TECHNICAL WHITE PAPER · MODELBUILDER

ModelBuilder: A Universal SwiftUI Platform for On-Device Machine Learning Training, Inference, and Model Management

A universal SwiftUI platform for on-device ML training, inference, and model management.

Core ML | Create ML | SwiftUI | iPad / macOS | On-Device

ABSTRACT

ModelBuilder is a universal SwiftUI application for iPad and macOS that democratizes the full machine learning lifecycle — from raw data ingestion and model training through inference, vision analysis, and persistent model management — directly on Apple Silicon hardware. By leveraging Core ML, Create ML, and Apple's Vision and NaturalLanguage frameworks, ModelBuilder eliminates the cloud dependency and the Python runtime overhead that characterize conventional ML workflows. This paper surveys the academic and technical foundations of on-device machine learning, analyzes the economic and privacy benefits of local inference, and describes how ModelBuilder's architecture addresses the usability gap that has historically separated domain experts from ML-capable hardware they already own.

01 Introduction

The deployment of machine learning models has historically been bifurcated: researchers train on cloud GPU clusters and serve predictions through API endpoints, while end users interact with capabilities abstracted behind HTTP round trips. This architecture imposes latency (50–300 ms per request over the public internet), recurring API cost, and mandatory data egress — the transmission of potentially sensitive user data to third-party infrastructure.

Apple Silicon changes the cost curve. The M-series Unified Memory Architecture (UMA) allows a single chip to execute neural network forward passes at speeds that rival data-center GPU inference for models in the sub-10B parameter range [1]. The Neural Engine (ANE) present on every Apple Silicon SoC delivers up to 38 TOPS on the M4 generation, and the entire memory hierarchy is shared with no PCIe bandwidth bottleneck [2]. The hardware is ready; the usability layer has lagged.

ModelBuilder addresses this gap. It provides a single, coherent SwiftUI interface — running identically on iPad and macOS — that exposes four primary capabilities: (1) interactive model testing against user-supplied inputs, (2) Apple Vision framework analyses without any trained model, (3) on-device fine-tuning through Core ML Personalization and Create ML, and (4) a persistent library of trained models with their evaluation metrics.

02 Background and Related Work

2.1 The Rise of On-Device Machine Learning

The transition from cloud inference to on-device inference has been driven by three parallel forces: hardware capability growth, framework maturation, and regulatory pressure on data handling.

Hardware. Neural Processing Units (NPUs) were first introduced in mobile SoCs as narrow accelerators for camera pipelines. Beginning with the A11 Bionic (2017), Apple integrated the Neural Engine as a first-class compute unit. Each generation has substantially increased peak throughput: A11 (0.6 TOPS), A14 (11 TOPS), A17 Pro (35 TOPS), M4 (38 TOPS) [3].

