JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 36 (2025) 357–364
Received: 20 May 2025
Accepted: 15 June 2025

Comparative analysis of cross-platform application development tools in terms of operating system integration

Analiza porównawcza narzędzi do tworzenia aplikacji wieloplatformowych pod kątem integracji z systemem operacyjnym

Rafał Milichiewicz*, Marcin Badurowicz

Department of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract

This article provides a comparative analysis of modern cross-platform graphical application frameworks, focusing on their ability to seamlessly integrate with the operating system features. Three frameworks (Electron, Tauri and PyQt) were evaluated based on their functional integration with operating system mechanisms, memory usage, and installer size. This research uniquely explores the topic of multiplatform system integration and expands the knowledge on previously untested frameworks. The findings reveal significant limitations in achieving seamless integration and large variations in resource usage.

Keywords: cross-platform; Electron; Tauri; PyQt; comparative analysis

Streszczenie

Artykuł przedstawia analizę porównawczą nowoczesnych, wieloplatformowych szkieletów programistycznych do tworzenia graficznych aplikacji, koncentrując się na ich zdolnościach do bezproblemowej integracji z funkcjami systemu operacyjnego. Trzy szkielety programistyczne (Electron, Tauri oraz PyQt) zostały ocenione pod kątem funkcjonalnej integracji z mechanizmami systemu operacyjnego, zużycia pamięci operacyjnej oraz wielkości instalatora. Niniejsze badania eksplorują temat integracji na wielu platformach oraz poszerzają wiedzę na temat wcześniej nietestowanych szkieletów. Wyniki pokazują istotne ograniczenia w osiąganiu bezproblemowej integracji oraz duże różnice w wykorzystaniu zasobów.

Słowa kluczowe: wieloplatformowość; Electron; Tauri; PyQt; analiza porównawcza

*Corresponding author

Email address: s95541@pollub.edu.pl (R. Milichiewicz)

1. Introduction

In the past decade, the personal computer landscape has experienced a significant shift and diversification. This phenomenon was mainly driven by the growing market share of non-Windows operating systems such as macOS along with various Linux distributions [1] and shift away from x86 architecture towards ARM-based solutions, a particularly significant move from Microsoft with Windows on ARM initiative and Apple's transition to Apple Silicon. As a result, programmers and companies face a serious challenge posed by this emerging trend. Focusing solely on a single platform and architecture may lead to market loss, reduced application reach, and a potential reduction in future revenue.

Consequently, a growing number of software solutions and frameworks are being developed to address the challenges of cross-platform and multi-architecture application development [2]. From the early days of Java and Swing to modern solutions like Electron, cross-platform frameworks have enabled broader reach for software, but this often comes with a new set of challenges, particularly regarding operating system integration.

Although widely used, cross-platform frameworks often struggle to deliver applications that fully leverage native operating system features. Issues such as drag-and-drop support, dialogs, notifications, multi-window management and communication, reactive UI, and distribution reveal the limits of a cross-platform approach. This begs the question: *To what extent can cross-platform application frameworks provide a user experience comparable to that of a native application?*

Existing research tends to focus mainly on mobile cross-application frameworks or on particular framework such as Electron. Comprehensive studies that take into the account the variety of available frameworks and platforms as well as the issue of system integration remain scarce. This paper aims to provide a comparative analysis of cross-platform frameworks from the perspective of system integration while offering insight that can help developers make more informed software stack choices and thus hopefully resulting in higher quality applications, reduced development time, and lower costs.

2. Related works

The concept of "Write Once, Run Everywhere" popularized by Java [3] laid the foundations for early cross-platform development with technologies like Swing, released in 1996, and later JavaFX [4]. Parallel to Java, in 1995, the Qt framework [5] introduced a robust environment for developing cross-platform applications in C++. In later years it gained popularity and bindings for other languages, including Python with official PyQt framework.

In recent years the emergence of frameworks based on web technologies, such as Electron [6], Neutralinojs [7] and Tauri [8], marked a shift in cross-platform strategies. By building on standardized web technologies, these frameworks sidestep the headaches of dealing with each platform's unique issues. Each of them differs in philosophy and solutions to achieve cross-platform experience. Electron ships a stripped-down Chromium engine, whereas Tauri and Neutralinojs rely on the system's integrated WebView.

