

# A Classification of Architectural Trade-Offs in Deploying Generative Models to Mobile Applications Under Device Resource Constraints

 **Vladislav Terekhov**<sup>1</sup>

<sup>1</sup> Mobile Applications Developer, Mobilesource Corp.,  
Miami, United States of America

**ABSTRACT:** Generative models have entered consumer mobile applications as a routine product feature, and their computational profile departs from that of the discriminative models that preceded them on the device. Image synthesis models exceed the memory and arithmetic budgets of a smartphone session, forcing the processing pipeline to span the device boundary. This review proposes a classification of the architectural trade-offs that govern such divided pipelines. Five axes organize the design space: the execution locus of each pipeline stage, the temporal contract the application makes with the user, the boundary that user data crosses, the governance of output quality, and the economics of a single invocation. The classification rests on a criterion that receives limited treatment in the deployment literature. The execution locus of a stage follows from how often a user invokes it within a session, the marginal cost of a remote call, and the technical feasibility of local execution; the latter enters the decision only after the frequency question has been settled. A stage repeated dozens of times within a single session belongs on the device, even when a server could perform it faster. The review also maps the mechanisms that substitute for automated assessment of generative output quality and identifies the absence of a computable proxy for aesthetic acceptability as an open problem. Observations from engineering practice in consumer mobile applications with generative image features supply illustrations for each axis. Those observations are descriptive and carry no controlled measurement.

**KEYWORDS:** generative models, mobile applications, on-device inference, split computing, architectural trade-offs, diffusion models, quality assessment, resource constraints.

## 1. Introduction

Text-to-image generation moved from the research literature to consumer software in a short span of years, and diffusion models became the dominant family driving that shift (Yang et al., 2024). Mobile applications adopted the capability as an ordinary product feature, thereby placing a new class of workload on a platform that had already accumulated a decade of machine learning practice.

A survey of models shipped in real applications on both major mobile platforms describes a population of compact discriminative networks that classify, detect, or segment, with model files small enough to fit within the application bundle (Hu et al., 2023). Generative image models operate on a different order of magnitude in terms of parameters, memory footprint, and arithmetic per output. A consumer smartphone cannot hold and execute

one of them within the thermal and battery budget of an interactive session. The pipeline divides across the device boundary, and the division becomes a permanent condition of the application design.

Research on divided inference has matured. Matsubara et al. (2022) survey split computing and early exiting, in which one network is split into a head executed on the device and a tail executed on a server, with the split chosen under bandwidth and latency constraints. Duan et al. (2023) extend the treatment to orchestration across end, edge, and cloud tiers. Cai et al. (2026) review inference techniques for edge deployment of large generative language models. This body of work treats the division as a placement problem for the layers of a single network. A consumer application presents a problem of another shape. Its pipeline contains several heterogeneous stages, some interactive and some batch, and the engineering question concerns which stage belongs on which side of

the boundary and on what criterion the assignment is made. No systematic classification of that question for consumer generative mobile applications was found in the literature reviewed for this article.

A delimitation from the author's earlier work is required here. That work examined discriminative models executed entirely on Android devices using the provided frameworks and addressed recognition and classification tasks whose models fit within the application (Terekhov, 2024b). The premise of that setting is local sufficiency. The present review addresses the condition in which local sufficiency fails and examines what follows for architecture when a model has to live elsewhere.

The article makes four contributions. First, it proposes a taxonomy of 5 axes along which architectural trade-offs in mobile generative pipelines are resolved. Second, it treats invocation frequency and marginal call cost as the leading criteria for the placement of a pipeline stage, ahead of technical feasibility. Third, it maps the mechanisms that serve as automated quality assessment of generative output and shows which region of the quality space each of them fails to reach. Fourth, it draws illustrations for each axis from engineering practice in consumer mobile applications, using generative image features, with each practice-derived statement marked as such in the text.

