REVIEW ARTICLE

FLUTTER: EMERGING TECHNOLOGY FOR CROSS-PLATFORM DEVELOPMENT

Smita Ketan Hadawale

Pillai College of Arts, Commerce and Science (Autonomous),

New Panvel, M.S., India

*Corresponding author E-mail: smitahadawale@mes.ac.in

DOI: https://doi.org/10.5281/zenodo.17442350

Abstract:

Flutter, Google's open-source UI toolkit, has emerged as a leading option for building cross-platform applications (mobile, web, desktop, and embedded). This research paper provides a structured review of Flutter's recent advancements — Impeller rendering engine, Dart 3 (patterns, records, switch expressions), and WebAssembly (Wasm) support for the web — and analyzes their technical, economic, and educational impacts. By comparing Flutter with alternatives such as React Native and Kotlin Multiplatform, we examine performance, developer productivity, ecosystem maturity, and long-term viability. Identified limitations include WebAssembly's early maturity, package compatibility, and large web binary sizes. A stepwise adoption roadmap is proposed for academic and industrial teams.

Keywords: Flutter, Dart 3, Impeller, WebAssembly, Cross-Platform, Performance, UI, Ecosystem.

1. Introduction:

The app market demands consistent user experiences across Android, iOS, web, and desktop platforms without duplicating codebases. Flutter addresses this need with:

- a high-performance rendering engine (Skia/Impeller),
- a modern programming language (Dart 3),
- unified tooling (hot reload, DevTools),
- compilation to native targets or WebAssembly for the web.

These innovations position Flutter as an "emerging" technology — rapidly evolving while steadily gaining maturity in both industry and academia.

2. Problem Statement and Objectives:

Problem:

To what extent does Flutter, in 2025, provide a better balance of performance, productivity, and platform coverage than alternative cross-platform frameworks?

Objectives:

1. Highlight technical advancements (Impeller, Dart 3, Wasm) from 2024–2025.

2. Assess their impact on performance, developer experience, and maintainability.
3. Identify current limitations and risks.
4. Provide adoption recommendations.

3. Methodology:

- **Literature Review:** Flutter/Dart official documentation, release notes, and architectural overviews.
- **Comparative Analysis:** Qualitative comparison with React Native (JavaScript/Bridge/JSI) and Kotlin Multiplatform (KMP).
- **Feasibility Scenarios:** Conceptual case studies for startups, SMEs, and academic curricula.

4. Recent Technological Landscape:

4.1 Flutter Architecture

Flutter renders UI using its own engine instead of relying on native views. The framework (widgets, layout, gestures) communicates directly with Skia, and increasingly with Impeller, a rendering runtime designed to eliminate shader compilation stutters during runtime.

4.2 Impeller: New Rendering Engine

Impeller precompiles a reduced set of shaders at engine build time, reducing runtime jank and improving frame predictability. Targeting Metal (iOS) and Vulkan/OpenGL (Android), it is optimized for modern GPUs and advanced graphical effects. Benefits include smoother animations, lower cold-start latency, and readiness for GPU-intensive scenarios.

4.3 Dart 3: Expressiveness and Safety

Dart 3 introduces pattern matching**,** records (immutable multi-value returns), exhaustive switch expressions, and class modifiers. Combined with null-safety improvements and advanced type inference, these features enhance code readability and maintainability.

4.4 Flutter Web and WebAssembly (Wasm)

Flutter for the web offers multiple rendering modes: CanvasKit (WebGL/WebGPU) and, more recently, Wasm mode. The --wasm flag activates skwasm, leveraging WasmGC when available and falling back to CanvasKit otherwise. Benefits include faster load times and improved CPU performance with native garbage collection. Current limitations include incompatibility with packages relying on dart:html/JS interop and incomplete browser coverage.

4.5 Tooling and Recent Releases

Versions 3.22–3.24 strengthened the web pipeline, multi-view embedding, and Impeller stabilization. Flutter's release cycle provides predictable updates and stable channels, reducing migration risks.

5. Comparative Analysis:

5.1 Performance and Rendering

- **Flutter (Impeller/Skia):** Predictable frame times, proprietary rendering pipeline, low interop overhead.
- **React Native:** Uses native views; JSI reduces bridge cost but maintains dual JS/native stacks.

- **Kotlin Multiplatform:** Shares business logic; UI remains platform-specific (Compose Multiplatform is maturing, but mobile UIs still require platform-specific work).

