

OpenLambdaVerse: A Dataset and Analysis of Open-Source Serverless Applications

Ángel C. Chávez-Moreno

Escuela Superior Politécnica del Litoral, ESPOL
Guayaquil, Ecuador
acchavez@espol.edu.ec

Cristina L. Abad

Escuela Superior Politécnica del Litoral, ESPOL
Guayaquil, Ecuador
cabad@fiec.espol.edu.ec

Abstract—Function-as-a-Service (FaaS) is at the core of serverless computing, enabling developers to easily deploy applications without managing computing resources. With an Infrastructure-as-Code (IaC) approach, frameworks like the Serverless Framework use YAML configurations to define and deploy APIs, tasks, workflows, and event-driven applications on cloud providers, promoting zero-friction development. As with any rapidly evolving ecosystem, there is a need for updated insights into how these tools are used in real-world projects. Building on the methodology established by the Wonderless dataset for serverless computing (and applying multiple new filtering steps), OpenLambdaVerse addresses this gap by creating a dataset of current GitHub repositories that use the Serverless Framework in applications that contain one or more AWS Lambda functions. We then analyze and characterize this dataset to get an understanding of the state-of-the-art in serverless architectures based on this stack. Through this analysis we gain important insights on the size and complexity of current applications, which languages and runtimes they employ, how are the functions triggered, the maturity of the projects, and their security practices (or lack of). OpenLambdaVerse thus offers a valuable, up-to-date resource for both practitioners and researchers that seek to better understand evolving serverless workloads.

Index Terms—serverless, serverless computing, function-as-a-service, cloud computing, characterization, repository mining.

I. INTRODUCTION

Since the release of AWS Lambda in 2014, the Function-as-a-Service (FaaS) model has seen increasing adoption. FaaS (serverless) solutions provide the tools to abstract infrastructure management away from the developer, while offering a pricing model based on usage, dynamically provisioning resources based on demand [1]. Current offerings enable the implementation of complex event-driven architectures, with features such as state management, workflow orchestration, and integration with cloud services like databases, storage systems, messaging queues, AI/ML pipelines, IoT services, and mobile backends. These offerings include commercial providers like AWS and Azure, and open-source alternatives like OpenWhisk and Kubeless; also available are tools that streamline development and deployment, like the popular Serverless Framework [2]. Despite its positives, serverless computing continues to present challenges that drive research, like dealing mitigating cold start latency, optimizing performance and cost-effectiveness for diverse workloads, addressing security concerns, and enabling interoperability.

To help focus research and development, others have collected and characterized applications that employ serverless functions [3]–[6]. The Wonderless dataset [3], [7], [8] consists of 1,877 projects from GitHub that use the Serverless Framework; the paper includes a brief characterization of the dataset, looking into programming languages, runtimes, and repository contributions. The Open-Source Serverless Search (OS³) [5] extended the application scraping with an improved approach and collection/curation code that excludes unlicensed and duplicate entries, and collected a dataset [9] with data from 5,981 repositories; OS³ focuses on improved code scraping and supporting search, and does not include a characterization of the data. AWSomePy [6], [10] also extended the Wonderless methodology to obtain a dataset with 145 serverless applications implemented in Python using the Serverless Framework for AWS Lambda; the characterization included analysis of plugins, code complexity, and cloud services and API usage.

With each new characterization effort, the community acquires a better understanding of how *current* serverless applications are being built. Our work updates and extends the Wonderless dataset, including updating the code so that it works with the new GitHub REST API and conforms to limit policies for code search and metadata collection [11]. *OpenLambdaVerse* is the most up-to-date serverless application (code) dataset; the dataset is a collection of open-source serverless applications that use the Serverless Framework for AWS Lambda, written with a diverse set of programming language and runtimes. We build upon the Wonderless methodology, and include the improvements introduced in OS³ and AWSomePy, providing key insights into the architecture and security policies on these applications, which could serve as reference for developers and researchers in the field.

The contributions from this work are:

- 1) A new serverless repository extraction tool,¹ based on the Wonderless code plus improvements based on best practices of the repository mining community.
- 2) An up-to-date, publicly available dataset² of applications on GitHub that use the Serverless Framework for AWS.
- 3) An analysis of the dataset showing current trends and a comparison with prior results, and insights from these.