## 2. Materials and Methodology

The review covers work published between 2021 and 2026 and retrieved from the ACM Digital Library, IEEE Xplore, and Scopus, supplemented by forward and backward citation tracing from the retrieved items. Query terms combined deployment vocabulary with generative modeling vocabulary, including split computing, edge-cloud inference, on-device inference, diffusion model deployment, text-to-image quality metrics, latency perception, and inference cost. An item was included when it addressed deployment architecture, evaluation of generative output, user response to system latency, or the economics of inference. Items confined to the model architecture or training procedure were excluded, since the classification concerns what happens once a model exists and must be shipped.

Two classes of statements appear in the text below. The first class rests on published sources and carries a citation.

The second-class rests on the author's engineering practice with consumer mobile applications that offer generative image features, and every statement of that class is marked with a phrase attributing it to that practice. The separation matters for a review of this kind because practice supplies the axes of the classification while published work supplies their justification.

The practice-derived statements are descriptive. No controlled experiment accompanies them, and no measured quantity appears anywhere in the article. Configuration parameters such as model input resolution, downscale threshold, polling interval, timeout ceiling, and free-attempt limit are chosen by an engineering team and reported as settings. Durations, failure rates, and comparative improvements are absent from the text because the corresponding measurements were not released for publication. The classification claims descriptive and organizing value, with no claim of empirical validation.

The scope covers consumer mobile applications on Android and iOS whose generative feature synthesizes an image from a user photograph, together with a set of textual settings. Technology names that appear in the illustrations belong to the Android implementation of the pipeline, while the axes themselves carry no platform commitment; the iOS build of the same product followed the same division of work. The application and the organization behind the described practice are unnamed, and the experience is presented as a generic engineering practice involving consumer mobile applications that incorporate generative features.

## 3. Results

### 3.1. Structure of the classification

A trade-off differs from a best practice in that no option along the axis dominates the others across all deployment contexts. Every axis below names a choice of that kind, so a design occupies a position along each of them, and a position that suits one product harms another. Table 1 sets out the 5 axes, the poles that bound each, the variable that drives the decision, and the property the decision surrenders. The order follows the sequence in which the choices arise during development, beginning with the placement of computation and ending with the economics that constrain every earlier choice.

**Table 1. Five axes of architectural trade-off in mobile generative pipelines**

| Axis               | Poles                                              | Leading decision variable                                           | Property surrendered                                 |
|--------------------|----------------------------------------------------|---------------------------------------------------------------------|------------------------------------------------------|
| Execution locus    | On-device, split, cloud-only                       | Invocation frequency per session and marginal cost of a remote call | Model capacity is bounded by device memory           |
| Temporal contract  | Blocking wait, progressive delivery, detached task | Expected completion time measured against user attention            | Detachment can destroy the delivery moment           |
| Data boundary      | Minimal payload, full context                      | Fields strictly required for synthesis                              | Personalization weakens as context is withheld       |
| Quality governance | Automated metric, indirect control                 | Availability of a computable proxy for output quality               | Remote reconfiguration removes build reproducibility |
| Unit economics     | Unmetered access, quota gating                     | Cost per generation measured against revenue per user               | Free-tier gating degrades the first-run experience   |

The fourth column of Table 1 carries the weight of the classification, since a list of aspects without a surrendered property would describe the components of a system and name no trade-off at all.

