http://doi.org/10.35784/iapgos.8358 received: 13.08.2025 | revised: 10.12.2025 | accepted: 06.01.2026 | available online: 30.03.2026

COMPARATIVE ANALYSIS OF WEB DEVELOPMENT FRAMEWORKS IN PHP: CodeIgniter, CakePHP AND Yii

Karol Rak, Mariusz Dzieńkowski
Lublin University of Technology, Department of Computer Science, Lublin, Poland

Abstract. *This study presents a comparative analysis of three PHP frameworks: CodeIgniter, Yii, and CakePHP, focusing on performance, reliability, resource efficiency, stability, and community support. Three identical applications performing CRUD (Create, Read, Update, Delete) operations on a database were developed for testing. The evaluation metrics included request handling speed, CPU and memory usage, error rates under varying workloads (10 to 16,000 virtual users), and framework popularity within the developer community. Experiments were conducted using Apache JMeter and PerfMon in a controlled testing environment. The results indicate that CodeIgniter outperformed Yii and CakePHP in most scenarios, demonstrating the lowest memory consumption, fastest response times, and greater stability under high concurrency. Its lack of an ORM (Object-Relational Mapping), replaced by a procedural QueryBuilder, contributed to this performance advantage. By contrast, Yii and CakePHP, which rely on ORM, required more resources and showed greater CPU instability, particularly under heavy load.*

Keywords: PHP frameworks, performance, reliability, resource usage, CodeIgniter, CakePHP, Yii

ANALIZA PORÓWNAWCZA WEBOWYCH SZKIELETÓW PROGRAMISTYCZNYCH JĘZYKA PHP: CodeIgniter, CakePHP ORAZ Yii

Streszczenie. *W ramach niniejszej pracy przeprowadzono analizę porównawczą trzech frameworków PHP: CodeIgniter, Yii i CakePHP, koncentrując się na wydajności, niezawodności, efektywności wykorzystania zasobów, stabilności i wsparciu społeczności. Na potrzeby badania przygotowano trzy identyczne aplikacje wykonujące operacje CRUD (Create, Read, Update, Delete) na bazie danych. Do porównań wykorzystano następujące metryki: szybkość obsługi żądań, zużycie procesora/pamięci, wskaźnik liczby błędów dla różnych obciążeń (od 10 do 16000 żądań) oraz popularność frameworków wśród społeczności programistów. Badania przeprowadzono za pomocą narzędzi Apache JMeter i PerfMon w kontrolowanym środowisku testowym. Kluczowe ustalenia wskazują, że CodeIgniter przewyższył Yii i CakePHP w większości scenariuszy, demonstrując najniższe zużycie pamięci, najszybszy czas odpowiedzi i lepszą stabilność przy dużej współbieżności. Brak ORM (Object-Relational Mapping) w CodeIgniter, zastąpiony proceduralnym QueryBuilderem, przyczynił się do jego większej wydajności. Z kolei Yii i CakePHP, polegające na ORM, potrzebowały więcej zasobów i wykazywały się większą niestabilnością pracy procesora, szczególnie w warunkach dużego obciążenia.*

Słowa kluczowe: szkielety programistyczne PHP, wydajność, niezawodność, wykorzystanie zasobów, CodeIgniter, CakePHP, Yii

Introduction

Developing software from scratch is time-consuming and requires writing significantly more lines of code compared to using pre-built solutions. Development frameworks offer a wide range of libraries to address various problems commonly encountered in software development, allowing developers to focus more on solving business problems rather than implementation details. Additionally, frameworks establish certain standards and coding practices, which are particularly helpful in team-based projects. Over the years, numerous PHP frameworks have emerged. With the continuous growth of the internet and its user base, increasing emphasis is placed on various application aspects, particularly performance under heavy and fluctuating loads.

The aim of this study is to compare three well-known web development frameworks in PHP. The findings of this analysis are intended to assist developers in selecting a framework, especially one that performs efficiently in basic CRUD operations on databases. The study examines three frameworks: CodeIgniter, Yii, and CakePHP, focusing on their latest versions to evaluate their suitability for modern requirements. The research assesses how applications built on these frameworks perform under prolonged load while ensuring optimal accessibility for users. An additional focus is on memory usage and CPU load, which can be crucial when hardware resources are limited and the software needs to be implemented in the most optimal manner.

The motivation for undertaking this study lies in the fact that many articles evaluate development frameworks based on a single criterion, such as performance, often overlooking other essential aspects. Therefore, the selected frameworks will also be analysed in terms of popularity, support size, and reliability

The four research hypotheses were formulated as part of the study:

H1: *CodeIgniter is the fastest framework in terms of request handling and response generation.*

H2: *CodeIgniter compared to CakePHP and Yii, has the lowest memory usage and CPU load.*

H3: *Yii among the three frameworks, is the most reliable and error-resistant.*

H4: *CakePHP exhibits the most significant stability drops under increasing load.*

1. Related works

Numerous articles analysing various development frameworks, not just for PHP, have been published to date. Article [7] indicates that Laravel handles the highest number of requests with shortest response time. Conversely, article [13] compares Symfony2 and PhalconPHP, noting that PhalconPHP is faster due to its C-based extensions. Study [6] analysed seven popular PHP frameworks, concluding that Laravel and Yii achieved the best performance results, while pure PHP applications, although fast, were difficult to maintain. In article [10], pure PHP, Laravel, and CodeIgniter were compared, with CodeIgniter achieving the best results and proving the most stable in many tests.