Microsoft also puts a great emphasis on cross-platform development with its involvement in .NET platform. Both the company's official tool, .NET MAUI (formerly Xamarin) [9], and third-party AvaloniaUI offer unified development of desktop applications.

Some frameworks, such as Flutter [10] from Google and Compose Multiplatform [11] from JetBrains, were created first as tools for creating mobile applications and only later extended their capabilities to target browsers and desktop applications.

Despite the wide range of available technologies, deep integration with native operating system features is rarely a primary focus. Official websites and promotional content tend to focus on simplicity, strong community support, and security, while not emphasizing the importance of cross-platform functionality.

The mobile revolution was triggered by the launches of iPhone in 2007 and the first Android devices in 2008. It introduced a significant operating system fragmentation, similar to the one the desktop landscape experiences now. Early studies [12][13] highlighted the promise, but also the limitations of cross-platform technologies like JQuery Mobile, Adobe PhoneGap and Titanium Mobile SDK. Integration with a given platform and performance shortcomings are one of many problematic areas for such technologies.

Studies [14][15] and [16] repeatedly stress that despite simplifying the development process, cross-platform frameworks deliver applications that feel less polished or less consistent with a given platform compared to native counterparts.

Cross-platform mobile technologies often leverage web technologies, but development is not free from challenges. A study of Stack Overflow and GitHub data [17] revealed difficulties with web library integration, limited tooling (debuggers, testing), API differences, and a lack of native modules, despite efforts to streamline development. However, the authors acknowledge continued potential for these technologies.

Another study [18] compared JavaFX and Electron, evaluating their performance on CRUD operations and outlining their respective strengths and weaknesses. The authors noted the limitations of their study and plan to investigate platform integration further, including features like notification menus and tray icons.

Finally, research [19] compared Electron to a native .NET application, focusing on implementation details such as data transfer between windows and menu creation. Testing was limited to the Windows platform.

While technical documentations and books on framework like Electron [20][21] and Qt [22] provide guidance on integrating with native functionalities like notifications, system tray, file handing and geolocation, this aspect typically receives only a superficial attention compared to principles of given technology and code structure.

Similarly, the aspect of different processor architectures (x86 vs. ARM) in cross-platform frameworks is sparse. Only a few sources, like book [23], mention deployment considerations for platforms like Raspberry Pi, which is one of the most prominent ARM SBCs.

The review reveals the following pattern: while cross-platform frameworks enable broader device target, the issue of seamless system integration remains underexplored. Research on mobile cross-application development highlights both performance and system feature access limitations. Desktop frameworks, especially those based on web technologies, face similar issues. Available resources, such as books, documentations, and tutorials, provide practical guidance, but rarely focus on deep integration with native operating system features and architecture-specific considerations.

3. Aim of the work and research thesis

The primary objective of this research is to evaluate the extent to which modern programming frameworks for cross-platform desktop application development allow seamless integration with the native mechanisms of operating systems they target.

The main research thesis posed in this work formulated as follows: cross-platform desktop application frameworks don't allow for the seamless creation of application that utilizes the integration with the operating system.

To verify and elaborate on this thesis, three research hypotheses have been formulated:

1. H_1 – Cross-platform desktop application frameworks offer limited functional integration with the operating system.
2. H_2 – Cross-platform desktop application frameworks exhibit significant differences in memory usage during runtime.
3. H_3 – Cross-platform desktop application frameworks exhibit significant differences in installer size.

Hypothesis H_1 addresses the functional aspect of the thesis, examining frameworks capabilities. Whereas hypotheses H_2 and H_3 focus on non-functional aspect and aim to detect deficiencies that may indicate imperfect integration with the target system.

4. Methods

For the purpose of this study, three programming frameworks were selected:

1. **Electron** – representing a web-based approach, is currently one of the most popular solutions both in the number of published works, and size of community

(measured by number of stars of project's repository and monthly downloads) [24].

2. **Tauri** – a modern alternative to Electron, built with Rust and security in mind. It utilizes a native system WebView and recently has gained popularity after the 2.0 release [25]. Rust additionally has been recommended by the U.S. government to achieve secure development [26].
3. **PyQt** – representing one of the oldest desktop cross-platform frameworks – Qt. It enables a more traditional approach of building a native-looking graphical application. Additionally, Python has consistently remained at the top three programming languages for many years [27].

Selected frameworks represent vast diversity of programming paradigms, ecosystems, and programming languages. It incorporates both established and innovative solutions.