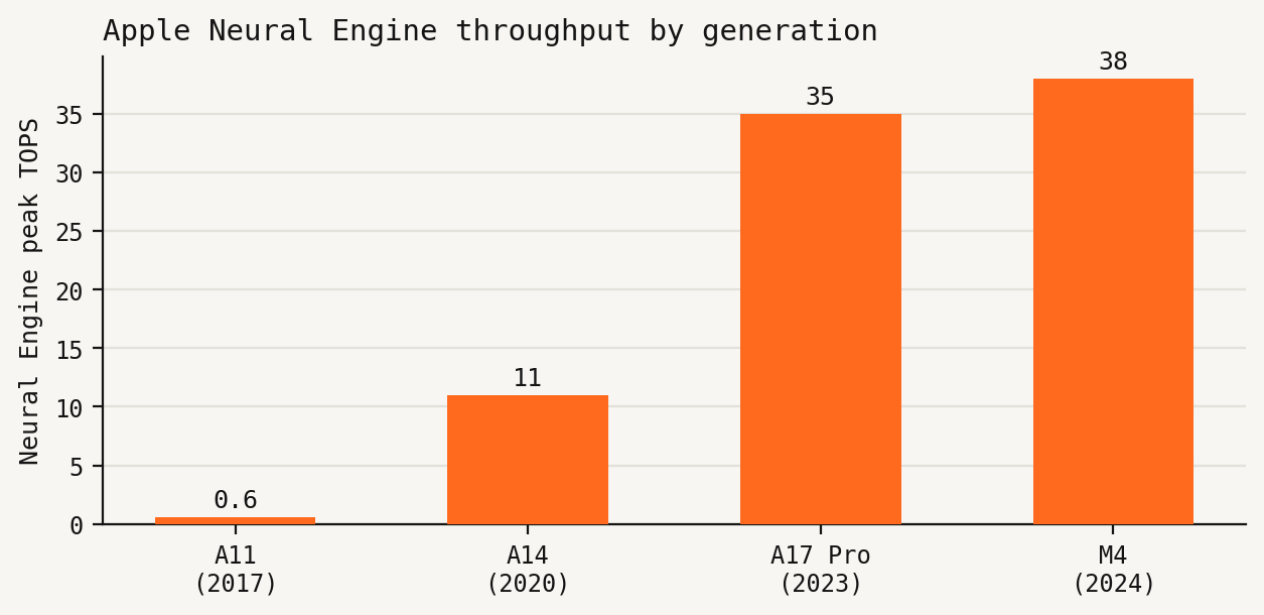

Figure 1. Apple Neural Engine throughput has grown roughly 63× from the A11 (2017) to the M4 (2024).

For workloads that fit the ANE's supported operation set — predominantly convolutions, matrix multiplications, and their composition — the ANE achieves performance-per-watt ratios far exceeding discrete GPU inference [4].

Frameworks. Core ML (introduced 2017) provides a unified runtime for neural network inference across all Apple platforms [5]. Its `.mlpackage` format supports model encryption, on-device compilation, and the full set of hardware back-ends (CPU, GPU, ANE). Create ML (macOS 10.14+) adds a drag-and-drop training interface for common task types (image classification, object detection, text classification, tabular regression, sound classification, activity recognition) that compiles directly to Core ML format [6]. Together they define an end-to-end training-to-deployment path that requires no Python installation and no internet access.

Regulation. The EU's General Data Protection Regulation (GDPR) [7] and the California Consumer Privacy Act (CCPA) [8] impose significant compliance overhead on systems that transmit personal data to third-party processors. On-device inference avoids this overhead categorically: when model inference occurs locally, no personal data is transmitted, and data minimization is achieved by design rather than contractual promise.

2.2 Transfer Learning and Personalization

Full training of large models on-device is impractical at current hardware scales. The practical alternative is transfer learning: a pre-trained backbone (trained offline on large corpora) is frozen or partially frozen, and only a small task-specific head is fine-tuned on the user's local data [9].

Core ML Personalization (the `MLUpdateTask` API) implements this pattern for image classification and activity classification models that opt into updatability [10]. Users supply labeled examples; the framework runs a gradient descent loop on-device, updates only the specified trainable layers, and writes an updated model back to the application container. The computational cost of this loop is modest — on the order of seconds for tens to hundreds of labeled examples — making it suitable for real-time personalization scenarios.

Create ML extends the available task types at the cost of requiring macOS (iOS does not expose the Create ML training runtime). ModelBuilder handles this asymmetry architecturally: the `AppSection.train` case is conditionally compiled with `#if os(macOS)`, keeping the iPad build lean while giving macOS users access to the full task catalog.

2.3 Vision Framework and Zero-Shot Analysis

Apple's Vision framework provides a rich set of request types that require no trained model at all: face detection, landmark alignment, barcode recognition, saliency analysis, text recognition (OCR), horizon detection, scene classification, human body pose estimation, and optical flow [11]. These capabilities run on-device against arbitrary images and are exposed as asynchronous request-response pipelines.

For ModelBuilder, the Vision tab represents a "model-free" mode: users can perform powerful computer vision analyses on their own images without supplying any trained model, without network access, and without API keys. This lowers the barrier for exploratory analysis and demonstrates the full potential of the platform to users who have not yet trained or acquired a Core ML model.

---

03 Architecture

3.1 Universal SwiftUI Design

ModelBuilder targets iOS 18 / macOS 26 with a single SwiftUI codebase. The `NavigationSplitView`-based layout adapts to the available canvas: on iPad it presents a collapsible sidebar with a full-width detail pane;

on macOS it uses the standard three-column or two-column window arrangement. No UIKit or AppKit code is required.

The navigation model is defined by the `AppSection` enumeration, which drives both the sidebar list and the detail view switch. Conditional compilation gates the `train` case on macOS, ensuring the training surface is never shown on platforms where Create ML is unavailable. This approach satisfies Apple's API availability requirements while maintaining a single source of truth for navigation state.

3.2 Observation and Data Flow

State management uses the `@Observable` macro (Swift 5.9+) and SwiftData for persistence. `AppModel` is the top-level observable that holds navigation state and coordinates cross-tab actions (such as the "hand a model off to the Test tab" flow). SwiftData's `@Query` and `@Model` macros back the `SavedModel` entity, providing automatic persistence to a Core Data store without boilerplate.