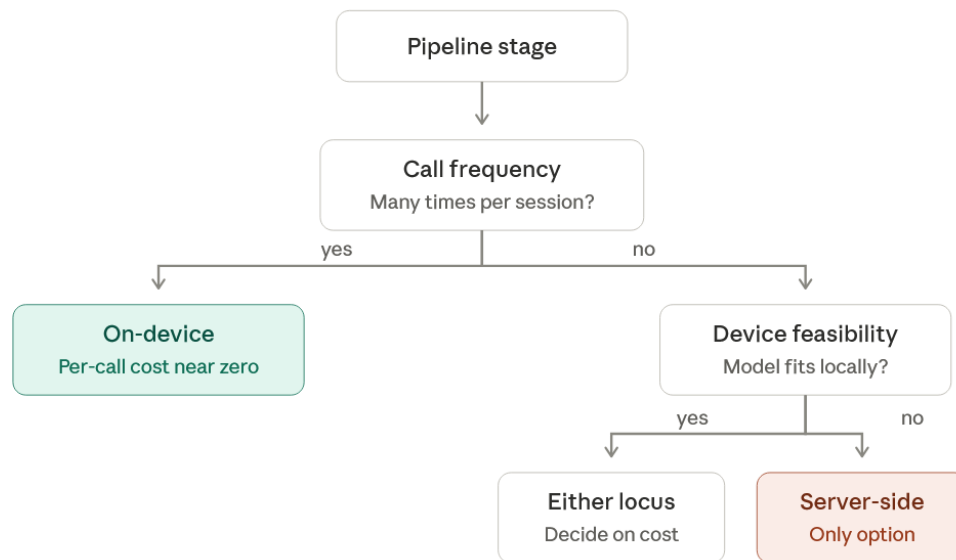
### 3.2. The execution locus of a pipeline stage

The first axis concerns where a stage of the pipeline executes. Its poles are full local execution, a divided pipeline, and full remote execution. Split computing research assigns a stage based on the arithmetic it requires and the size of the intermediate representation that must cross the network (Matsubara et al., 2022). In a consumer application, a second variable dominates the first.

The dominant variable is invocation frequency inside a session, weighted by the marginal cost of a remote call. A stage that a user triggers dozens of times within one session must execute locally, even when a server could perform it faster. Two consequences follow from the remote execution of such a stage. The interaction loses

immediacy because every touch incurs a round-trip network delay and a queueing delay that the user cannot predict. The cost of the feature scales with exploratory behavior, which is the behavior the feature exists to support, so the product pays most for its most engaged users. In the author's engineering practice, this reasoning assigned surface segmentation of a room photograph to the device and kept it there.

Fig. 1 shows the resulting decision procedure. A stage enters the procedure and first meets the frequency test. A stage with high invocation frequency resolves to local execution without further examination. A stage with low frequency proceeds to the feasibility test, and a stage that cannot execute locally resolves to the server. The asymmetry of the two branches carries the substance of the criterion, since technical feasibility enters the decision only after frequency has failed to settle it. A stage that is infrequent and feasible on the device leaves the procedure unresolved, and the cost per invocation decides it.



**Fig. 1.** Decision procedure for selecting the execution locus of a pipeline stage

The local side of the pipeline in the author's practice performs semantic segmentation of the photograph into wall, floor, and ceiling regions using a DeepLab v3 network with a MobileNetV2 backbone, trained on the ADE20K dataset, at an input resolution of 513×513, and executed through LiteRT with a GPU delegate. The segmentation output drives an overlay rendered as the user touches the surface, along with manual mask correction by finger. The same local side prepares the frame for transmission, which includes decoding, applying the EXIF orientation to the pixel data, downscaling to 1280 px on the long side, and JPEG encoding. A local catalog backed by Room holds source images, results, and project state.

The remote side performs one operation: the synthesis of the final image. The application submits a photograph together with the textual settings, receives a task identifier, and awaits completion. The division produces an asymmetric pipeline in which the device handles frequent, inexpensive work and the entire interactive state, while the server performs a single expensive operation, invoked once per delivered result.

The cloud-only pole retains its uses. A product whose generative feature is invoked once per session, whose interactive surface is a form, and whose audience runs old hardware gains little from local execution of any stage. Work on efficient inference at the edge pushes the boundary of what a device can hold (Cai et al., 2026), and each advance move individual stages leftward along this axis without changing the criterion that decides them.