The research will be conducted on three operating systems: Windows, Linux, and macOS.

Tests will be performed both on devices with x86 and ARM CPUs to ensure comprehensive coverage of available personal computing platforms.

4.1. Evaluation methodology

The application requirements were formulated by integrating standards such as WCAG, Nielsen's usability heuristics, and platform-specific design guidelines for macOS, Windows, GNOME, and KDE. These requirements also address challenging areas identified in the literature review.

Prior to the detailed analysis, each framework underwent an initial evaluation based on a predefined set of necessary conditions detailed in Table 1.

Table 1: Necessary conditions

Condition	Rating
Application launching	+/−
Window appearance consistent with platform guidelines	+/−
Application menu appearance consistent with platform guidelines	+/−

Each condition was evaluated on a binary scale (+ for satisfying the condition and − for not satisfying). Failure to meet all necessary conditions will result in a final score of zero for that specific platform, ensuring that only frameworks capable of meeting all essential criteria were considered for detailed evaluation.

If framework satisfies all necessary conditions, it undergoes the evaluation according to three criteria.

4.1.1. Criterion 1 – Checklist

Based on the previously defined requirements, the checklist has been created and is shown in Table 2.

Table 2: Checklist

Area	Function	Score
Windows	Creating an application menu	
	Displaying multiple windows	
	Multi-window communication	
	Creating system tray icon	
	Displaying context menu	
	Displaying a loading screen	
	Sending a system notification	
Files	Opening a native file selection menu	
	Reading a file	
	Drag and drop functionality	
	Persisting data	
Start-up and installation	Adding application to start-up items	
	Creating an installer for given platform	
	Creating an installer for different platform (cross-compilation)	
Devices and sensors	Accessing connected USB devices	
	Camera usage	
	Retrieving user geolocation	
	Defining and detecting a keyboard shortcut	
	Detecting change in device orientation	
	Playing sounds	
	Microphone usage	

The score for checklist criterion is calculated using the following formula:

$$K_{1\,s,p} = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{m}\sum_{j=1}^{m} a_{s,p,i,j} \qquad (1)$$

where n is number of areas, m is number of functionalities in area i, $a_{s,p,i,j}$ is evaluation of functionality j in area i for framework s on platform p, and $K_{1\,s,p}$ is the final score of Criterion 1 for the framework s for the platform p.

Each functionality is evaluated using the scale:

- **0** – No native mechanism or official plugin is present to enable given functionality or solution presented in documentation does not work,
- **0.5** – Implementation is unstable or there are significant limitations,
- **1** – Implementation is stable and functional.

Thus K_1 ranges between 0 (the worst) and 1 (the best). Based on the functional requirements and accompanying checklist, a detailed graphic design of the application was constructed within Figma. This design serves as a universal reference base, visually outlining the user interface and key interactions to ensure consistent implementation across platforms.

4.1.2.Criterion 2 – Memory usage

In the work [28] comparing native applications to cross-platform frameworks in mobile applications, the authors showed that native applications use less RAM. A programming framework using integration mechanisms with a given platform should therefore have this consumption at the lowest possible level. Memory usage is measured 5 minutes after the application launch to ensure the resource stabilization. RAM usage is sampled at 100 ms intervals over 1 minute. Criterion 2 is expressed as:

$$K_{2\,s,p} = \frac{1}{n}\sum_{i=1}^{n} a_{s,p,i} \tag{2}$$

where n is number of measurements, $a_{s,p,i}$ is RAM usage measurement i for framework s on platform p (in MB), and $K_{2\,s,p}$ is the final score of Criterion 2 for the framework s for the platform p.

4.1.3.Criterion 3 – Installer size

In the work [28], the size of the application was also studied and it was also shown that native applications have a smaller one. The Criterion 3 value is calculated as:

$$K_{3\,s,p} = a_{s,p} \tag{3}$$

where $a_{s,p}$ is the size of installer for framework s on platform p (in MB), and $K_{3\,s,p}$ is the final score of Criterion 1 for the framework s for the platform p.

