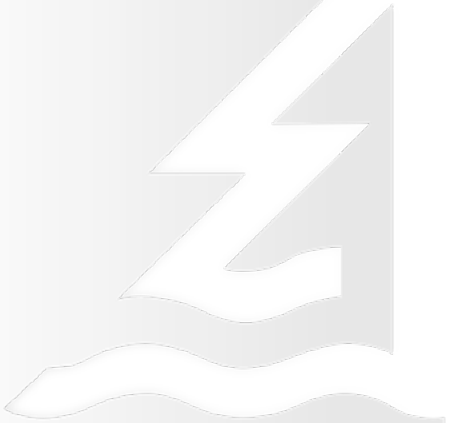
Embedded UI Technologies:

Navigating Hardware Constraints, Performance and Framework Selection



Executive Summary

Selecting the right user interface (UI) technology for embedded systems is no longer a mere software engineering decision; it is a strategic business choice. End-users expect fluid, smartphone-like experiences — 60 FPS animations, high-resolution displays, and intuitive touch interactions — whether they are operating an industrial control panel, a medical ventilator, or a smart home appliance.

Achieving this level of performance requires careful consideration of hardware capabilities, performance goals, development resources, and long-term maintenance. Over-engineering the hardware to support a heavy UI framework inflates the Bill of Materials (BOM) budget. Conversely, selecting a rudimentary UI framework for capable hardware leaves GPU resources unutilized and extends time-to-market due to poor developer tooling.

This guide provides a neutral, platform-oriented overview of the leading embedded UI technologies, with a particular focus on Qt, Slint, LVGL, and Flutter. It maps these solutions across the hardware spectrum — from resource-constrained microcontrollers (MCUs) to high-performance application processors (MPUs).

While KDAB has a long history with Qt and is also a Slint partner, this paper aims to minimize vendor advocacy. Our goal is to help organizations make informed, objective decisions based on their specific hardware platforms, project requirements, and team capabilities.

1. Understanding the Hardware Landscape and UI Implications

Before evaluating software frameworks, it is important to understand the hardware foundation. The embedded hardware spectrum is vast, and the boundaries between Microcontrollers (MCUs) and Microprocessors (MPUs) are increasingly blurring. Hardware platforms still fall into distinct categories, each imposing constraints that materially shape UI technology selection.

1.1 Hardware Platform Categories: The Baseline

PLATFORM CLASS	TYPICAL EXAMPLES	TYPICAL RESOURCES	PRIMARY UI CONSTRAINTS
Microcontrollers (MCU)	STM32H7, NXP i.MX RT1060, Espressif ESP32	1–2 MB Flash, 512 KB – 1 MB RAM, no 3D GPU	Minimal RAM/Flash; relies on software rendering; bare-metal or RTOS only.
Entry-Level MPU	NXP i.MX 6UL/6ULL, TI AM335x	256–512 MB RAM, basic GPU, single/dual Cortex-A	Deployed in highly cost-sensitive products where software efficiency is essential to keep the BOM low. Development cost is a one-time expense and does not directly inflate unit BOM.
Mid-Range MPU	NXP i.MX 8M Nano/ Mini, TI AM62x	1–4 GB RAM, modern GPU, quad Cortex-A53	Good CPU/RAM headroom with entry-level 3D hardware acceleration (i.MX 8M Mini limited to OpenGL ES 2.0). Multi-display is technically supported but memory bandwidth often constrains real-world frame rates. Often bound by industrial temperature ranges.
High-Performance MPU	NXP i.MX 8M Plus, NXP i.MX 95, TI AM62A	2–8 GB RAM, advanced GPU/NPU, Cortex-A55/A53 (1.4–2.0 GHz)	Capable of rich multi-display UIs (with realistic bandwidth expectations), NPU integration for ML, higher power budget.