In article [14], frameworks were evaluated based on adherence to best programming practices. The Lift framework stood out by offering the most tools and meeting these criteria. Next article [3] provides a guide on how to choose appropriate framework based on project requirements. Another study [5] suggests that Laravel and CodeIgniter are best suited for small to medium projects, while Symfony and Zend are more appropriate for larger ones. Article [15], which compared Laravel and Slim, found that Slim achieved better results due to its minimalist structure.

Article [16] compared tools commonly used for measuring web application performance, namely Apache JMeter and SoapUI. It concluded that JMeter is the better choice due to its broader technology support and higher reliability.

Performance remains the most significant criterion in comparative analyses of frameworks. Article [8] compared PHP, Python, and Node.js performance, revealing that Node.js was the most efficient in I/O operations. Articles also focus on optimization. Article [2] shows that CodeIgniter achieved better throughput and response time due to simple optimization mechanisms, while Laravel was better suited for more complex applications. Regarding resource management, article [4] analysed

resource requirements for various languages, concluding that language choice significantly impacts this aspect.

Study [9] demonstrated that CodeIgniter outperformed pure PHP in CRUD operations in terms of response speed and the number of requests per second. Article [12] analysed CRUD operations in ASP.NET and PHP, showing that ASP.NET was more efficient. Article [17] compared PHP and JSP, indicating that JSP is preferable when application performance is critical. Article [11] examined ASP.NET and PHP to determine which technology is better suited for different application sizes, considering maintenance costs. Article [1] highlighted that while Laravel handles a higher number of requests, Phalcon exhibited shorter response times and required fewer loaded files.

2. Methodology

To conduct performance tests of server applications written in PHP frameworks, three applications performing the same CRUD operations were developed. Despite having identical functionalities, tests were carried out to compare the frameworks in terms of performance, stability, and reliability. The created applications are simple movie management systems. The three identical test applications were built using the following PHP frameworks: CodeIgniter, Yii, and CakePHP. The key factor influencing performance – the main comparative metric – is ORM (Object-Relational Mapping) mechanism. ORM tools convert classes and inter-class relationships defined in code into database tables and entities, enabling database operations via pre-built object methods. This simplifies code and unifies queries, as they can be converted into different SQL dialects. Both Yii and CakePHP have their own ORM implementations, while CodeIgniter uses QueryBuilder, a more script-oriented tool that returns results as arrays (not objects) and requires manual handling of relationships using functions.

2.1. Measurement tools

The study measured query execution speed using Apache JMeter, an open-source tool by the Apache Software Foundation for testing request speeds and monitoring application behavior under varying loads. The primary goal was to determine how quickly each framework-based application processes requests under different user loads (defined by the number of virtual users). Reliability tests were also conducted using JMeter's error reporting capabilities. Additionally, CPU and RAM usage under load were monitored via the Apache JMeter PerfMon extension. Finally, the popularity of the frameworks was compared by analyzing community support, update frequency, and Google Trends popularity indicators**.**

2.2. Test cases

Performance, stability and resource usage were tested using http requests sent by JMeter acting as a client to the server. Each framework were tested using the most common http methods: GET, POST, PUT and DELETE. Every method was tested in eight various loads i.e. 1 request was sent by 10, 100, 500, 1,000, 2,000, 4,000, 8,000 and 16,000 virtual users.

2.3. Test environment

Tests were performed on a single machine running OpenSUSE Tumbleweed, with an AMD Ryzen 7 7730U processor, integrated AMD Barcelo GPU (2 GB allocated), 14 GB DDR4 3200 MHz RAM, a 512 GB PCIe NVMe 4.0 SSD, and a swap partition equal to RAM size. Non-essential processes were terminated to avoid interference. All applications used a MariaDB 11.7.2 database. The frameworks versions were as follows:

- CodeIgniter: 4.0 with PHP 8.1
- Yii: 2.0.45 with PHP 8.1
- CakePHP: 5.1.5 with PHP 8.1

No external web servers (e.g., Apache or Nginx) were required, as the built-in PHP development servers provided by each framework were used. This approach was adopted to ensure that all frameworks were evaluated under their default configurations, which natively include these built-in server environments.

2.4. Tested applications

All three applications used the same database Fig. 1 as a data source. This database schema presents simple movie related system. For testing purposes applications uses a fragment containing data about movies and actors in many-to-many relations so third junction table is also used.

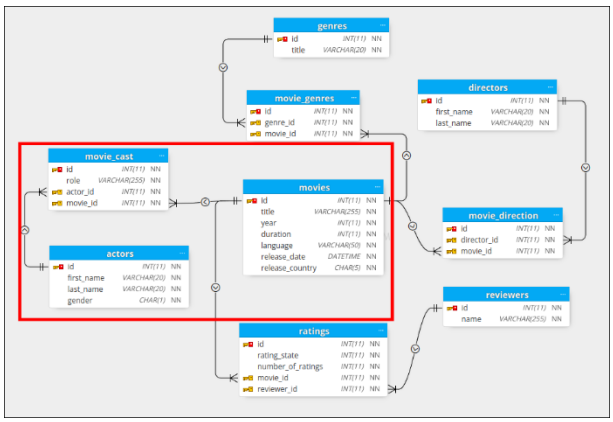

Fig. 1. Database diagram with the area used by the test applications [6]

Data generator was written in form of migration file which populates the data so this database has 20,000 records for movies and actors tables. On the other hand movie_cast has 160,000 records because generator assigns to for each movie record 8 unique records from actors table.

