24%	+4.8%	+52%
Brazil	−36%	−29%	+18%	+3.4%	+41%
Indonesia	−47%	−44%	+26%	+5.2%	+58%
Nigeria	−55%	−51%	+31%	+6.1%	+67%
Philippines	−42%	−37%	+21%	+4.1%	+48%

Table 9. Ablation Study Isolating ML Contribution. Values are Medians with 95% Bootstrap Cis. Full ML Adds +18% Session Duration over Rule-Based ($p < 0.001$)

Configuration	Session	Interact	7-Day Ret
SDUI + Rule-Based	11.8 min	2.2	63%
SDUI + ML (device)	12.6 min	2.5	65%
SDUI + Full ML	13.9 min	2.8	68%

Table 10: Engineering Efficiency Gains from Codebase Unification.

Metric	Dual Codebase	AIDE
Engineers for EM support	12–18	3–4
Feature parity lag	6–12 months	0 days
Code divergence	35–45%	2.7%
Effort to adapt	3–5 eng-days	0.5 eng-days

Experiment infra	Separate systems	Unified
Incidents/month	2.4	0.6

8.2. Feature Development Workflow

Under AIDE, an engineer building a new feature builds the component once with quality parameters through a standard interface, writes a degradation manifest (a configuration file specifying 3–4 quality levels and resource requirements), registers the component in the SDUI registry, and lets the ML model learn optimal variant se- lection automatically. Emerging market optimization is embedded in the standard development process.

8.3. Tier-Pinned Dogfooding

Engineers use the application at Lite and Minimal tier settings for one week per quarter. A developer tool pins DCS scores to any tier, and a "constraint mode" throttles network speed and limits available memory. This practice catches an average of 4.2 UX issues per quarter that are invisible in metrics.

9. Threats to Validity

9.1. Internal Validity

Selection bias in A/B assignment: User-level randomization may not fully account for household-level device sharing common in emerging markets. We mitigated this by stratifying on account age and activity patterns, but residual confounding may exist. Seasonal and event effects: Our six-week experiments overlap with region-specific events. We validated by running the India experiment across two non-overlapping windows and observing consistent effect directions (±3% variation). ML model co-adaptation: The ML model learns during the experiment. The two-week warmup exclusion mitigates initial transients, but the treatment group may benefit from ongoing model improvement.

9.2. External Validity

Platform specificity: Results are from social media platforms with high daily engagement. Applications with different usage patterns may see different tradeoffs. Market concentration: Our evaluation covers five markets. Results may not generalize to markets with substantially different device distributions or network infrastructure. Temporal validity: The emerging market device landscape evolves rapidly. Optimal tier thresholds may require recalibration.

9.3. Construct Validity

Engagement as proxy for value: Our reward function optimizes for session duration and interaction count, which are imperfect proxies for user value. We partially address this through the "meaningful interactions" component. User satisfaction measurement: Our analysis relies on in-app surveys with inherent response bias.

10. Reproducibility Considerations

Full reproduction requires proprietary infrastructure and data at scale. We identify reproducible aspects:

- Fully reproducible: Signal taxonomy (Table 1), component variant design pattern (Table 3), degradation manifest schema.
- Reproducible with standard tools: Constraint-filtering + con- textual bandit architecture, two-tower model, pairwise hinge loss, IPS correction, Thompson Sampling.
- Reproducible with instrumentation: Network quality measurement, thermal state monitoring, DCS computation algorithm.
- Not reproducible: Absolute metric values, carrier data pricing database, scale-dependent effects.

We plan to release the degradation manifest schema, DCS computation reference implementation, and a synthetic dataset for bandit model development as open-source contributions.

11. Lessons Learned

11.1. From Collaboration Platform to Social Platform

The collaboration platform optimization provided foundational techniques: shader complexity tiering and layout computation deferral transferred directly. Background sync optimization required adaptation for asynchronous notification delivery. Infinite-scroll feed memory management was entirely new—the SDUI component variant system was essential.

While our primary evaluation targets social feed surfaces, we have also applied AIDE's component variant and signal framework to non-feed surfaces including messaging inboxes, transactional flows, and content creation screens, suggesting that the architectural patterns generalize beyond media-rich feed experiences.