### 3.3. The temporal contract with the user

The second axis concerns what the application asks the user to do while the remote stage runs. Its poles are a blocking wait on a dedicated screen, progressive delivery of partial output, and a detached task that continues to live outside the screen that started it. The middle pole depends on the output modality. A text stream admits partial delivery token by token, while an image produced by an iterative denoising procedure yields intermediate states of low informational value, so the middle pole is weak for the class of application under review.

Classic interaction thresholds treat delay as a cost to be minimized. Recent evidence complicates that position. Tan et al. (2026) report that participants who experienced shorter latencies rated the model output as less thoughtful and less useful than participants who waited longer, thereby turning latency into a design variable with a nonmonotonic effect on perceived quality. Lee et al. (2025) find that users of image generation tools accept waiting as an inherent part of the creative process and interpret longer waits as a signal of a better result. Both findings undercut the assumption that the temporal contract should be shortened wherever possible.

The engineering consequence in the author's practice is a task that lives outside the waiting screen. A project appears in the catalog with an in-progress marker at the moment of submission, the screen can be dismissed, and a notification announces completion. The client polls for readiness at 5 s intervals, with a ceiling of 5 min, after which the task is counted as failed. The user regains the session and spends the wait on something else.

One detail of that design carries more weight than the progress animation. The control that dismisses the waiting screen appears after a 15 s delay. The reasoning recorded in the author's practice is that a result arriving within 10 s makes dismissal harmful, because a user who leaves misses the moment the image appears, and that moment supplies the reason the feature exists. The trade-off along this axis runs between the attention that detachment returns to the user and the delivery moment that detachment can destroy. A delayed-dismissal control resolves the tension by withholding the option during the window when it is harmful.

The generalization available from this observation concerns the shape of the control. An application that offers detachment at once optimizes for the worst-case latency distribution and pays for it in the common case. An application that never offers detachment optimizes for the common case and abandons the user in the tail. The timing of the control is the design variable and depends on the latency distribution of the specific generative backend, which varies with model version and server load.

### 3.4. The data boundary

The third axis concerns what crosses the device boundary. Its poles are a minimal payload that carries only what synthesis requires, and a full-context payload that carries user identity, history, and session state for personalization. Movement toward the minimal pole reduces exposure and weakens every feature that depends on knowing the user.

In the author's practice, the payload includes the photograph along with textual settings covering the tool, style, room type, and palette. Account identity, login credentials, and device identifiers stay on the device, and the remote task is addressed through an anonymous identifier issued for that task. The application holds a private copy of every photograph it processes inside its own storage area, which other applications cannot read, and the photograph reaches the system gallery only when the user saves or shares it.

The private copy addresses a reliability requirement before a privacy requirement. The file permissions the system grants when a user selects a photograph expire when the application restarts, so a project without its own copy of the source image breaks in the catalog on the next launch. The privacy benefit arises from a decision made

for durability. A second effect of the same character follows from re-encoding. The pipeline rebuilds the transmitted image from decoded pixels, so metadata of the source file, including capture coordinates, never leaves the device.

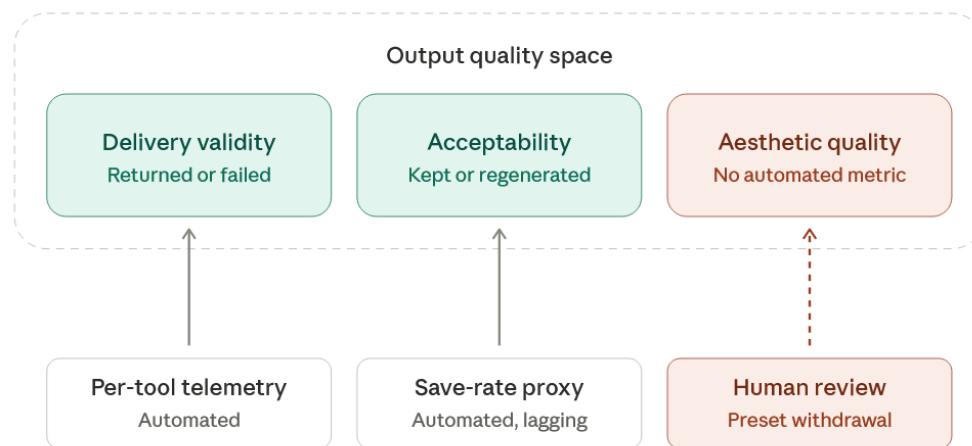
Effects of this kind deserve caution in their description, and the classification records them with a qualification attached. Ganesh et al. (2025) demonstrate a mismatch between the privacy expectation attached to data minimization and the privacy benefit it delivers, since minimization, when measured against a task objective, leaves several facets of real-world privacy risk untouched. An unintended property of a pipeline offers weaker assurance than a designed control because it survives only as long as the implementation detail that produced it. A rewrite of the upload path for performance reasons could restore the metadata without anyone noticing.