3. Results

This chapter presents comprehensive analysis of performance, reliability and resource usage of provided application in CRUD operations. The results are presented in graphs. The performance tests detailed in the first subsection are focused on average response time of processing related to number of virtual users. The second subsection shows how many errors (measured in percentage) each framework generated. Next section shows how much each application consumed memory performing tests as well as CPU usage graphs. The last one shows various comparisons in terms of popularity and community support such as how many users search for a given framework.

3.1. Performance tests

Figures 2–5 show the results for web application tests implemented on three PHP web frameworks, during which various request packets ranging from 10 to 16,000 were sent. The x-axis indicates the number of virtual users considered in each test, while the y-axis presents the average processing time for a single request by each test application. Trend lines have been added to the graphs to show trends in results for different numbers of virtual users for each test application.

It is important to clarify that, in these tests, the number of virtual users corresponds to the number of requests executed. In JMeter, this is configured by setting both the Number of Threads and the Loop Count to 1, ensuring that each thread executes a single request before the next thread performs the same task. The test concludes once all threads have completed their execution.

Another essential point is that each project was using default options provided by frameworks i.e. each framework was configured to use file based catching mechanisms, so tests not

necessarily were testing raw performance. Instead, they gave insights how this default setup performs in action.

Performance results are quite interesting, because each framework as the load were increasing the performance was getting better. Big performance gains are also visible in 8,000 and 16,000 virtual users scenarios when applications, especially CakePHP application started to generate errors. When those errors started to appear, throughput also was getting higher. This may suggest that server started to skip some requests, allowing to have higher traffic overall. This shows that looking only at one factor can be misleading in this case and we have to look at other factors.

In the context of this study, standard deviations were computed for the measured performance results. However, it was not feasible to depict them on the charts, as the standard deviation values were generally very small (SD < 0.1), reflecting a high degree of consistency among the results. Only in the case of a small number of virtual users were the standard deviations appreciably larger.

GET request

CakePHP in every case achieved the highest processing time, approximately three time slower comparing to the rest when looking at results from 10 to 4,000 virtual users (Fig. 2). On the other hand both Yii and CodeIgniter had similar results but for Yii results in terms of response processing time was slightly higher. CakePHP got closer in processing speeds from 8000 virtual users scenario, but error rate was significantly higher (Fig. 6).

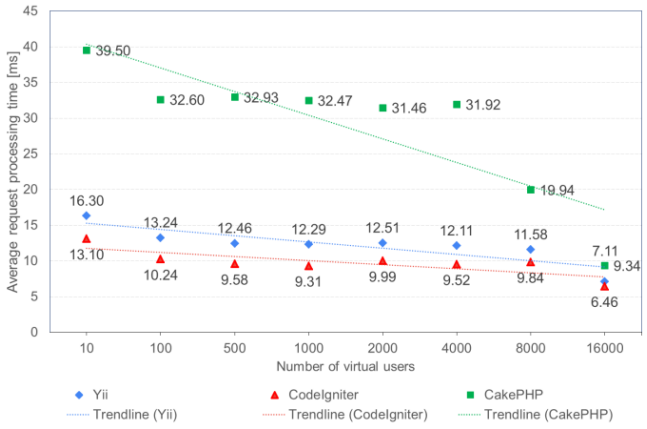

Fig. 2. Performance results for GET request

POST request

For POST request, the Yii and CodeIgniter frameworks achieved similar results as in the case of the GET request (**Błąd! Nie można odnaleźć źródła odwołania.**). In contrast, CakePHP showed consistently longer processing times, which remained nearly unchanged under loads ranging from 10 to 4,000 virtual users.

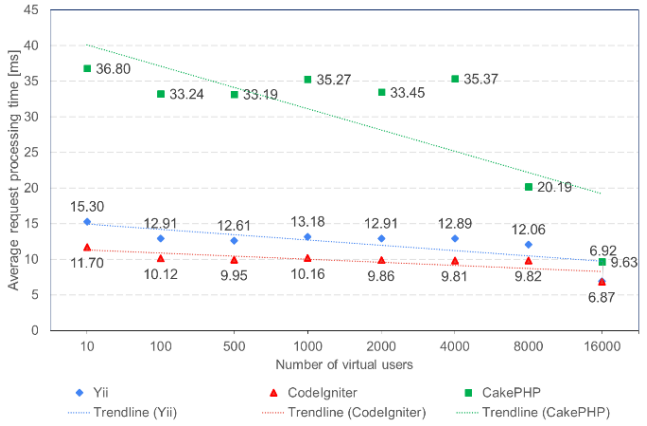

Fig. 3. Performance results for POST request

PUT request

The PUT performance graph confirmed the previous trend (Fig. 4), with the frameworks performing similarly to their behavior on GET and POST requests.

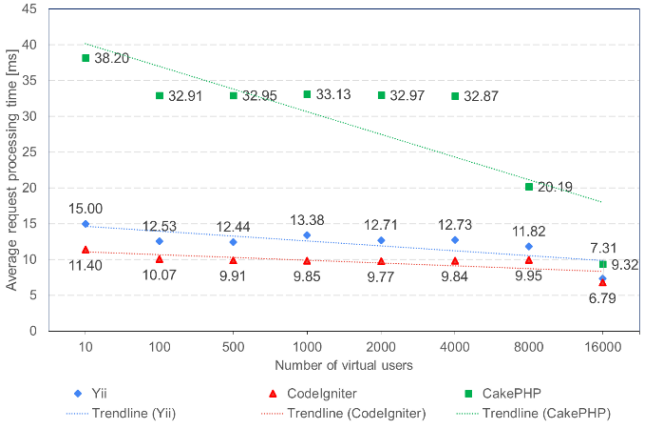

Fig. 4. Performance results for PUT request

DELETE request

In the case of DELETE operation results was also very similar but Yii and CodeIgniter performed slightly better e.g. Yii always stayed below 12 milliseconds starting from 100 requests and CodeIgniter did not reach value close to 10 milliseconds.