4.1.4.Final score

The guidelines for ensuring the application's look and feel aim to ensure compliance with platform standards while making the application accessible and usable by as broad a user base as possible. For this reason, the final evaluation of frameworks on a given platform assigns weights to criteria that reflect these priorities: 0.6 for the Checklist Criterion 0.2 for the Memory Usage Criterion, and 0.2 for the Installer Size Criterion. Criteria 2 and 3 were deemed as equally important. The overall evaluation is calculated using the following formula:

$$K_{s,p} = K_{0\,s,p}\,(0.6K_{1\,s,p} + 0.2\frac{K_{2\,p\,min}}{K_{2\,s,p}} + 0.2\frac{K_{3\,p\,min}}{K_{3\,s,p}}) \tag{4}$$

where $K_{0\,s,p}$ is necessary conditions rating (0 or 1), for the programming framework s on platform p, $K_{1\,s,p}$ is checklist score, $K_{2\,p\,min}$ and $K_{3\,p\,min}$ are respectively the lowest RAM usage and installer observed among all frameworks on platform p, $K_{2\,s,p}$ and $K_{3\,s,p}$ are respectively RAM usage installer size for the programming framework s on platform p.

Thus, the final score $K_{s,p}$ ranges from 0 (the worst) to 1 (the best).

4.2. Testing devices and platforms

Tests were conducted using the devices whose specifications are summarized in Table 3.

Table 3: Specification of devices selected for testing

Device	Component	Value
PC	CPU	AMD Ryzen 7 7700X (8 cores)
	GPU	AMD Radeon RX 6700XT
	Storage	SSD 2TB M.2 NVME PCIe 4.0
	RAM	2x16GB DDR5-6000
Raspberry Pi	Model	Raspberry Pi 5
	CPU	Broadcom BMC2712 (4 cores)
	GPU	VideoCore VII
	RAM	8GB LPDDR4
	Storage	SSD 512GB M.2 NVME PCIe 3.0
Mac Mini	Model	Apple Mac Mini 2024
	SoC	Apple Silicon M4 (10 cores)
	RAM	24GB Unified LPDDR5 Memory
	Storage	SSD 512GB NVME PCIe 4.0
MacBook	Model	MacBook Pro 13-inch 2016
	CPU	Intel Core i5-6360U (2 cores)
	GPU	Intel Iris Graphics 540
	RAM	8GB LPDDR4
	Storage	SSD 512GB NVME PCIe 3.0
Surface	Model	Microsoft Surface Pro X
	SoC	Microsoft SQ1 (4 cores)
	RAM	8GB LPDDR4X
	Storage	SSD 256GB M.2 NVME PCIe 4.0

Based on these devices, the platforms included in Table 4 will be tested.

Table 4: Specifications of tested platforms

Platform	Specification	Value
Windows_x86	OS name	Windows
	OS version	Windows 10 22H2
	Graphical environment	Built-in
	Assigned device	**PC**
Windows_arm	OS name	Windows
	OS version	Windows 11 24H2
	Graphical environment	Built into the system

	Assigned device	**Surface**
Linux_x86	OS name	Fedora
	OS version	Fedora 41
	Graphical environment	GNOME and Mutter (Wayland)
	Assigned device	**PC**
Linux_arm	OS name	Raspberry Pi OS
	OS version	Raspberry Pi OS based on Debian 12
	Graphical environment	LXDE and OpenBox (X11)
	Assigned device	**Raspberry Pi**
macOS_x86	OS name	macOS
	OS version	macOS 13 Ventura
	Graphical environment	Built-in
	Assigned device	**MacBook**
macOS_arm	OS name	macOS
	OS version	macOS 15 Sequoia
	Graphical environment	Built-in
	Assigned device	**Mac Mini**

Testing platforms listed in Table 4 cover over 99% of the personal computer operating system market [1] and thus will highlight frameworks' cross-platform capabilities.

Identical versions of each framework and associated tools were installed across all platforms, ensuring reproducible comparisons (details in Table 5).

Table 5: Used software versions

Framework	Details
Electron	Electron 35, Node v22.13.0 (LTS)
Tauri	Tauri 2.4.0, Node v22.13.0 (LTS), Rust 1.84.0
PyQt	PySide 6.8.3, Python 3.13.0

5. Results

All frameworks met the Necessary conditions outlined in Table 1 for every platform. Thus, every framework obtains the score of 1 in this criterion and was evaluated in further criterions.

The Criterion results are presented in Tables 6-9, which detail the scores for Criterion 1,2,3 and the final *K* score across all platform-framework combinations.