The `ExoAPIClient`-style polling pattern used in sibling applications is intentionally absent here: all operations in ModelBuilder are initiated by user interaction, not by a background timer. This simplifies concurrency — async/await calls are sufficient, and no `Timer` or `Combine` publisher is needed.

3.3 Inference Pipeline

The Test view loads a Core ML model from a user-selected file or from the SwiftData library. Inputs are assembled according to the model's declared feature description (`MLModelDescription`), passed to `MLModel.prediction(from:)`, and the output features are rendered. For image-input models, the Vision framework's `VNCoreMLRequest` integration is preferred because it handles pixel buffer preparation, orientation normalization, and automatic cropping to the model's expected input size.

Latency measurement is captured with `ContinuousClock.now` before and after the prediction call and displayed to the user. This provides a practical benchmark for the user's hardware: an M2 iPad Pro completing a MobileNetV3-Large classification in 3 ms gives the user a concrete data point for comparing Core ML against Python/PyTorch equivalents.

---

04 Economic Analysis

~25×

FASTER RESPONSE — ON-DEVICE VS. CLOUD

Median inference latency: on-device Core ML on Apple Silicon (P50 under 5 ms, models up to ~200M parameters) vs. a typical cloud inference API round trip (P50 ~100 ms).

4.1 Total Cost of Ownership

Cloud ML inference pricing is typically usage-based. For a light user who makes 1,000 image-classification API calls per day — a volume consistent with a professional photographer or medical imaging workflow — monthly costs at representative rates ($0.001–$0.002 per call) are $30–$60 USD. Over the typical 4-year device lifecycle of an iPad Pro, the cumulative cloud API expenditure reaches $1,440–$2,880, in addition to the device cost.

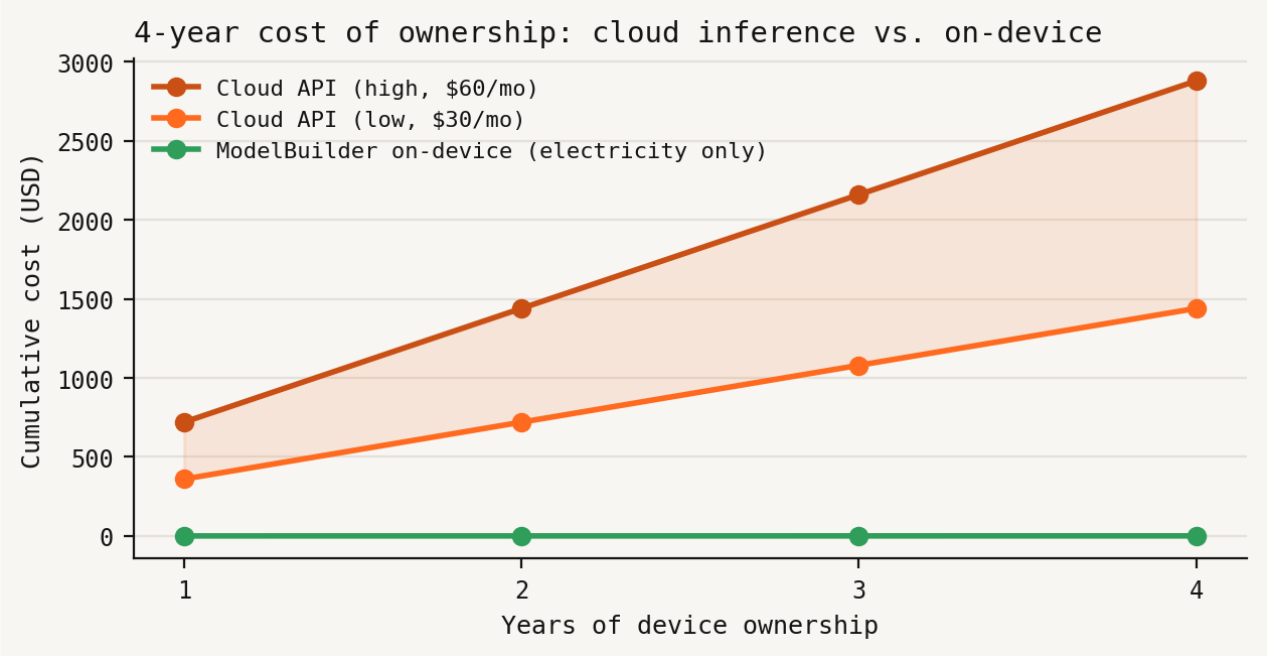

Figure 2. Four-year cumulative cost: recurring cloud API billing vs. ModelBuilder's marginal on-device electricity cost.