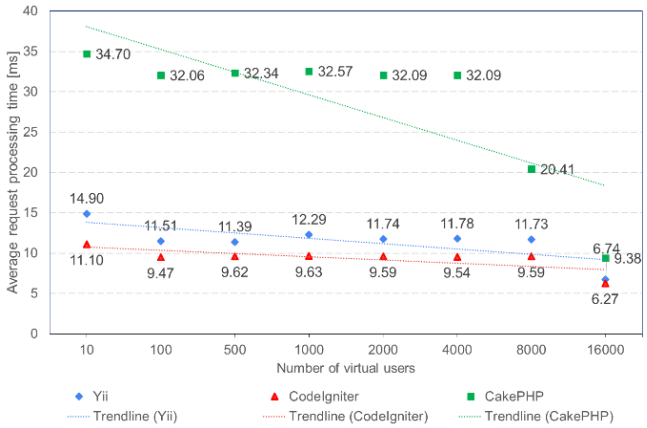

Fig. 5. Performance results for DELETE request

3.2. Reliability tests

Reliability is shown in Fig. 6–9, where the x-axis represents the load (number of virtual users) and the y-axis represents the error rate (percentage of requests not handled correctly).

GET request

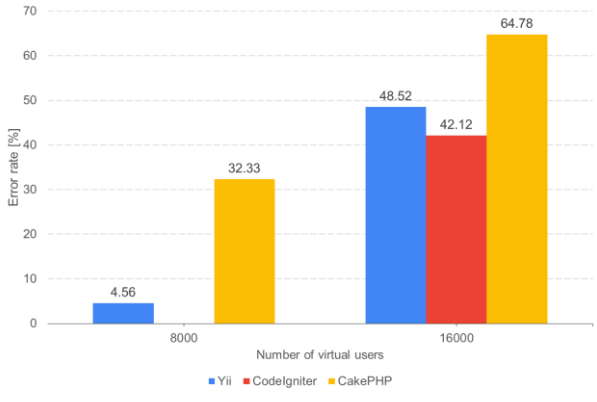

Fig. 6. Reliability test result for GET request

CodeIgniter performed best in terms of reliability (Fig. 6). It showed no errors at a load of 8,000 requests and maintained the lowest error rate at 16,000 virtual users (42.12%). For the Yii and CakePHP frameworks, errors began to appear at 8,000 requests. However, CakePHP performed significantly worse, producing 32.33% errors at 8,000 requests and 64.78% at 16,000, whereas Yii generated 4.56% and 48.52% errors at the same loads.

POST request

Reliability tests for POST operations (Fig. 7) produced results similar to those for GET operations (Fig. 6). The Yii and CakePHP frameworks exhibited errors even at a load of 8,000 requests, with Yii generating 11.74% invalid responses and CakePHP 34.29%. At 16,000 requests, all tested frameworks struggled to handle the load and generated numerous errors. CakePHP had the highest error rate at 64.98%, followed by Yii at 51.92%, while CodeIgniter performed best with the lowest error rate of 41.39%.

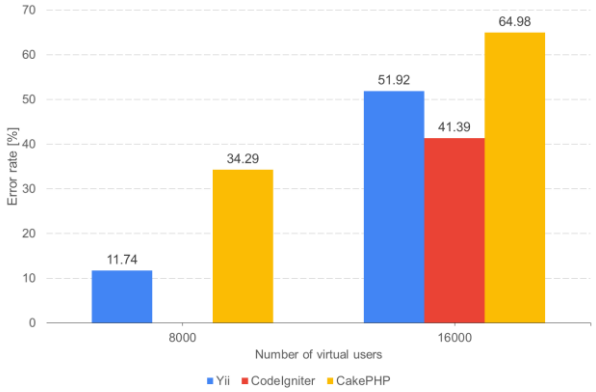

Fig. 7. Reliability test results for POST request

PUT request

Reliability tests for the PUT operation (Fig. 8) produced results very similar to those for the POST operation, with differences not exceeding 2%.

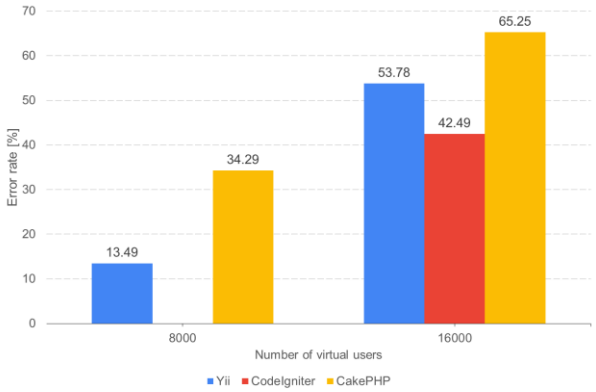

Fig. 8. Reliability test results for PUT request

DELETE request

Reliability results for DELETE operation, presented in Fig. 9, indicate that CodeIgniter again generated no errors at 8,000 requests and maintained the lowest error rate (43.66%) at 16,000 requests. Yii also performed relatively well in this case, showing virtually no errors (0.03%) at 8,000 requests and recording an error rate of 43.66% at 16,000 requests – the lowest it achieved across the previously discussed scenarios (Fig. 6–8).