Table 6: Criterion 1 scores

Platform	Electron	Tauri	PyQt
Windows_x86	0.78	0.83	0.76
Windows_arm	0.78	0.83	0.73
Linux_x86	0.72	0.67	0.76
Linux_arm	0.73	0.68	0.68
macOS_x86	0.85	0.77	0.76
macOS_arm	0.85	0.77	0.76

Table 7: Criterion 2 scores

Platform	Electron	Tauri	PyQt
Windows_x86	0.05	1.00	0.32
Windows_arm	0.08	1.00	0.32
Linux_x86	0.26	1.00	0.96
Linux_arm	0.16	0.69	1.00
macOS_x86	0.14	1.00	0.57
macOS_arm	0.18	1.00	0.55

Table 8: Criterion 3 scores

Platform	Electron	Tauri	PyQt
Windows_x86	0.03	1.00	0.07
Windows_arm	0.03	1.00	0.09
Linux_x86	0.06	1.00	0.06
Linux_arm	0.36	1.00	0.19
macOS_x86	0.05	1.00	0.09
macOS_arm	0.05	1.00	0.09

Table 9: Final *K* scores

Platform	Electron	Tauri	PyQt
Windows_x86	0.48	0.90	0.53
Windows_arm	0.49	0.90	0.52
Linux_x86	0.50	0.80	0.66
Linux_arm	0.54	0.75	0.65
macOS_x86	0.54	0.86	0.59
macOS_arm	0.55	0.86	0.58

Figures 1-3 contain radar plots that display the scoring distributions for Windows_x86, Linux_arm and macOS_arm platforms, respectively. These visualizations offer insights into how each framework performs in areas of the Checklist outlined in Table 2.

Figure 4 provides a bar chart comparing, the RAM usage of frameworks across platforms.

Finally, Figure 5 provides a bar chart comparing, the size of installers across platforms.

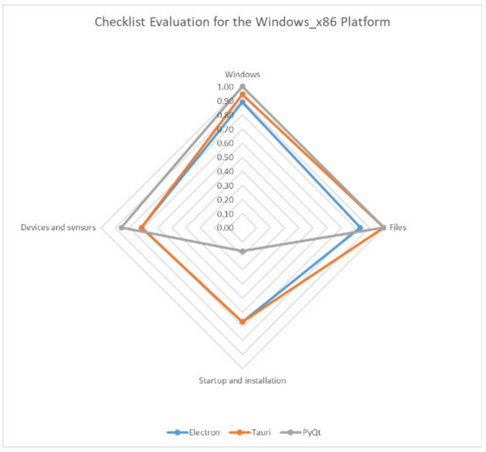


Figure 1: Checklist Evaluation for the Windows_x86 Platform.

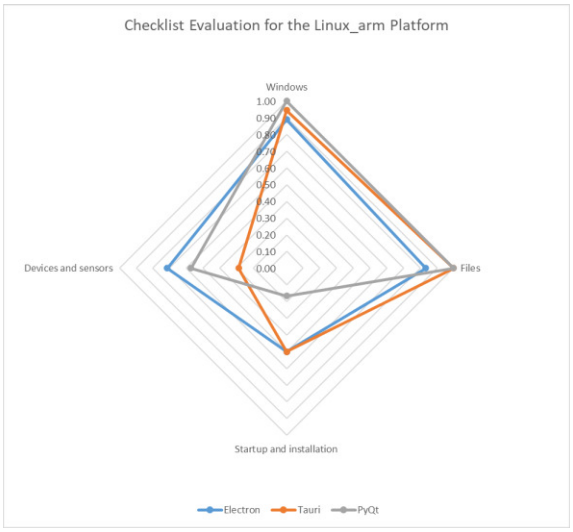


Figure 2: Checklist Evaluation for the Linux_arm Platform.