¹<https://github.com/disel-espol/openlambdaverse>

²<https://zenodo.org/records/16533581>

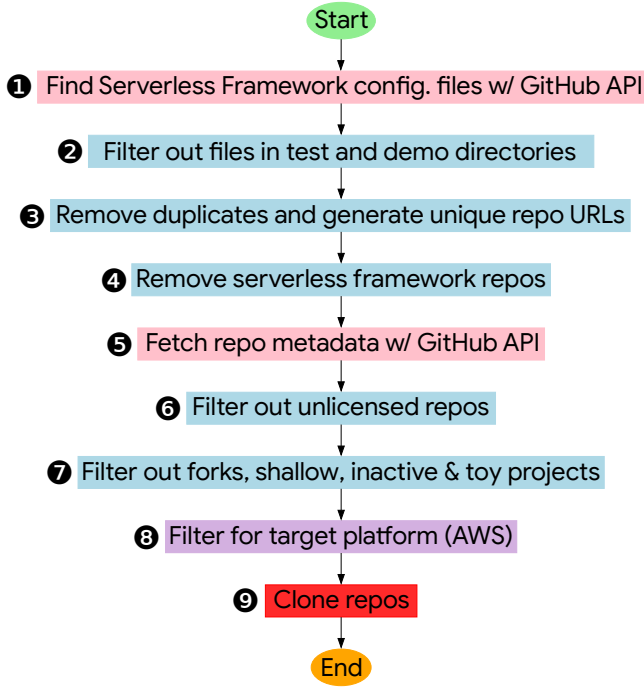


Fig. 1. Repository extraction methodology. Steps in pink are where GitHub REST API is consumed. Filtering steps are in blue, except for the selection of AWS as target platform, in purple. Cloning of master branches, in red.

II. DATASET COLLECTION AND CURATION

We ran our collection process on April 24, 2025, using the methodology outlined in Figure 1 and described below.

1 Configuration files: As in Wonderless and AWSomePy, OpenLambdaVerse collects serverless applications that implement the Serverless Framework using at least one *serverless.yml* file for configuration and deployment. In this first step, we obtained 34,320 configuration file URLs.

2 Filtering out test and demo directories: We filter out 8,925 URLs because they were part of example, test or demo directories. This is a step done on previous studies as well.

3 Removing duplicates and generating unique repository URLs: 20,144 URLs remained after removing duplicates stemming from more than one *serverless.yml* in the same repo.

4 Removing Serverless Framework contributions: We identified and filtered out 64 repositories belonging to two official Serverless Framework accounts (users: *serverless* and *serverless-components*), leaving us with 20,080 URLs.

5 Fetching repository metadata: We fetch metadata from each of the repositories by parsing the *serverless.yml* file and by using GitHub’s REST API to obtain: plugins, runtimes, events, provider, size, forks, stars, topics, primary programming language, flags: [archived, disabled, (is a) fork], bytes per programming language, contributors, private vulnerability reporting, tags, last commit date, stars, watchers, open issues, and license. We compile this in a JSONL, where each JSON object contains key-value pairs with a repository’s metadata.

6 Filtering unlicensed projects: As in OS³ [5], we exclude projects that aren’t licensed. An unlicensed repository can’t be reused without the owner’s permission. This makes our scraping follow a more direct path to applications that are purposely made open-source, leaving us with 4,080 repositories.

7 Filtering forks, shallow, inactive and toy projects: Before collecting (cloning) the repositories, we filter some out based on criteria collected by surveying papers in the Mining Software Repositories (MSR) research community:

- Forks. As forks are derived from an existing project they contain duplicated code and distort the results [12].
- Shallow projects. We define shallow projects as those < 100KB, a heuristic to filter trivial/toy projects [12].
- Inactive projects. We define inactive projects as those that have not been updated in the last 24 months; they are filtered out because they may not represent *recent* trends in application development [13]–[17].³
- Toy projects. We filter out projects that contain any of the following keywords in the name, description or topics: (‘example’, ‘tutorial’, ‘demo’, ‘sample’, ‘starter’, ‘playground’, ‘hello-world’, ‘test’, ‘template’, ‘learn’, ‘workshop’, ‘exercise’, ‘skeleton’, ‘boilerplate’, ‘mock’, ‘poc’, ‘guide’). This list was compiled from the code of Wonderless and AWSomePy, plus 5 keywords we added after an exploratory analysis (last 5 in the list).

These filters leave us with 816 repositories in our list.

8 Selecting AWS as the target platform: We keep only the projects where AWS is the target platform (in *serverless.yml*), as it is the only platform currently supported by the Serverless Framework.⁴ This brings the final count to 668 repositories.

9 Cloning the repositories: Finally, we clone the main branch of the repositories to preserve the snapshots associated with the dataset. The aggregate size of the repositories is 51GB and has been released compressed (6GB) on [Zenodo](#).