Additional reliability tests were conducted to determine the point at which each platform begins to generate errors for each CRUD operation. Table 1 summarizes, for each database operation type, the load level (number of virtual users) at which the server application starts to producing errors, assuming an error threshold of less than 1%.

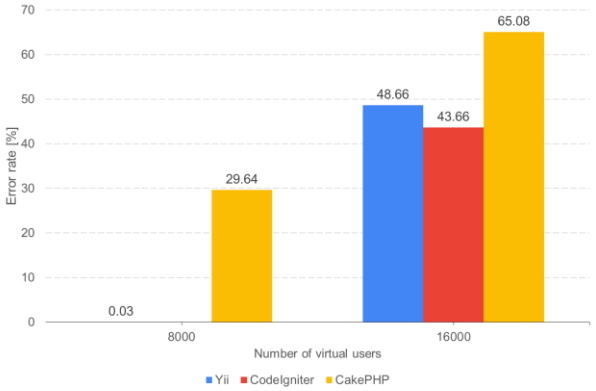

Fig. 9. Reliability test results for DELETE request

Table 1 shows that CodeIgniter performed best under heavy load across all CRUD operations. The number of users it could handle without generating errors ranged from 8,180 for GET to 9,150 for DELETE. In contrast, CakePHP managed only 5,100 virtual users for GET and 5,300 for DELETE.

Table 1. Number of virtual users (server load) at which the server becomes overloaded: summary tested PHP frameworks for CRUD operations

Request type	Yii	CodeIgniter	CakePHP
GET	7300	8180	5100
POST	7360	8250	5150
PUT	7300	8240	5130
DELETE	8000	9150	5300

3.3. Memory and CPU usage tests

The following four graphs present CPU and memory utilization results from a load test with 8,000 virtual users. This load was selected to illustrate how applications use resources over time and how utilization changes as errors begin to occur. For each CRUD operation, two graphs are provided, showing CPU and memory usage for each application during the test. Resource utilization is expressed as a percentage, and time is measured in seconds.

GET request

When performing CRUD operations under heavy load (8,000), server applications built on CakePHP, Yii, and CodeIgniter exhibited significant CPU instability (**Błąd! Nie można odnaleźć źródła odwołania.**).

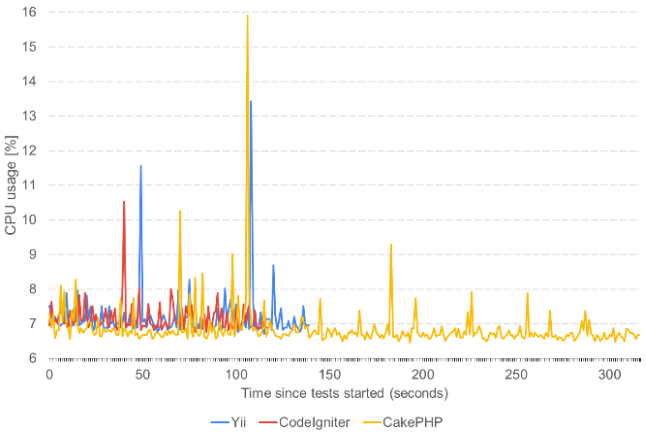

Fig. 20. CPU usage results for GET request

Among the three, CakePHP performed the worst, Yii somewhat better, and CodeIgniter the best. The plotted lines illustrate CPU behavior over time. Overall CPU usage levels were similar across the applications, with CakePHP showing the lowest usage – but at the cost of generating the most errors under load, as it rejected and failed to process many requests.

In terms of RAM consumption, CakePHP achieved the worst result and consumed the most RAM throughout the test (Fig. 11). Yii used the least memory in this scenario, although CodeIgniter used one percentage point more memory at the beginning of the test, gradually approaching Yii's usage. At the end of the test, both frameworks used a very similar amount of memory.

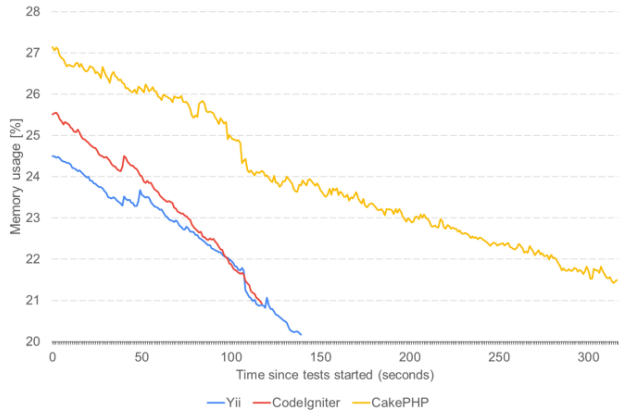

Fig. 31. Memory usage result for GET request

POST request

During POST requests, CPU usage was unstable across all three tested applications. As shown in Fig. 12, CPU utilization exhibited numerous spikes, reaching up to 14%. Under normal operation, CPU usage remained around 6.5–7.5%.