5.2 Productivity/Developer Experience

- **Flutter:** Cohesive widget system, fast hot reload, rich package ecosystem, moderate learning curve.
- **React Native:** Leverages JavaScript/TypeScript familiarity; tooling depends on multiple versioned ecosystems.
- **KMP:** Strong for shared logic; higher UI development effort across platforms.

5.3 Web

- **Flutter:** Wasm trajectory promising; CanvasKit provides universal fallback.
- **React Native:** Web support via React Native Web; good for simple apps, but feature parity is limited.
- **KMP:** Compose Web improving but not yet mainstream for complex apps.

6. Limitations and Risks (2025):

1. **Wasm:** Package incompatibility (JS interop), limited browser support for WasmGC. Hybrid strategy recommended.
2. **Web binary size:** Requires optimization (tree-shaking, deferred loading, asset management).
3. **Platform-specific APIs:** Dependent on plugins; package quality/maintenance must be audited.
4. **3D/graphics:** Flutter GPU and 3D support still evolving; not production-ready for advanced use cases.

7. Adoption Scenarios:

- **Startup (B2C):** Single Flutter codebase for mobile + web; deploy CanvasKit for stability, experiment with Wasm in beta.
- **SME (internal tools):** Desktop (Windows/macOS) + tablet support; Flutter desktop viable with unified tooling.
- **University curriculum:** Teach Dart 3 (patterns, records), Flutter architecture, and state management; lab projects on web (Wasm vs CanvasKit).

8. Practical Recommendations:

- **Architecture:** Separate presentation/state/domain; begin with simple state management (Provider/Riverpod), scale to Bloc for larger apps.
- **Performance:** Enable Impeller, profile with DevTools, minimize over-rendering, cache images, test on entry-level devices.
- **Web:** Start with CanvasKit; test --wasm for compatible browsers; monitor package compatibility.
- **Quality:** Use CI/CD (Flutter test, golden tests), static analysis (flutter_lints), and package license checks.

9. Future Outlook (12–24 months):

- Wider adoption of WasmGC in mainstream browsers → faster startup and better runtime performance.

- Expansion of Impeller and GPU APIs to support 3D/advanced visuals in production.
- Growing ecosystem maturity (iOS integration via SPM, multi-view web support, rendering optimizations).

Conclusion:

Flutter is solidifying its role as a comprehensive cross-platform solution. Its latest advancements — Impeller, Dart 3, and Wasm — address critical challenges in performance, code quality, and web support. Despite transitional limitations (especially Wasm maturity and package compatibility), Flutter in 2025 offers a strong balance of speed, UX consistency, and developer productivity for most 2D content-driven apps.

References:

1. Flutter. (2025). *Impeller rendering engine*. Flutter Documentation. https://docs.flutter.dev/perf/impeller
2. Flutter. (2025). *Flutter architectural overview*. Flutter Documentation. https://docs.flutter.dev/resources/architectural-overview
3. Flutter. (2025, May 20). *Flutter 3.24.0 release notes*. Flutter Documentation. https://docs.flutter.dev/release/release-notes/release-notes-3.24.0
4. Flutter. (2025). *Web renderers: CanvasKit and SkWasm*. Flutter Documentation. https://docs.flutter.dev/platform-integration/web/renderers
5. Dart. (2025). *What's new in Dart 3*. Dart Documentation. https://dart.dev/resources/whats-new
6. Ryan, J., & Belanger, M. (2025, June 3). *Dive into Dart's patterns and records*. Google Codelabs. https://codelabs.developers.google.com/codelabs/dart-patterns-records
7. Bizzotto, A. (2023, May 19). *What's new in Dart 3: Introduction*. Code With Andrea. https://codewithandrea.com/articles/whats-new-dart-3-introduction/
8. Vivek, K. (2024, December 25). *Summarized Flutter in 2024 and what's new for 2025*. DEV Community. https://dev.to/3lvv0w/summarized-flutter-in-2024-and-whats-new-for-2025-27gd
9. Flutter. (2025, August 13). *What's new in the docs*. Flutter Documentation. https://docs.flutter.dev/release/whats-new
10. iFlair. (2025, July 24). *Flutter 2025: Powerful updates for cross-platform success*. iFlair Blog. https://www.iflair.com/whats-new-in-flutter-2025-features-updates-and-insights/