11.2. Counter-Intuitive Findings

- Consistency > peak quality: Users preferred stable 30fps over oscillating 60/15fps. The ML model converged on this independently.
- Less navigation = more engagement: 3-tab nav on Minimal devices increased session duration 8%. Less visual clutter reduced cognitive load.
- Text-first feed isn't always worse: In India, prepaid users near data cap showed 31% longer sessions with text-first feed.
- Charging state is the strongest engagement predictor: Con- trolling for all other variables, whether the device is charging predicts 26% higher engagement.83269

11.3. Principles Preserved from Lite Apps

Reliability > Richness: The jank penalty ensures the ML model never sacrifices stability for visual quality. Explicit data control: Three user-selectable data modes preserve user agency. Fast first render: Server-composed skeleton screens render instantly while content loads, matching FB Lite's sub-2-second cold start.

12. Future Work

- Predictive UI pre-composition: Using usage pattern models to pre-compose the next screen's UI tree before the user navigates.
- Cross-app UI coherence: For app families, ensuring adaptive decisions are coherent across apps.
- On-device ML inference: Moving the personalization model to on-device inference for offiine-capable adaptation.
- User-facing adaptation controls: Giving advanced users explicit control over quality-performance tradeoffs.
- Automated perceptual quality testing: ML-based visual quality scoring in the CI/CD pipeline.
- ML-assisted variant generation: Using generative models to propose new component variant configurations within the existing variant schema.

13. Conclusion

Facebook Lite and Instagram Lite proved that emerging markets need adapted interfaces. But the structural cost of parallel codebases— delayed features, duplicated engineering, organizational silos—limits the approach at scale.

AIDE demonstrates a more powerful alternative: a single application whose UI is dynamically composed by the server and continuously optimized by ML models that learn the right interface for every user-device-context combination. By measuring what matters—actual throughput rather than reported network type, real thermal state rather than spec sheet capability, individual data sensitivity rather than market averages—the system delivers personalized experiences that outperform both baselines.

Six-week controlled experiments across 2.4B+ devices confirm: 34% lower crash rates ($p < 0.001$), 41% faster TTI, 22% longer sessions, and 50% broader feature usage versus dedicated lite applications. The ML personalization layer adds a further 18% session duration over rule-based SDUI alone.

The core insight is simple: adaptation belongs in infrastructure, not product code. When every component exposes quality knobs and every screen is server-composed, serving the next billion users is the default behavior of the platform—not a separate workstream.

References

[1] Meta Engineering. Building Facebook Lite: Engineered for the Developing World. *Meta Engineering Blog*, 2016.
[2] Meta Engineering. Reintroducing Instagram Lite: A Lighter Way to Use Instagram. *Meta Engineering Blog*, 2021.
[3] M. Hasan et al. Characterizing the Android Ecosystem in Emerging Markets. In *ACM MobiSys*, 2024.
[4] Y. Liu et al. Adaptive Bitrate Streaming: A Survey. *IEEE Communications Surveys & Tutorials*, 26, 2024.
[5] Google. Android Vitals: Building for Billions. *Android Developer Documentation*, 2025.
[6] A. Pathak et al. Fine-Grained Energy Profiling for Power-Aware Application Design. In *ACM EuroSys*, 2023.

[7] S. Kumar et al. Network Quality Estimation on Mobile Devices. *IEEE Transactions on Mobile Computing*, 23, 2024.
[8] J. Chen et al. Adaptive Rendering for Heterogeneous Mobile GPUs. In *ACM SIGGRAPH Mobile*, 2024.
[9] R. Rosen et al. Thermal-Aware Task Scheduling in Mobile Systems. In *USENIX ATC*, 2023.
[10] P. Aggarwal et al. Understanding Mobile User Behavior in Bandwidth- Constrained Environments. In *ACM CHI*, 2024.
[11] D. Ferreira et al. Battery-Aware Application Design Patterns for Mobile Platforms. In *IEEE PerCom*, 2024.
[12] L. Li et al. A Contextual-Bandit Approach to Personalized News Article Recom- mendation. In *WWW*, 2010.
[13] A. Agarwal et al. Exploration–Exploitation in Mobile App UI Optimization. In *KDD*, 2023.
[14] Meta. Server-Driven Rendering for Mobile Applications at Scale. *Meta Systems Research*, 2024.
[15] W. Enck et al. Memory Management Strategies for Resource-Constrained Android Devices. In *ACM MobiSys*, 2023.
[16] GSMA. The Mobile Economy 2025: Emerging Markets. *GSMA Intelligence Report*, 2025.
[17] D. Agarwal et al. Overlapping Experiment Infrastructure at LinkedIn. In *KDD*, 2017.
[18] M. Dudik et al. Doubly Robust Policy Evaluation and Optimization. In *ICML*, 2014.