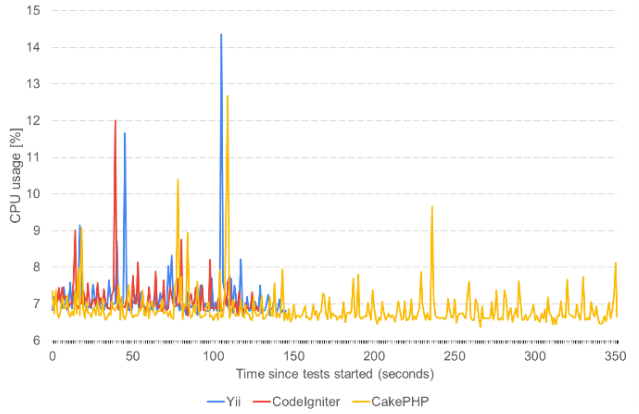

Fig. 42. CPU usage results for POST request

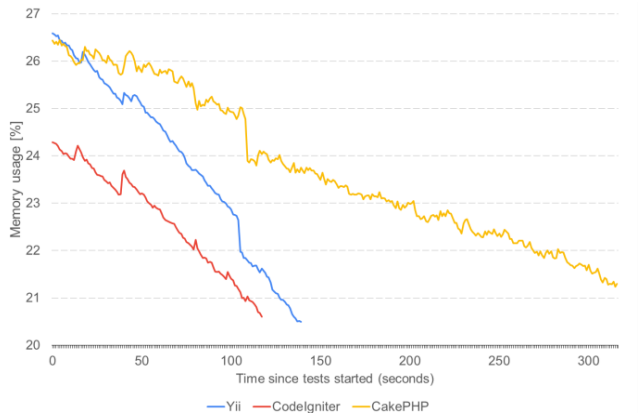

Fig. 53. Memory usage results for POST request

Memory usage was highest in the CakePHP-based application, starting at just over 26% and decreasing slowly throughout the test, consistently exceeding that of the other two applications (Fig. 13). Yii initially required the same amount of memory as CakePHP, but its memory usage declined much more rapidly. CodeIgniter maintained the lowest memory usage throughout the test.

PUT request

For PUT requests, CPU jitter was observed during approximately the first 100 seconds, characterized by numerous usage peaks across all tested applications, with CakePHP exhibiting the highest peaks (Fig. 14).

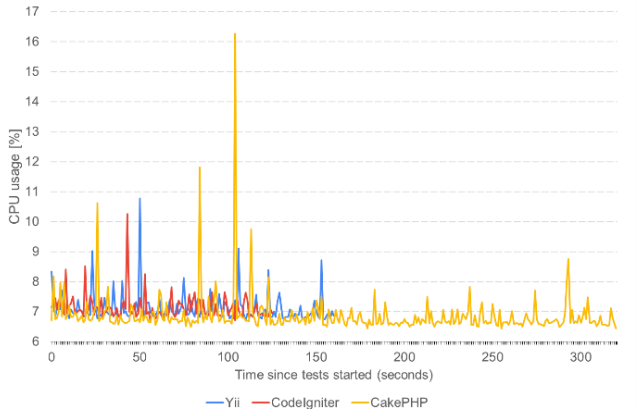

Fig. 64. CPU usage results for PUT request

Memory consumption for each framework for PUT request decreased over time (Fig. 15), again as in previous tests (Fig. 11) and (Fig. 13). CakePHP recorded the highest RAM consumption, being approximately 2–3% higher than the others. CodeIgniter generated higher memory consumption than Yii at the beginning of the test, but after about a minute, the results were very similar.

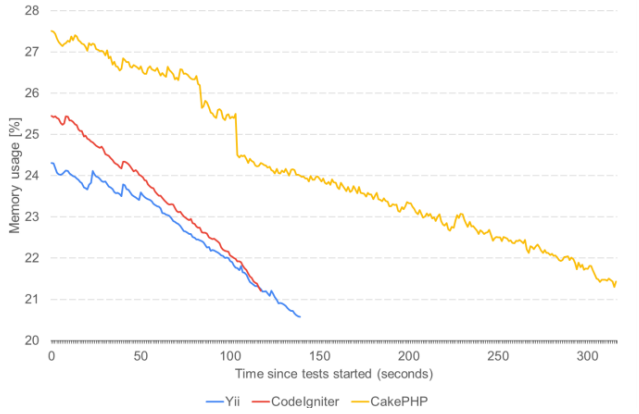

Fig. 75. Memory usage results for PUT request

DELETE request

For the DELETE query (Fig. 16), processor behavior was similar to that observed in the previous tests (Fig. 10, Fig. 12 and Fig. 14). Interestingly, the largest peak occurred at around 100 seconds in each test, as was also the case here.