KDAB Perspective: The Total Cost of Ownership (TCO) Equation

In architectural reviews we frequently see teams selecting a UI framework on developer familiarity alone (e.g., web technologies or Flutter), only to discover late that the framework needs ~2 GB of RAM to run smoothly. Upgrading from a $15 entry-level MPU to a $40 high-performance MPU to absorb software bloat destroys the product's margin. Hardware and UI software must be selected symbiotically.

1.2 Featured Platform Specifications & Architecture Context

To contextualize how UI frameworks behave in the real world, it helps to examine the specific architectures of today's most prevalent embedded silicon.

Note: Many of these SoCs are rarely deployed as custom chip-down designs from day one. Instead, they reach products via robust System-on-Module (SoM) ecosystems from industry leaders such as Toradex, Variscite, and Ezurio. SoM partners accelerate industrial time-to-market and simplify hardware certification for modern IoT devices.

The Texas Instruments AM62x Family

- **Architecture:** Quad Cortex-A53 up to 1.4 GHz plus a 400 MHz Cortex-M4F co-processor, which allows real-time tasks to run independent of the Linux-driven UI.
- **Graphics & Display:** Imagination AXE-1-16M GPU with 2D/3D acceleration; dual LVDS up to 1920×1080 @ 24-bit.
- **UI Impact:** A strong target for declarative, hardware-accelerated frameworks such as Qt, Slint, or Flutter. Multi-display setups are feasible, though real-world frame rates depend heavily on memory bandwidth.

The NXP i.MX 6 Series (The Legacy Standard)

- **Architecture:** Cortex-A7 (528–900 MHz) on 6UL/6ULL; dual or quad Cortex-A9 on 6Dual/Quad; 128–512 MB RAM.
- **Graphics & Display:** No GPU on 6UL/6ULL; Vivante GC2000 on 6Dual/Quad.
- **UI Impact:** Slint or LVGL are the natural fit on 6UL/6ULL; Qt is possible with careful optimization. KDAB has shipped many i.MX 6 projects where Qt runs smoothly when the application is architected well.

The NXP i.MX 8 Series

- **Architecture:** 4× Cortex-A53 cores across the range — up to 1.6 GHz on Nano/Mini, up to 1.8 GHz (1.6 GHz industrial) on Plus. The Plus also integrates a Cortex-M7 co-processor and a 2.3 TOPS NPU.
- **Graphics & Display:** Vivante GC7000UL / GC NanoUltra on Nano/Mini with 1–4 GB RAM; superior Vivante GPU with up to 6 GB RAM on the Plus.

- **UI Impact:** Nano/Mini comfortably host Qt, though heavy frameworks like Flutter still show high CPU utilization. The Plus enables uncompromised design including Qt Quick 3D and ML-driven vision UIs.

NXP i.MX 95

- **Architecture:** 6× Cortex-A55 cores up to 2.0 GHz plus a Cortex-M7 for real-time processing.
- **Graphics & Vision:** 500 MP/s pixel throughput (vs 375 MP/s on the i.MX 8M Plus) and an advanced vision pipeline.
- **UI Impact:** Resource headroom is generous for an embedded SoC, though it still cannot be equated with a workstation-class machine. Framework choice is increasingly driven by developer ecosystem, code sharing, and functional-safety (ISO 26262) needs rather than raw performance.

Microcontroller Platforms

STM32H7, NXP i.MX RT1060, and ESP32 bridge MCUs and entry-level MPUs but top out around 1–2 MB Flash and ~1 MB RAM. Standard Linux UI stacks cannot run here. Specialized, deterministic UI frameworks optimized for software rendering or basic 2D accelerators (e.g., ST's Chrom-ART) are required. LVGL, Slint, and Qt for MCUs are all viable, provided the UI is not overly complex — for example, the Slint printer demo runs on a Raspberry Pi Pico 2, which is less powerful than most boards in this class.

2. Technology Deep Dive: The Frameworks

2.1 Qt: Cross-Platform Application Framework

Overview and Positioning

Qt is a complete, C++-based application platform — not just a UI renderer — with over 25 years of continuous development. It provides tooling, cross-platform deployment, and long-term commercial support (LTS) suited to products with 10-to-15-year lifecycles.