### 3.5. Quality governance for non-deterministic output

The fourth axis concerns how an application governs the quality of output, which varies with each invocation. Its poles are an automated metric computed over the output and indirect control through signals that correlate with quality. The axis has no counterpart in the previous generation of mobile machine learning, where a discriminative model produced a label that could be compared against the ground truth, and where a regression in accuracy appeared on a test set.

Evaluation of text-to-image generation has grown into a field with its own taxonomy of metrics (Hartwig et al., 2025). Tian et al. (2025) survey quality assessment for the same class of models. Both reviews converge on a limitation that matters for deployment. Distributional metrics such as the Fréchet Inception Distance were designed to compare model populations; they show poor agreement with human judgment on a single image and require a reference distribution that no mobile client carries. Learned preference models align better with human ratings and demand computing and model weights beyond the budget of a consumer device.

In the author's practice, there is no automated assessment of the aesthetic acceptability of a generated image, since a phone has no instrument for judging whether a picture looks good. Quality is governed through four indirect mechanisms. Fig. 2 shows which region of the quality space each mechanism reaches and which region remains outside automated coverage.



**Fig. 2.** Coverage of output-quality dimensions by available assessment mechanisms

The first mechanism records a binary technical outcome for every request, resolved as delivered or failed, and aggregates the records per tool. Across a large volume of requests, the statistic reveals which tool fails more often and which runs longer than the others, but it remains silent on whether a delivered image is worth showing. The second mechanism is an explicit regeneration control on the result screen, which captures rejection of a single instance and misses silent abandonment, since a dissatisfied user often closes the application without pressing anything. The third mechanism is behavioral. The application requests a store review after the user has saved a result twice, and a save is treated as acceptance, though a user may also save several candidates to compare them later.

The fourth mechanism is human review coupled with remote configuration. The catalog of styles, palettes, room types, and prompts lives in a remote configuration store outside the application binary. A style that produces unusable output over a period is corrected or withdrawn from the catalog without a release, and this was the principal instrument of quality control in the described practice. The instrument carries a cost that belongs on this axis. A single application version behaves differently at different points in time because the catalog it reads has changed underneath it, and the build ceases to be a reproducible artifact for the purpose of quality investigation. A bug report from three months ago cannot be reproduced against the version it names.

### 3.6. Unit economics as an architectural force

The fifth axis concerns the cost of a single generation and its effect on architecture. Its poles are unmetered

access and quota gating. The axis is often treated as a product concern outside the scope of an architectural description, which misrepresents how decisions are actually made.

Luccioni et al. (2024) measured inference cost across categories of machine learning systems and found image generation to be the most demanding among the tasks examined, with multi-purpose generative architectures exceeding task-specific systems by orders of magnitude, even when controlling for parameter count. A feature built on such a model incurs a per-invocation cost that no amount of client-side optimization can eliminate. The cost is introduced into the architecture as a boundary condition, present before the first line of code.