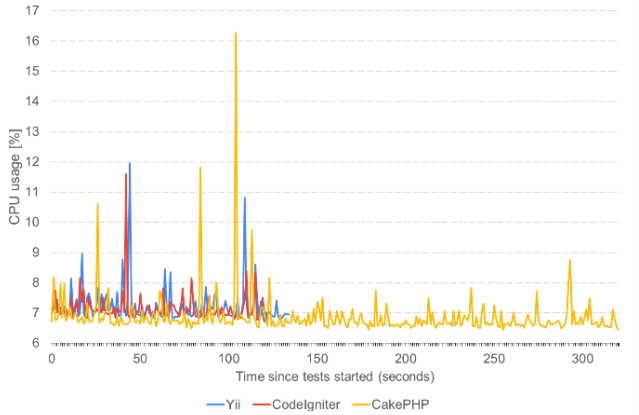

Fig. 86. CPU usage results for DELETE request

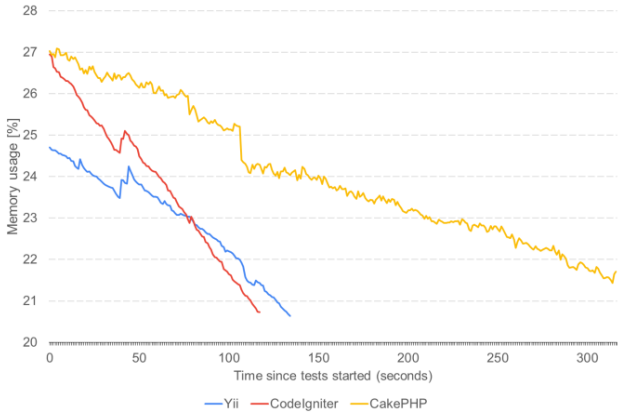

Fig. 97. Memory usage results for DELETE request

CakePHP again required the most memory (Fig. 17). CodeIgniter initially required the same amount as CakePHP, but its memory usage decreased rapidly over time, reaching the lowest point after about 80 seconds – lower even than Yii.

3.4. Popularity and community support

As shown in table 2, CakePHP is the most actively developed framework based on the number of open and closed pull requests on Github, while Yii is the most liked project on the platform. CodeIgniter, on the other hand, has the slowest development pace, with CakePHP falling in between. On Stack Overflow CodeIgniter was the most popular topic, with 44 questions – more than the other frameworks.

Table 2. Community and usage metrics (Q3 2025)

Metric	CodeIgniter	Yii	CakePHP
GitHub Stars	~5600	~14300	~8700
Number of PRs (2025)	211	84	317
Stack Overflow Questions (2025)	44	5	10
Latest version	4.6.1	2.0.53	5.2.5

Table 3 shows how frequently these technologies have been searched over the past four years. Google reports this data on a relative scale, allowing for easier comparison. The results indicate that both CodeIgniter and CakePHP have declined in popularity, while Yii has remained stable. This year, CodeIgniter became the least searched framework.

Table 3. Google trends popularity (relative scale 0–100)

Year	CodeIgniter	Yii	CakePHP
2022	88	91	85
2023	72	81	75
2024	81	90	85
2025	72	90	76

4. Conclusions

Studies have shown that two of the four hypotheses presented in this article were true.

The hypothesis H1 that proved true was that CodeIgniter is the fastest framework in terms of request handling and response generation. Unlike Yii and CakePHP, CodeIgniter does not use an ORM solution; instead, it employs QueryBuilder, a more procedural approach without the overhead of object-relational mapping, which involves converting database records into objects. This design choice had a positive impact on performance. Similar findings were reported in [15], which compared the lightweight PHP framework Slim to Laravel. However, CPU and memory usage in CodeIgniter was not always lower – for example, during PUT requests, memory usage was slightly higher compared to Yii, which uses ORM.

The second hypothesis, which states that CodeIgniter, compared to CakePHP and Yii, has the lowest memory and CPU usage, was not supported. Results for this framework were not consistent. For POST and DELETE request CodeIgniter did demonstrate better results but for PUT and GET memory usage were not best because Yii used less memory during tests. The lightweight nature of this framework did not help for all cases especially when we look at memory usage but it is worth noting that CPU usage was generally the lowest across all test so we can say that this hypothesis was partially confirmed.

Yii was neither the most reliable nor the most fault-tolerant framework, as CodeIgniter consistently generated fewer errors in every case. Yii's ORM-based approach, which introduces an additional layer of abstraction, proved less fault-tolerant, particularly under high-load scenarios (8,000 virtual users). While Yii began processing requests incorrectly, CodeIgniter continued to handle the heavy load without generating errors.

Based on the presented data, CakePHP demonstrated significantly higher resource utilization and less stability under heavy load, with the most erratic CPU usage spikes. Similarly, CakePHP's memory consumption was higher than that of the other frameworks.

In terms of popularity, the concept can be interpreted in different ways. For example, CodeIgniter is the most developed framework when considering the number of pull requests, and it also has the most questions on Stack Overflow. On the one hand, this may indicate that the framework is gaining more features; on the other, it could suggest that it has many issues to resolve and is, in fact, evolving the slowest. Popularity can also be assessed by the number of stars on GitHub, which reflects how well-liked a framework is among developers. In this regard, Yii was the most favored, with approximately 14,300 stars.

Finally, it is worth noting the limitations of this study. One limitation is the use of very simple test applications limited to CRUD operations. Frameworks with a richer set of features, such as CakePHP, might perform and scale more effectively in real large-scale production environments. Additionally, testing different versions of the selected frameworks would be valuable, as real-world applications often rely on varying versions, and such results could provide useful insights for migration decisions. Moreover, the present study focuses exclusively on performance aspects, whereas real-world applications require consideration of a broader set of factors. Although ORM mechanisms may have a negative impact on system performance, they offer several important advantages, such as improved development efficiency, database portability, and built-in security features. Furthermore, projects that employ frameworks such as CakePHP or Yii tend to be easier to maintain, as these frameworks automate many routine tasks, thereby reducing the responsibilities and potential error sources for developers.