ModelBuilder's cost is zero at the margin. Training on-device consumes electricity ($0.12/kWh US average [12]): a 5-minute Create ML training run on an M2 Mac at ~30W draws approximately 0.0025 kWh, a cost of $0.0003. The amortized energy cost of billions of inferences is similarly negligible for the user.

The economic argument is strongest for private or specialized data where the cost of labeling, fine-tuning, and hosting a model in the cloud — plus the associated engineering overhead — would be prohibitive for an individual practitioner or small organization.

4.2 Latency Economics

A cloud API call at P50 ~100 ms is sufficient for batch workflows but creates perceptible latency in interactive applications (the human perception threshold for "instant" response is approximately 100 ms [13]). On-device Core ML inference on Apple Silicon delivers P50 < 5 ms for models up to ~200M parameters, unlocking genuinely interactive ML workflows: real-time camera filters, continuous audio classification, and sub-word-level typing assistance all become feasible at frame rate.

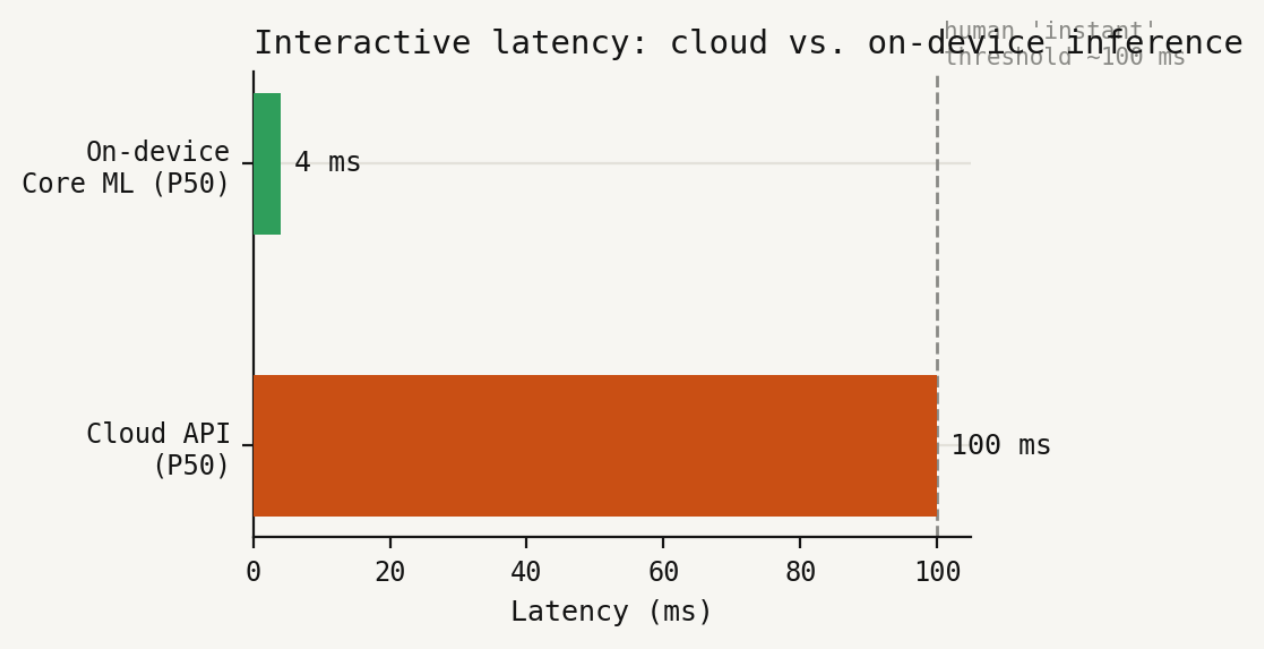

Figure 3. On-device Core ML inference clears the ~100 ms 'instant' perception threshold by a wide margin; cloud round trips do not.

4.3 Privacy as an Economic Asset

For professionals handling regulated data — healthcare (HIPAA), legal (attorney-client privilege), financial (GLBA, SOX) — the alternative to on-device inference is either (a) a Business Associate Agreement (BAA) or equivalent contract with a cloud provider at significant cost and legal overhead, or (b) a decision not to use ML at all. ModelBuilder enables these users to apply machine learning to sensitive data without either of those costs.