Technology Architecture

- **Qt Quick (QML):** Declarative UI on a hardware-accelerated scene graph.
- **Qt Widgets:** Traditional imperative C++ UI toolkit; relevant for data-heavy or CPU-rendered interfaces.
- **Qt Quick Compiler:** Ahead-of-time compilation of QML to reduce startup time and memory footprint.
- **Qt Quick 3D:** Native integration of 3D content inside 2D interfaces.
- **Qt for MCUs (Qt Quick Ultralite):** A custom QML subset and deterministic runtime targeting microcontrollers.

Platform Suitability & Real-World Scaling

- **High-Performance MPUs (i.MX 8M Plus, i.MX 95, TI AM62A):** Qt is well-suited, with full hardware acceleration via OpenGL ES 2.0/3.0 and established integrations for NPU/ML and GStreamer.
- **Mid-Range MPUs (i.MX 8M Nano/Mini, TI AM62x):** The sweet spot. Qt supports multi-display on these chips; typical moderately complex applications can be optimized to ~90 MB RAM, and the Qt Quick Compiler meaningfully reduces launch time.
- **Entry MPUs (i.MX 6, 6UL/ULL):** Viable with rigorous optimization. Minimize QML nesting depth and render via DRM/KMS directly to the graphics stack (the legacy framebuffer path is generally avoided today). Static linking and bespoke Qt configurations can still help but should be applied selectively rather than by default.
- **Microcontrollers (NXP i.MX RT1060, STM32H7, ESP32):** Qt for MCUs targets deeply constrained devices with a deterministic runtime. Public benchmarks from the

Qt for MCUs team report competitive frame rates and sub-megabyte footprints on the NXP i.MX RT1060; refer to the Qt for MCUs benchmark whitepaper for methodology and figures (see the Qt blog for the Spyrosoft comparative study at https://www.qt.io/blog/qt-for-mcus-vs-lvgl-a-comparative-study-from-design-to-deployment).

KDAB Perspective: Licensing vs. Development Velocity

Commercial Qt licensing is a direct and ongoing cost. The real comparison is between tool ecosystems: Qt Creator and Design Studio, Slint's live-preview tooling, and Flutter's hot reload each accelerate different workflows. Teams should evaluate tool fit against their product rather than assuming any one framework is uniquely productive.

The Business Case: Advantages and Challenges

Strengths: Mature ecosystem; Qt Creator and Qt Design Studio (with a Figma bridge); single-codebase deployment across Linux, RTOS, Windows, iOS, and Android; certification support for IEC 62304 and ISO 26262 via the Qt Safe Renderer (a separate product; these certifications do not cover the full Qt framework); Squish test automation.

Challenges: Large footprint versus minimal frameworks; learning curve around CMake/qmake; licensing complexity for closed-source products or static linking scenarios.

2.2 Slint: The Modern Lightweight Alternative

Overview and Positioning

Slint is a Rust-based declarative UI framework designed from the ground up for extreme efficiency, memory safety, and developer productivity. Slint reports a runtime RAM footprint under 300 KB while still supporting animations, touch input, and hardware-accelerated rendering.

Technology Architecture

- **Declarative UI Language:** A custom, highly readable markup language with deep tooling support.
- **Multiple Rendering Backends:** Software renderer for MCUs, plus OpenGL, Metal, Vulkan, Direct3D, and Skia accelerated renderers for capable MPUs.
- **Language Bindings:** First-class Rust and C++ integrations, with additional bindings for Python and JavaScript/TypeScript.

Platform Suitability & Real-World Scaling

- **Entry MPUs and Microcontrollers:** Slint runs efficiently on MCUs in bare-metal or RTOS environments (FreeRTOS, Zephyr, RT-Thread, QNX, and bare-metal) and on

entry-level Linux MPUs like the i.MX 6UL. When using the Rust binding, the bare-metal workflow integrates directly with Rust's toolchain and memory model. Predictable memory and no garbage collection suit cost-sensitive BOM-efficient designs.