References

[1] Abutaleb, H., Tamimi, A., & Alrawashdeh, T. (2021). Empirical Study of Most Popular PHP Framework. *2021 International Conference on Information Technology (ICIT)*, 608–611. https://doi.org/10.1109/ICIT52682.2021.9491679

[2] Ahmed, M. K., Bello, A. H., Jauro, S. S., & Dawaki, M. (2024). A comparative analysis of performance optimization techniques for benchmarking PHP frameworks: Laravel and Codeigniter. *Dutse Journal of Pure and Applied Sciences*, *10*(3c), 284–295. https://doi.org/10.4314/dujopas.v10i3c.27

[3] Anjum, N., & Alam, S. (2019). A Comparative Analysis on Widely used Web Frameworks to Choose the Requirement based Development Technology. *IARJSET*, *6*(9), 16–24. https://doi.org/10.17148/IARJSET.2019.6902

[4] Arif, Md. A., Hossain, M. S., Nahar, N., & Khatun, M. D. (2014). An empirical analysis of C#, PHP, JAVA, JSP, and ASP.Net regarding performance analysis based on CPU utilization. *Banglavision Research Journal*, *1*(14), 173–188.

[5] Benmoussa, K., Laaziri, M., Khoulji, S., Larbi, K. M., & Yamami, A. E. (2019). A new model for the selection of web development frameworks: Application to PHP frameworks. *International Journal of Electrical and Computer Engineering (IJECE)*, *9*(1), 695. https://doi.org/10.11591/ijece.v9i1.pp695-703

[6] Burokas, E., & Barisas, D. (2016). Comparison of popular PHP frameworks. *XXI international master and PhD students conference "Information Society and University Studies" (IVUS 2016)*, 23–27. https://ivus.vdu.lt/wp-content/uploads/2023/04/IVUS_2016_Proceedings.pdf#page=23

[7] Laaziri, M., Benmoussa, K., Khoulji, S., & Kerkeb, M. L. (2019). A Comparative study of PHP frameworks performance. *Procedia Manufacturing*, *32*, 864–871. https://doi.org/10.1016/j.promfg.2019.02.295
[8] Lei, K., Ma, Y., & Tan, Z. (2014). Performance Comparison and Evaluation of Web Development Technologies in PHP, Python, and Node.js. *2014 IEEE 17th International Conference on Computational Science and Engineering*, 661–668. https://doi.org/10.1109/CSE.2014.142
[9] Muqorobin, M., & Rozaq Rais, N. A. (2022). Comparison of PHP Programming Language with Codeigniter Framework in Project CRUD. *International Journal of Computer and Information System (IJCIS)*, *3*(3), 94–98. https://doi.org/10.29040/ijcis.v3i3.77
[10] Niarman, A., Iswandi, & Candri, A. K. (2023). Comparative Analysis of PHP Frameworks for Development of Academic Information System Using Load and Stress Testing. *International Journal Software Engineering and Computer Science (IJSECS)*, *3*(3), 424–436. https://doi.org/10.35870/ijsecs.v3i3.1850
[11] Odeh, A. H. (2019). Analytical and Comparison Study of Main Web Programming Languages – ASP and PHP. *TEM Journal*, 1517–1522. https://doi.org/10.18421/TEM84-58
[12] Patel, S. J., & Pancholi, P. D. (2018). Implementation and Comparison of MVC Model in ASP. net Framework and PHP Framework. *International Journal of Research and Analytical Reviews*, *4*(5), 827–835.
[13] Prokofyeva, N., & Boltunova, V. (2017). Analysis and Practical Application of PHP Frameworks in Development of Web Information Systems. *Procedia Computer Science*, *104*, 51–56. https://doi.org/10.1016/j.procs.2017.01.059
[14] Salas-Zárate, M. D. P., Alor-Hernández, G., Valencia-García, R., Rodríguez-Mazahua, L., Rodríguez-González, A., & López Cuadrado, J. L. (2015). Analyzing best practices on Web development frameworks: The lift approach. *Science of Computer Programming*, *102*, 1–19. https://doi.org/10.1016/j.scico.2014.12.004
[15] Sunardi, A. & Suharjito. (2019). MVC Architecture: A Comparative Study Between Laravel Framework and Slim Framework in Freelancer Project Monitoring System Web Based. *Procedia Computer Science*, *157*, 134–141. https://doi.org/10.1016/j.procs.2019.08.150
[16] Tiwari, V., Upadhyay, S., Goswami, J. K., & Agrawal, S. (2023). Analytical Evaluation of Web Performance Testing Tools: Apache JMeter and SoapUI. *2023 IEEE 12th International Conference on Communication Systems and Network Technologies (CSNT)*, 519–523. https://doi.org/10.1109/CSNT57126.2023.10134699
[17] Trent, S., Tatsubori, M., Suzumura, T., Tozawa, A., & Onodera, T. (2008). Performance Comparison of PHP and JSP as Server-Side Scripting Languages. In V. Issarny & R. Schantz (Eds), *Middleware 2008* (Vol. 5346, pp. 164–182). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-89856-6_9
[18] *Simple Movie DataBase Design*. (2025). [Data set]. Retrieved https://github.com/abdelwahab-ahmed-shandy/Database-Design-with-ERD-EERD-Relational-Schemas-SQL-Implementation/tree/main/01-Mini%20Project%20Data%20Base/Simple%20Movie%20DataBase%20Design

B.Sc. Karol Rak
e-mail: s95542@pollub.edu.pl

Graduate of the Lublin University of Technology in the field of Computer Science.
His research interests include web development and, in particular, PHP and Java based technologies.

https://orcid.org/0009-0002-9277-710X

Ph.D. Mariusz Dzieńkowski
e-mail: m.dzienkowski@pollub.pl

Assistant professor in the Computer Science Department at the Faculty of Electrical Engineering and Computer Science at Lublin University of Technology.
His scientific interests include human-computer interaction, eye tracking applications and web application development.

https://orcid.org/0000-0002-1932-297X