Several decisions in the author's practice trace back to that condition. Downscaling to 1280 px on the long side before upload reduces the transfer volume, as does the work the server performs on the input. Surface segmentation on the device removes the most frequent step of the whole interaction from the metered path, so the step that a user repeats dozens of times costs nothing per repetition. Polling at a 5 s interval keeps request volume to the task endpoint below what a shorter interval would produce, and a 5 min ceiling prevents a stalled task from polling indefinitely.

The same constraint reaches the product surface. Free use is limited to 2 attempts per day per tool. Beyond that limit, three paths exist. A subscription removes the limit, a rewarded advertisement pays for one attempt in the countries where that mechanism operates, and a subscription screen appears in the remaining cases. Results produced under the free tier carry a watermark, so

the cost of computation is recovered through the distribution of the watermark.

The boundary between technical and product decisions dissolves along this axis. A downscale threshold and a free-attempt limit answer one question: how the cost of generation is covered. An architectural comparison that omits cost per invocation describes an incomplete system, and a design chosen on latency alone can be unshippable on margin.

### 3.7. Interaction among the axes

The 5 axes are coupled, and movement along one of them displaces the position on another. Moving a stage to the server widens the data boundary because the payload must now carry everything the stage consumes, and it lengthens the temporal contract because the stage incurs network latency on top of its own execution time. Reducing cost through downscaling to a low resolution limits the quality the generative backend can achieve from the input, which constrains the fourth axis relative to the fifth. Detaching a task from its screen weakens the behavioral quality signal, because a user who returns through a notification interacts with the result differently from a user who watched it appear, and the save that the third mechanism counts arrives in a different context.

A classification that presented the axes as an orthogonal basis would misdescribe the field. The axes function as a set of coupled decisions, and a design is a point in a space whose dimensions correlate. The practical value of separating them lies in making the coupling visible, so that a team changing a downscale threshold for cost reasons understands that it has also moved the quality ceiling.

## 4. Discussion

### 4.1. Relation to existing research programs

The classification proposed here overlaps with three research programs and departs from each at a definable point. Split computing places a cut within a network under network conditions and evaluates the cut by the accuracy retained relative to the bandwidth consumed (Matsubara et al., 2022). The taxonomy above draws a distinction between heterogeneous stages of a product pipeline under interaction and cost conditions, so the unit of decision shifts from a layer to a user-visible step, and the evaluation criterion shifts from accuracy retained to interaction quality preserved.

Orchestration across end, edge, and cloud tiers optimizes system-level objectives across a population of clients and often assumes an operator who controls the whole path (Duan et al., 2023). A consumer application optimizes the experience of one client and the margin on one account, and it controls neither the network nor the device. Work on efficient inference at the edge pursues the goal of moving more computation onto the device (Cai et al., 2026; Semerikov et al., 2025), and the criterion described in this review sometimes prescribes the opposite movement, since a stage that is expensive and invoked once per session gains nothing from local execution.

The evaluation literature stands apart from all three. Hartwig et al. (2025) and Tian et al. (2025) build taxonomies of metrics for comparing research models, and a deployed application needs a decision procedure for a single image at inference time. The distance between those two purposes is the substance of the fourth axis.

### 4.2. Open problems

Five problems emerge from the axes and remain unresolved in the literature reviewed. First, the field has no computable proxy for the aesthetic acceptability of a generated image that a mobile client can evaluate at delivery time. Existing metrics serve model comparison, and existing preference models serve offline ranking, so a product that must decide whether to show one image to a given user has no instrument at all (Hartwig et al., 2025; Tian et al., 2025). Every mechanism cataloged in Fig. 2 measures something adjacent to the question, leaving the question itself open.

Second, remote configuration of generative catalogs has removed reproducibility from the applications that use it, and no established practice reconciles the rapid withdrawal of a defective preset with the ability to reconstruct the behavior of a past release. A versioned configuration history would restore reproducibility at the cost of storage and complexity, and the trade-off has received no systematic treatment.