- **Mid-Range MPUs (i.MX 8M Nano, TI AM62x):** Aligns with Slint's sweet spot. GPU-accelerated backends work well where supported; note that wgpu requires OpenGL ES 3.0, which is unavailable on the i.MX 8M Mini and some Nano revisions — plan the backend accordingly.
- **High-Performance MPUs:** Runs flawlessly. With RAM abundant, the primary advantages shift to Rust memory safety, simpler licensing, and cross-language bindings.

KDAB Perspective: The Rust Revolution in Embedded

As a service partner for Slint, KDAB closely monitors adoption. We are seeing growing interest from EV charging and IoT industrial device teams. These projects share tight memory/CPU budgets, long lifespans, and a critical need for software security — areas where Rust shines. Slint allows these teams to ship fluid animated interfaces on hardware that traditionally supported only basic, low-resolution UI.

The Business Case: Advantages and Challenges

Strengths: A ~300 KB runtime, Rust-level memory safety, live preview and hot reload, and a Figma import that is useful if you already have Figma designs (though it is not a primary authoring workflow). The dual GPLv3 + Commercial license is straightforward. Low CPU wake events also make Slint attractive for battery-powered devices.

Challenges: Slint is a younger framework. Third-party widgets and ecosystem depth trail Qt. The core design tools are excellent, but IDE integration and comprehensive documentation for complex edge cases are still maturing, and the certified-domains track record is still in its infancy.

2.3 LVGL: The Microcontroller Specialist

Overview and Positioning

The Light and Versatile Graphics Library (LVGL) is the leading open-source UI framework specifically optimized for microcontrollers. Written entirely in C with minimal external dependencies, it enables sophisticated graphical interfaces on hardware that historically supported only segment or monochrome bitmap displays.

Technology Architecture

- **Pure C Implementation:** Maximum portability across compilers and platforms.
- **Hardware Abstraction:** Pluggable display, input, and filesystem drivers.

- **Rendering Agility:** Highly optimized CPU software rendering with hooks for platform-specific GPU acceleration (e.g., Chrom-ART).
- **RTOS Integration:** Fully compatible with FreeRTOS, Zephyr, RT-Thread, or bare-metal.

Platform Suitability & Real-World Scaling

- **Microcontrollers (Primary Target):** LVGL functions with as little as 64 KB Flash and 16 KB RAM. In typical 128–512 KB Flash / 64–256 KB RAM deployments it delivers feature-rich interfaces with anti-aliasing and smooth scrolling. Published LVGL benchmarks on the NXP i.MX RT1060 show strong frame rates at sub-megabyte footprints (https://lvgl.io/blog/showcase-nxp-imxrt1060-evk)
- **Entry MPUs (i.MX 6UL/ULL):** Highly viable for cost-sensitive applications; minimal footprint preserves resources for application logic.
- **Mid-Range to High-Performance MPUs:** LVGL's value diminishes rapidly. Other frameworks use GPU acceleration more effectively, and manual C cannot match the velocity of declarative QML or Slint.

The Business Case: Advantages and Challenges

Strengths: True MCU optimization, a permissive MIT license with unrestricted commercial use, an active community, and LVGL Pro for visual design.

Challenges: Manual C UI code is slower and more error-prone than declarative approaches. Pure C lacks Rust's memory-safety guarantees, placing the burden of correct allocation and pointer handling on developers. The imperative API becomes hard to maintain as state complexity grows.

2.4 Flutter: The Mobile-First Approach

Overview and Positioning

Flutter is Google's vision for cross-platform development. It is widely adopted among mobile developers (consistently ranking among the top cross-platform mobile frameworks in developer surveys). Its primary focus remains mobile and web; embedded Linux support is capable but community-driven rather than a fully supported first-party effort.