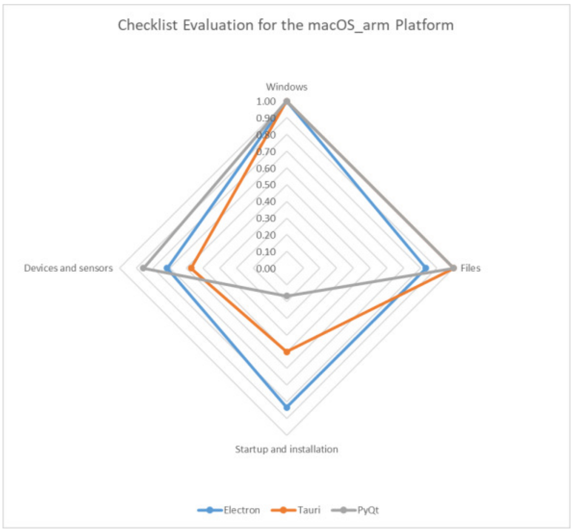
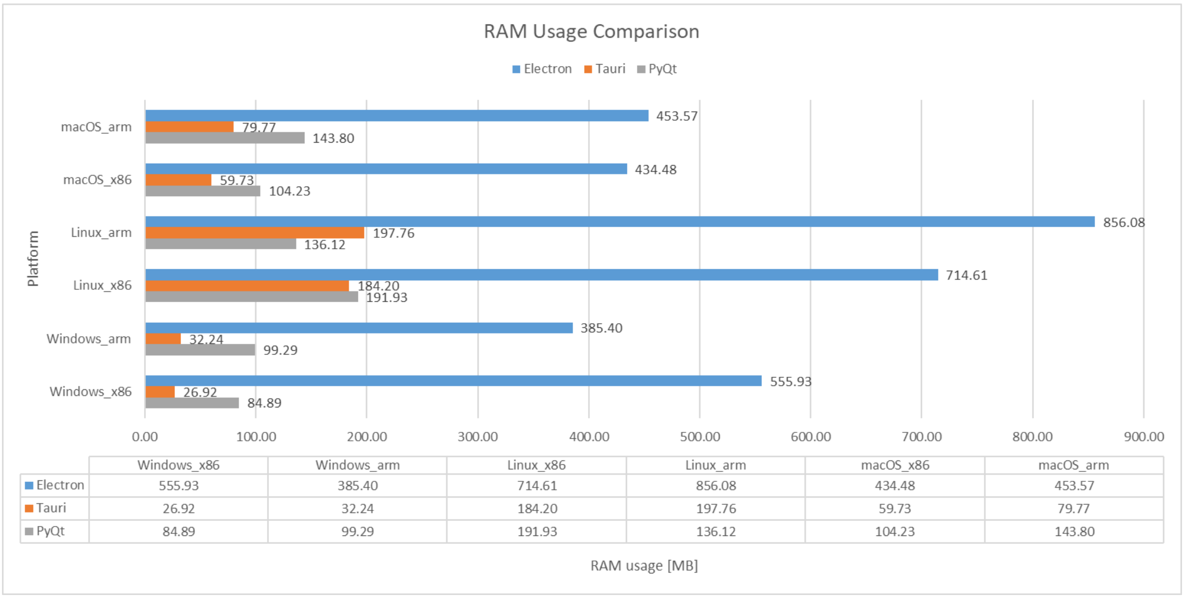
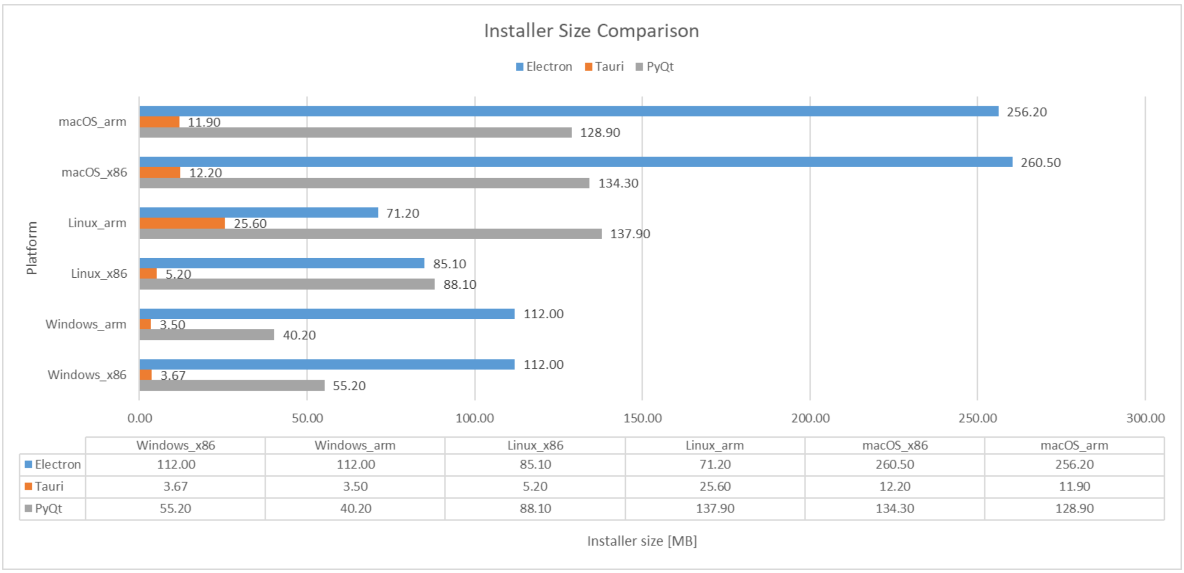