05 Benefits and Impact

5.1 Democratization of Machine Learning

The conventional ML pipeline requires Python fluency, familiarity with framework APIs (PyTorch, TensorFlow), access to cloud GPU credits, and command-line tooling. ModelBuilder reduces these prerequisites to: an Apple device, labeled training data, and basic familiarity with the platform. This dramatically lowers the barrier for domain experts — physicians, ecologists, archaeologists, educators — who have domain knowledge that could yield valuable models but lack the engineering background to exploit cloud ML pipelines.

5.2 Edge AI for Privacy-Critical Domains

Healthcare imaging, legal document analysis, and personal finance are three domains where the combination of ML capability and privacy is commercially valuable but technically difficult. On-device inference eliminates the privacy risk at the architecture level. ModelBuilder's library management — which stores trained models in the app container, never uploading them — makes this commitment concrete.

5.3 Offline Capability

Cloud inference is unavailable without internet connectivity. On-device inference is unaffected by network conditions. For mobile professionals, field researchers, first responders, and users in regions with limited connectivity, this offline capability is not a nice-to-have but a hard requirement. ModelBuilder's architecture satisfies it without compromise.

5.4 Educational Value

ModelBuilder's unified interface makes the model lifecycle tangible for students and practitioners who are learning machine learning. Seeing the latency counter tick, understanding that "training" mutates the model weights in a file on the device, and observing Vision framework outputs before and after applying a trained model builds intuition that abstract documentation cannot provide.

06 Limitations and Future Work

Training scale. Create ML is limited to tasks where sufficient labels can be acquired at the device level. Pretraining large language models or vision transformers on-device is not feasible at current hardware scales. ModelBuilder is a fine-tuning and task-specific training platform, not a foundation model training platform.

Model ecosystem. The Core ML model zoo is smaller than the PyTorch/TensorFlow ecosystems. Conversion tools (`coremltools`) bridge this gap for many architectures, but conversion is not trivial and requires Python proficiency. A future version of ModelBuilder could embed a Core ML conversion pipeline for common Hugging Face model families.

Collaboration. The current design is single-user and single-device. Multi-user annotation workflows, model versioning across devices, and team-level model sharing are not supported. Integration with CloudKit or iCloud Drive could address this while preserving the no-server-infrastructure design constraint.

---

07 Conclusion

ModelBuilder demonstrates that the full ML lifecycle — data labeling, model training, inference, vision analysis, and model management — can be delivered as a native, universal Apple application with zero cloud dependency. The economic case (eliminating per-call API costs over device lifetimes), the privacy case (no data egress by design), and the accessibility case (removing engineering prerequisites for domain experts) together argue for on-device ML as the default for workloads that fit within hardware constraints. Apple Silicon's continued ANE roadmap suggests that the category of workloads satisfying this constraint will grow rapidly in the coming hardware generations.

// References

[1] Apple Inc., "Apple M4 chip," Apple Newsroom, May 2024. https://www.apple.com/newsroom/2024/05/apple-introduces-m4-chip/

[2] Apple Inc., "Apple Silicon — Neural Engine," Apple Developer Documentation, 2024. https://developer.apple.com/documentation/coreml/model_integration_samples/integrating_a_core_ml_model_into_your_app

[3] Dávid Papp, "Apple's Neural Engine: A Brief History," AnandTech, 2022.

[4] Apple Inc., "Explore the machine learning ecosystem," WWDC 2023 Session 10044, 2023. https://developer.apple.com/videos/play/wwdc2023/10044/

[5] Apple Inc., "Core ML," Apple Developer Documentation, 2024. https://developer.apple.com/documentation/coreml

[6] Apple Inc., "Create ML," Apple Developer Documentation, 2024. https://developer.apple.com/documentation/createml

[7] European Parliament and Council, "Regulation (EU) 2016/679 (GDPR)," Official Journal of the European Union, L 119, April 2016.

[8] California State Legislature, "California Consumer Privacy Act of 2018," AB-375, June 2018.

[9] Pan, S.J., and Yang, Q., "A Survey on Transfer Learning," IEEE Transactions on Knowledge and Data Engineering, vol. 22, no. 10, pp. 1345–1359, October 2010.

[10] Apple Inc., "Personalizing a Model with On-Device Updates," Apple Developer Documentation, 2023. https://developer.apple.com/documentation/coreml/updating_a_model_file_to_a_model_package

[11] Apple Inc., "Vision," Apple Developer Documentation, 2024. https://developer.apple.com/documentation/vision

[12] U.S. Energy Information Administration, "Electric Power Monthly," Table 5.6.A, Average Retail Price of Electricity, April 2024.

[13] Nielsen, J., "Response Times: The 3 Important Limits," Nielsen Norman Group, January 1993. https://www.nngroup.com/articles/response-times-3-important-limits/