Technology Architecture

- **Dart Language:** Garbage-collected, with JIT (development, hot reload) and AOT (production) compilation.
- **Custom Rendering:** All pixels drawn via the hardware-accelerated Skia engine (transitioning to Impeller), ensuring pixel-perfect consistency.
- **Widget Composition:** An "everything-is-a-widget" philosophy with highly reactive updates.

Platform Suitability & Real-World Scaling

- **High-Performance MPUs (i.MX 8M Plus, i.MX 95, TI AM62A):** Flutter is viable here. Requires 2+ GB RAM and robust GPU acceleration; 60 FPS is achievable.
- **Mid-Range MPUs (i.MX 8M Nano, TI AM62x):** Flutter demands become problematic. In our experience, Flutter applications consume substantially more RAM than equivalent Qt applications on the same hardware; Independent benchmarks should be consulted for specific figures, as results vary significantly by workload and hardware revision.
- **Entry MPUs and Microcontrollers:** Unsuitable. There is no MCU support.

The Business Case: Advantages and Challenges

Strengths: An unparalleled developer experience: hot reload, IDE support, rich built-in widgets (Material/Cupertino), and a massive package ecosystem (pub.dev). The ability to share a codebase between a mobile companion app and the embedded device reduces hiring bottlenecks.

Challenges: Severe resource overhead often forces hardware BOM upgrades. Embedded Linux support is largely community-driven: notable embedders include Toyota's ivi-homescreen (https://github.com/toyota-connected/ivi-homescreen), used in AGL-based automotive infotainment, and flutter-pi (https://github.com/ardera/flutter-pi) for Raspberry Pi and similar boards. Dart's garbage collection introduces unpredictable latency, making Flutter unsuitable for hard real-time requirements.

KDAB Perspective: The Flutter "Hidden Cost" Trap

Because Flutter is BSD-licensed, organizations often assume it is the cheapest route. But if your C/C++ team must learn Dart, and the hardware team has to double RAM and upgrade the GPU to handle Flutter's overhead, the TCO frequently exceeds the cost of commercial licenses for highly optimized frameworks like Qt or Slint.

2.5 Other Notable Frameworks

- **Avalonia UI:** Cross-platform .NET framework (C# / XAML). Can run on Linux via DRM/KMS. Trade-off: .NET runtime overhead; not suitable for MCUs.
- **.NET MAUI:** Microsoft's Xamarin successor. Strong for mobile/desktop; embedded Linux support is effectively community-only. Trade-off: Better for companion apps than the embedded device itself.
- **TouchGFX (STMicroelectronics):** Highly optimized for STM32 with Chrom-ART support and STM32CubeMX integration. Trade-off: Locked to STM32 silicon.
- **Embedded Wizard:** Commercial MCU/MPU framework strong in automotive and white goods. Trade-off: Higher licensing cost.
- **Kanzi (Rightware):** Premium framework dominant in automotive infotainment. Trade-off: Premium licensing.
- **Unreal Engine UMG:** AAA game engine used in high-end signage and luxury EV dashboards. Trade-off: Heavy hardware needs, steep learning curve, large binaries.
- **Crank Software:** A commercial GUI toolchain spanning MPUs and MCUs, marketed for broad silicon support and rapid GUI prototyping; worth evaluating for teams wanting a vendor-agnostic design/runtime pipeline.