Third, benchmarks for divided generative pipelines do not exist. Benchmarks for split computing measure layer placement under network conditions, and a benchmark for a product pipeline would need to capture invocation frequency, cost per call, and interaction quality as a single figure of merit. The absence of such a benchmark keeps

the criterion described in section 3.2 at the level of engineering judgment.

Fourth, the energy behavior of divided generative pipelines on the device side remains uncharacterized. The author's earlier work on power consumption in Android applications addressed a different workload generation (Terekhov, 2024a), and sustained GPU load from repeated on-device segmentation within a session raises thermal and battery-related questions that the reviewed literature does not answer. A stage assigned to the device by the frequency criterion may be paid for through throttling.

Fifth, the cost per invocation is absent from academic architectural comparison. Luccioni et al. (2024) established the magnitude of the quantity for generative workloads, and design-level work continues to compare architectures on latency and accuracy alone, which produces recommendations that a commercial team cannot follow.

#### 4.3. Limitations

The observations that supply the axes come from one class of application, which synthesizes an image from a user photograph in a consumer setting. Generalization to text, audio, and video generation was not tested, and the temporal contract depends on output modality in a way that limits transfer, since a text stream admits progressive delivery that an image does not. The practice-derived statements lack controlled measurement, and the classification rests solely on their descriptive content.

The reviewed literature spans 2021 through 2026 in a field where deployment technique changes across quarters. The feasibility boundary separating local from remote execution will move, and individual assignments made under the criterion of section 3.2 will change with it. The criterion is expected to outlast the assignments it produces, and that expectation is a claim about the field that the article cannot test.

#### 4.4. Directions for future work

Four directions follow from the axes as testable propositions. The frequency criterion allows an experiment that varies invocation frequency across placements and measures abandonment alongside cost per session. The delayed dismissal control allows replication as a controlled study of the timing of delivery, with the delay before the control appearing as the manipulated variable. The behavioral quality proxy can be validated against human ratings of the same outputs,

thereby establishing how much of the acceptability signal a save actually carries. The reproducibility cost of remote configuration can be measured as variance in output quality across a fixed application version over an extended period.

### 5. Conclusion

Generative models arrived in consumer mobile applications under a condition that the previous generation of mobile machine learning never faced. The model does not fit the device, and the pipeline divides. This review organized the resulting design space into 5 axes: the execution locus of each stage, the temporal contract the application makes with the user, the boundary that user data crosses, the governance of output quality, and the economics of a single invocation.

The leading axis rests on a criterion that inverts an intuition carried over from earlier work on divided inference. The execution locus of a stage follows from how often a user invokes it and from what a remote invocation costs; the technical feasibility of local execution enters the decision only after the frequency question has been settled. A stage repeated dozens of times within a single session belongs on the device, even when a server could execute it faster, because remote execution of such a stage degrades interaction quality and scales the feature's cost with exploratory behavior.

The fourth axis exposes the widest gap. Applications that ship generative image features govern quality through a binary technical outcome, an explicit regeneration control, a save-based behavioral proxy, and human review coupled with remote withdrawal of defective presets. None of these mechanisms evaluates whether a delivered image is visually acceptable, and the last one removes reproducibility from the build. A computable proxy for aesthetic acceptability that a mobile client could evaluate at delivery time would close the gap, and no such instrument was found in the literature reviewed for this article.

The classification claims organizing value. Its axes came from engineering practice, and its justification came from published work, it carries no empirical validation. Its use lies in providing a practitioner with a set of questions to ask in a defined order and a researcher with a map of where the unanswered questions lie.