Figure 3: Checklist Evaluation for the macOS_arm Platform.



	Windows_x86	Windows_arm	Linux_x86	Linux_arm	macOS_x86	macOS_arm
Electron	555.93	385.40	714.61	856.08	434.48	453.57
Tauri	26.92	32.24	184.20	197.76	59.73	79.77
PyQt	84.89	99.29	191.93	136.12	104.23	143.80

Figure 4: Bar chart comparison of RAM usage of selected frameworks on selected platforms.



	Windows_x86	Windows_arm	Linux_x86	Linux_arm	macOS_x86	macOS_arm
Electron	112.00	112.00	85.10	71.20	260.50	256.20
Tauri	3.67	3.50	5.20	25.60	12.20	11.90
PyQt	55.20	40.20	88.10	137.90	134.30	128.90

Figure 5: Bar chart comparison of installer size of selected frameworks on selected platforms.

6. Conclusions

Hypothesis H_1 was primarily examined through Criterion 1. The data demonstrated clear disparities between frameworks and platforms. No single framework achieved universal success (the score of 1) across any platform. The closest to that were Electron on macOS (x86 and ARM) and Tauri on Windows (x86 and ARM). PyQt while offering a different approach and many interesting mechanisms, failed to achieve higher scores. The biggest barrier to achieving integration with system was Start-up and installation area. Most frameworks struggled with cross-compilation and sometimes with generating an installer for a given platform. Devices and Sensors area revealed serious shortcomings, especially for Tauri. Relying on system WebView, like Tauri does, can provide advantages, but does not allow for seamless implementation across all platforms. Those findings support H_1, though with the caveat that **some platform-framework combinations can still achieve integration with the operating system, but very inconsistently**.

Hypothesis H_2 was validated by the observed platform-dependent, but consistent discrepancies in memory usage. Electron persistently displayed the highest RAM usage across all platforms. Despite its popularity, the choice to essentially ship a whole web browser, Chromium, leads to significant overhead. In contrast Tauri exhibited most efficient memory usage, particularly on macOS platforms. PyQt and Tauri were relatively close on Linux platforms.

Hypothesis H_3 was proven true by substantial variability in installer size. Tauri consistently produced the smallest installers across all platforms – its largest installer (for Linux_arm) was still nearly one and a half times smaller than the smallest installer for any other framework-platform combination. While Electron's installer sizes were predictably large, due to bundling its runtime environment, it was not always the largest. On Linux platforms, Electron installer sizes were smaller than those of PyQt.

The main thesis is largely supported by the evidence, but not without some caveats. While tested frameworks may support essential functionalities (or in case of PyQt very specific) and enable multi-platform application development, this comes at a trade-off. Seamless integration with the operating system remains **incomplete and inconsistent**. The most substantial evidence was that only a few framework-platform combinations provided high checklist scores. Frameworks that performed well on non-functional aspects, like Tauri on installer size and RAM usage, struggled with some functional aspects.

This research focused on three frameworks – Electron, Tauri and PyQt. While these are representative, they do not capture the sheer number of available solutions and tools. Additionally, checklist could be further expanded with deeper system integration and more specific functionalities (e.g., accessibility or Bluetooth functionality). These aspects could be subject of a future work.

References

[1] Report on the percentage market share of operating systems according to Statcounter, https://gs.statcounter.com/os-market-share/desktop/worldwide/#monthly-200901-202411, [01.05.2025].

[2] Cross-platform software toolkits and environments list, https://en.wikipedia.org/wiki/Cross-platform_software, [01.05.2025].

[3] Sun Microsystems Press release, https://web.archive.org/web/20070310235103/http://www.sun.com/smi/Press/sunflash/1996-01/sunflash.960123.10561.xml, [01.05.2025].