3. The KDAB Decision Framework & Implementation Best Practices

3.1 Platform-First Selection Guide

Target Platform Class	Primary Recommendations	Alternative Options	Generally Unsuitable
Microcontrollers (MCU) (STM32, i.MX RT, ESP32)	Slint (software renderer, Rust safety), LVGL (free, C-based), Qt for MCUs (commercial), TouchGFX (STM32 only)	Embedded Wizard, Crank Software	Qt Quick (QML), Flutter, Unreal Engine
Entry MPU (i.MX 6UL, AM335x)	Slint (modern, low footprint), LVGL (permissive license)	Qt Quick (highly optimized), Embedded Wizard, Qt Widgets (only with substantial UX investment)	Flutter, Unreal Engine, Avalonia / .NET
Mid-Range MPU (TI AM62x, i.MX 8M Nano)	Qt Quick, Slint	Flutter (if >2 GB RAM), LVGL (underutilizes GPU), Avalonia (for C# teams)	Unreal Engine (unless 3D is required)
High-Performance MPU (i.MX 8M Plus, i.MX 95)	Qt Quick, Flutter, Slint	Unreal (gaming UI), Kanzi (premium auto), Avalonia (.NET ecosystem)	LVGL (massively underutilizes hardware)

3.2 Decision Criteria Beyond the Silicon

1. Licensing and Business Model

- **Open-source permissive:** LVGL (MIT), Flutter (BSD).
- **Open-source copyleft:** Qt (GPL/LGPL), Slint (GPL). Review static-linking and tivoization implications.
- **Commercial required:** Qt (closed-source/static), Slint (commercial), Qt for MCUs, Embedded Wizard, Kanzi.

2. Team Expertise & Development Velocity

- **C/C++ embedded teams:** Qt, Slint (C++ bindings), or LVGL.
- **Rust teams:** Slint.

- **C# / .NET teams:** Avalonia.
- **Mobile developers:** Flutter (hardware permitting).

Note on Velocity: Declarative frameworks (QML, Slint, Flutter, XAML) generally yield significantly faster iteration than imperative manual coding (LVGL, Qt Widgets).

KDAB Perspective: Common Scenarios

- Industrial HMI on mid-range hardware (e.g., AM62x): Qt Quick for proven comprehensiveness, or Slint for modern efficiency.

- Cost-optimized appliance UI (e.g., i.MX 6UL): Slint (minimal footprint) or LVGL, to maximize resources and minimize BOM.

- Multi-platform consumer device (mobile app + embedded panel): Flutter or Qt Quick — both can share a single codebase across mobile and embedded. Choose based on team language skills and hardware budget.

3. Long-Term Maintenance & Certification

- **Established track record:** Qt (25+ years), LVGL (10+ years), Embedded Wizard.
- **Emerging:** Slint (newer, commercially backed), Flutter embedded (community-driven).
- **Safety-critical (IEC 62304 / ISO 26262):** Qt (commercial) and Embedded Wizard are the conservative choices. Validating Slint is possible but requires investment. Flutter's garbage collection makes hard real-time certification exceedingly challenging.

3.3 Implementation Best Practices: Optimizing Performance

Memory & Graphics Management

- **Deterministic memory:** Static allocation where possible; profile worst-case RAM early; pool frequently created/destroyed objects.
- **Test on target hardware early:** A common and expensive pitfall is developing the entire UI on a workstation and only validating on the target shortly before launch. Run on target silicon from the first sprint, including thermal and boot-time measurements.
- **Asset optimization:** Compress and stream assets. On Qt projects, always use the Qt Quick Compiler.
- **Rendering efficiency:** Minimize overdraw through efficient layer composition. Use hardware acceleration via OpenGL, OpenGL ES, Vulkan, or platform-native APIs wherever available.

Power Efficiency

- Anything that can be suspended when not in use should be — GPU, display controller, sensors, and peripherals. Co-design the UI state machine with the system's power manager.

Startup Time Considerations (The "Cold Boot" Problem)

Application launch time critically impacts the perceived quality of embedded devices.

- **Qt:** The Qt Quick Compiler reduces cold boot by hundreds of milliseconds through AOT compilation.
- **Slint:** Minimal startup overhead thanks to its tiny compiled binary size.
- **LVGL:** Fast initialization with minimal runtime setup.
- **Flutter:** Longer initialization due to Dart VM and framework loading.

Universal optimization: Trimmed dependencies and precompilation contribute to fast boots. Static linking can help but is not a silver bullet — apply it selectively rather than as a default performance lever.