### References

1. Cai, G., Tian, R., Yang, L., Jia, Y., Li, L., & Wang, J. (2026). Efficient inference for edge large language models: A survey. *Tsinghua Science and Technology*, 31(3), 1365–1380. <https://doi.org/10.26599/TST.2025.9010166>
2. Duan, S., Wang, D., Ren, J., Lyu, F., Zhang, Y., Wu, H., & Shen, X. (2023). Distributed artificial intelligence empowered by end-edge-cloud computing: A survey. *IEEE Communications Surveys & Tutorials*, 25(1), 591–624. <https://doi.org/10.1109/COMST.2022.3218527>
3. Ganesh, P., Tran, C., Shokri, R., & Fioretto, F. (2025). The data minimization principle in machine learning. In *Proceedings of the 2025 ACM Conference on Fairness, Accountability, and Transparency* (pp. 3075–3093). Association for Computing Machinery. <https://doi.org/10.1145/3715275.3732195>
4. Hartwig, S., Engel, D., Sick, L., Kniesel, H., Payer, T., Poonam, P., Glöckler, M., Bäuerle, A., & Ropinski, T. (2025). A survey on quality metrics for text-to-image generation. *IEEE Transactions on Visualization and Computer Graphics*, 31(10), 9464–9483. <https://doi.org/10.1109/TVCG.2025.3585077>
5. Hu, H., Huang, Y., Chen, Q., Zhuo, T. Y., & Chen, C. (2023). A first look at on-device models in iOS apps. *ACM Transactions on Software Engineering and Methodology*, 33(1), Article 26. <https://doi.org/10.1145/3617177>
6. Lee, H. M., Yadav, D., Lee, S., Govindarazan, K., Chen, C., & Sundar, S. S. (2025). While we wait... How users perceive waiting times and generation cues during AI image generation. In *Extended Abstracts of the 2025 CHI Conference on Human Factors in Computing Systems* (Article 602). Association for Computing Machinery. <https://doi.org/10.1145/3706599.3719725>
7. Luccioni, A. S., Jernite, Y., & Strubell, E. (2024). Power hungry processing: Watts driving the cost of AI deployment? In *Proceedings of the 2024 ACM Conference on Fairness, Accountability, and Transparency* (pp. 85–99). Association for Computing Machinery. <https://doi.org/10.1145/3630106.3658542>
8. Matsubara, Y., Levorato, M., & Restuccia, F. (2022). Split computing and early exiting for deep learning applications: Survey and research challenges. *ACM Computing Surveys*, 55(5), Article 90. <https://doi.org/10.1145/3527155>
9. Semerikov, S. O., Vakaliuk, T. A., Kanevska, O. B., Ostroushko, O. A., & Kolhatin, A. O. (2025). Edge intelligence unleashed: A survey on deploying large language models in resource-constrained environments. *Journal of Edge Computing*, 4(2), 179–233. <https://doi.org/10.55056/jec.1000>
10. Tan, F. F.-Y., Messerschmidt, M. A., Yin, W., & Nov, O. (2026). The impact of response latency and task type on human-LLM interaction and perception. In *Proceedings of the 2026 CHI Conference on Human Factors in Computing Systems* (Article 14). Association for Computing Machinery. <https://doi.org/10.1145/3772318.3790716>
11. Terekhov, V. (2024a). Approaches to optimizing power consumption of Android applications: Practical methods and tools. *American Research Journal of Computer Science and Information Technology*, 7(1). <https://www.arjonline.org/papers/arjcsit/v7-i1/16.pdf>
12. Terekhov, V. (2024b). Implementation of machine learning in Android applications. *International Journal of Computer (IJC)*, 53(1), 72–79. <https://ijcjournal.org/InternationalJournalOfComputer/article/view/2306>
13. Tian, Y., Liu, Y., Wang, S., & Kwong, S. (2025). Quality assessment for text-to-image generation: A survey. *IEEE MultiMedia*, 32(2), 44–52. <https://doi.org/10.1109/MMUL.2025.3538862>
14. Yang, L., Zhang, Z., Song, Y., Hong, S., Xu, R., Zhao, Y., Shao, Y., Zhang, W., Cui, B., & Yang, M.-H. (2024). Diffusion models: A comprehensive survey of methods and applications. *ACM Computing Surveys*, 56(4), Article 105. <https://doi.org/10.1145/3626235>