[4] JavaFX – official website, https://openjfx.io, [01.05.2025].

[5] List of Qt releases, https://wiki.qt.io/Qt_version_history, [01.05.2025].

[6] Electron – official website, https://www.electronjs.org, [01.05.2025].

[7] Neutralino.js – official website, https://neutralino.js.org, [01.05.2025].

[8] Tauri – official website, https://tauri.app, [01.05.2025].

[9] .NET MAUI – official website, https://dotnet.microsoft.com/en-us/apps/maui, [01.05.2025].

[10] Flutter – official website, https://flutter.dev, [01.05.2025].

[11] Compose Multiplatform – official website, https://www.jetbrains.com/compose-multiplatform/, [01.05.2025].

[12] A. Holzinger, P. Treitler, W. Slany, Making Apps Useable on Multiple Different Mobile Platforms: On Interoperability for Business Application Development on Smartphones, (eds) Multidisciplinary Research and Practice for Information Systems, CD-ARES (2012) Lecture Notes in Computer Science https://doi.org/10.1007/978-3-642-32498-7_14.

[13] L. Corral, A. Janes, T. Remencius, Potential Advantages and Disadvantages of Multiplatform Development Frameworks–A Vision on Mobile Environments, Procedia Computer Science 10 (2012) 1202–1207, https://doi.org/10.1016/j.procs.2012.06.173.

[14] A. Hammershøj, A. Sapuppo, R. Tadayoni, Challenges for mobile application development, In 14th International Conference on Intelligence in Next Generation Networks (2010) 1–8, https://doi.org/10.1109/ICIN.2010.5640893.

[15] T. Dong, E. Churchill, F. Nichols, Understanding the Challenges of Designing and Developing Multi-Device Experiences, Proceedings of the 2016 ACM Conference on Designing Interactive Systems (DIS '16) (2016) 62–72, http://dx.doi.org/10.1145/2901790.2901851.

[16] P. Kowalczyk, M. Dzieńkowski, Comparison of Performance of Xamarin and Flutter Cross-Platform Frameworks, J. Comput. Sci. Inst. 32 (2024) 199–204, https://doi.org/10.35784/jcsi.6277.

[17] G. Scoccia, P. Migliarini, M. Autili, Challenges in Developing Desktop Web Apps: a Study of Stack Overflow and GitHub, In IEEE/ACM 18th International

Conference on Mining Software Repositories (MSR) (2021) 271–282, http://dx.doi.org/10.1109/MSR52588.2021.00039.

[18] A. Alkhars, W. Mahmoud, Cross-Platform Desktop Development (JavaFX vs. Electron), Bachelor thesis, Linnaeus University, Växjö, 2017.

[19] D. Alymkulov, Desktop Application Development Using Electron Framework: Native vs. Cross-Platform, Bachelor thesis, South-Eastern Finland University of Applied Sciences, Kouvola, 2019.

[20] P. B. Jensen, Cross-Platform Desktop Applications: Using Node, Electron, and NW.js, Manning, Shelter Island, 2017.

[21] D. Sheiko, Cross-platform Desktop Application Development-Electron, Node, NW.js, and React, Packt Publishing, Birmingham, 2017.

[22] Z. L. Eng, R. Rischpater, Application Development with Qt Creator. Build cross-platform applications and GUIs using Qt 5 and C++ - Third Edition, Packt Publishing, Birmingham, 2020.

[23] S. Chin, The Definitive Guide to Modern Java Clients with JavaFX 17: Cross-Platform Mobile and Cloud Development, Apress, New York, 2021.

[24] Electron – official repository, https://github.com/electron/electron, [01.05.2025].

[25] Tauri – official repository, https://github.com/tauri-apps/tauri, [01.05.2025].

[26] White House press release, https://web.archive.org/web/20250118014817/https://www.whitehouse.gov/wp-content/uploads/2024/02/Final-ONCD-Technical-Report.pdf, [01.05.2025].

[27] Stack Overflow 2023 Developer Survey, https://survey.stackoverflow.co/2023/#technology, [01.05.2025].

[28] P. Nawrocki, K. Wrona, M. Marczak, B. Śnieżyński, A Comparison of Native and Cross-Platform Frameworks for Mobile Applications, Computer 54(3) (2021) 18–27, http://dx.doi.org/10.1109/MC.2020.2983893.