Power Efficiency (Battery-Powered Devices)

- Implement adaptive frame rates based on UI activity (e.g., 60 FPS active, 30 FPS idle, 1 FPS static).
- Anything that can be suspended when not in use should be — GPU, display controller, sensors, and peripherals. Co-design the UI state machine with the system's power manager.
- Batch display updates to reduce memory bandwidth and minimize CPU wake events.

3.4 Future Trends and Considerations

The embedded UI landscape continues to evolve rapidly. Periodic re-evaluation ensures your technology choices remain optimal across long product lifecycles.

- **WebAssembly and Web-Based UIs:** Growing interest in deploying UIs via WebAssembly for dashboards and remote monitoring. Both Qt for WebAssembly and Slint for WebAssembly are viable paths. The trade-off remains portability vs. runtime overhead.
- **Machine Learning Integration:** Modern platforms (i.MX 8M Plus, i.MX 95) increasingly include NPUs, and UI frameworks are integrating with ML pipelines for gesture recognition and predictive interfaces. Qt already offers ML framework bindings.

- **The Rust Ecosystem Growth:** Memory safety without garbage collection is highly attractive for embedded systems. Slint leads as a Rust-native UI framework, and Qt is actively working on experimental Rust bindings.
- **Hardware Evolution:** New MCUs are approaching entry-level MPU performance, blurring the MCU/MPU boundary and enabling richer UIs on traditional MCU budgets. Vulkan adoption is growing, promising more consistent graphics APIs across vendors.

Conclusion: KDAB's Perspective

Selecting an embedded UI technology is an exercise in compromise. There is no single "best" framework — only the best framework for a given combination of hardware, team, licensing posture, and product lifecycle.

- **For microcontrollers:** LVGL offers a proven, permissive-licensed solution. Slint is equally viable for many MCU projects, particularly where Rust memory safety and modern tooling matter. Qt for MCUs provides the fastest development at a commercial cost, and TouchGFX remains strong on STM32.
- **For entry-level MPUs:** Slint's minimal footprint and modern tooling make it compelling for cost-sensitive designs. LVGL remains a strong permissively licensed alternative.
- **For mid-range industrial MPUs:** Qt Quick provides the most mature, comprehensive solution, with Slint offering a lightweight, memory-safe alternative.
- **For high-performance consumer devices:** Qt Quick remains the embedded Linux reference, while Flutter is justified when sharing code with a mobile app offsets its resource overhead. For enterprise teams tied to C#, Avalonia is a credible alternative.

Frameworks that do not suit today's project constraints may become viable as hardware capabilities increase and ecosystems mature, so treat this selection as a decision to revisit — not a one-time choice.

How KDAB Can Help Your Organization

As an independent consultancy, KDAB works with organizations to evaluate UI frameworks objectively, grounded in specific project requirements rather than vendor preference. Our embedded consultants provide:

- Platform-specific performance benchmarking and optimization.
- Framework evaluation and proof-of-concept (PoC) development.
- Migration strategies from legacy UI technologies.
- Custom development and long-term support for Qt and Slint.
- Training and knowledge transfer for embedded engineering teams.

For assistance evaluating UI frameworks for your specific hardware platform, contact KDAB.

About the KDAB Group

The KDAB Group is the world's leading software consultancy for architecture, development and design of Qt, C++ and OpenGL applications across desktop, embedded and mobile platforms. KDAB is the biggest independent contributor to Qt and is the world's first ISO 9001 certified Qt consulting and development company. Our experts build runtimes, mix native and web technologies, solve hardware stack performance issues and porting problems for hundreds of customers, many among the Fortune 500. KDAB's tools and extensive experience in creating, debugging, profiling and porting complex applications help developers worldwide to deliver successful projects. KDAB's trainers, all full-time developers, provide market leading, hands-on, training for Qt, OpenGL and modern C++ in multiple languages.

www.kdab.com