Notes on the consumption of the GitHub REST API

Our scraping code includes functionality to conform to the limits of the current GitHub REST API version 2022-11-28 for authenticated (non-enterprise) users [19], [20], and the rate limit changes to the search API introduced in 2023 [21]:

- Only files <384KB can be found via code search.⁵ This should not affect results as large YAML files are uncommon (e.g., we found only 3 in 86KB-168KB range).
- GitHub’s API rate limit is 5,000 requests per hour; we use a sleep timer to avoid exceeding this limit.
- The code search endpoint has a rate limit of 10 reqs/min; to conform, we use 7-sec sleep timers between requests.

³This criteria replaces the project maturity threshold of Wonderless (last – first commit $\geq 1y$), which could stem from the purpose of these projects; e.g., created as part of a competition. However, most projects created on those contexts are filtered out with our current criteria (e.g. size), and we decided not to exclude recent projects as they have the potential to provide additional insights in the rapidly evolving serverless ecosystem.

⁴For v4, the framework deprecated the support of non-AWS providers [18].

⁵Wonderless doesn’t include *serverless.yml* files <0.5KB as they tend to be empty. We do not exclude small files unless other steps exclude the project.

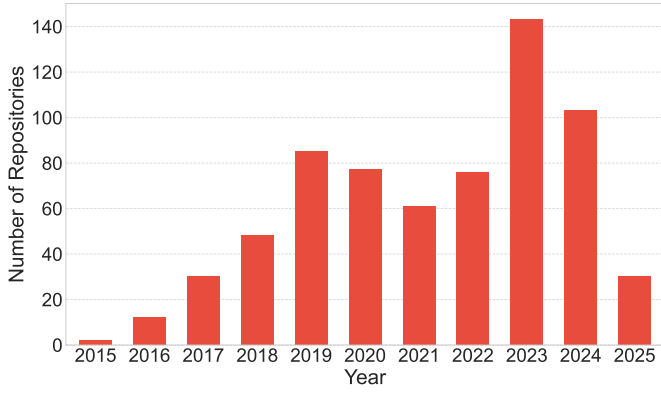


Fig. 2. Creation year of the repos in OpenLambdaVerse (cutoff: 24/04/2025).

TABLE I
TOP 11 SERVERLESS FRAMEWORK PLUGINS.

Plugin	Count
serverless-offline	270
serverless-webpack	91
serverless-dotenv-plugin	87
serverless-plugin-typescript	80
serverless-python-requirements	71
serverless-prune-plugin	49
serverless-domain-manager	42
serverless-esbuild	31
serverless-bundle	25
serverless-iam-roles-per-function	25
serverless-dynamodb-local	24

III. DATASET CHARACTERIZATION

We present an analysis of OpenLambdaVerse with respect to the year of creation, use of plugins, code complexity, run-times (and programming languages), topics, repository sizes, popularity, event triggers and security-related metadata.

A. Year of creation

As the Serverless Framework may be declining in popularity [22], we wanted to make sure that OpenLambdaVerse was not biased towards older repositories. Fig. 2 shows the number of repositories per year of creation. A declining popularity of the framework is not yet observable, as the dip in repositories observed in 2025 corresponds to it being a partial year.

B. Plugins

The Serverless Framework provides plugins to extend its core functionalities. Table I describes the top 11 plugins in the dataset and the number of repositories that use each. The leading plugin is *serverless-offline*, which emulates AWS Lambda and API Gateway on a local machine; this is useful for prototyping while avoiding cloud usage costs. Next in popularity is *serverless-webpack*, used to bundle Lambda functions using Webpack; this plugin simulates local API Gateway endpoints. Similarly for local testing is *serverless-dynamodb-local*. *serverless-dotenv-plugin* handles secrets in .env files, while *serverless-plugin-typescript* and *serverless-python-requirements* are needed when using Typescript and

TABLE II
LINES OF CODE PER PROGRAMMING LANGUAGE.

Language	Files	LOC	Percentage (%)
PHP	25,454	6,163,832	46.15%
JavaScript	16,522	5,081,974	38.05%
TypeScript	22,880	1,713,025	12.83%
Python	2,053	179,825	1.35%
Rust	49	74,902	0.56%
Go	698	57,893	0.43%
C#	269	19,805	0.15%
Cython	43	16,243	0.12%
Java	231	15,788	0.12%
Ruby	27	915	0.01%
PowerShell	2	160	<0.01%
Other	808	30,913	0.23%
Total	69,036	13,355,275	100.00%

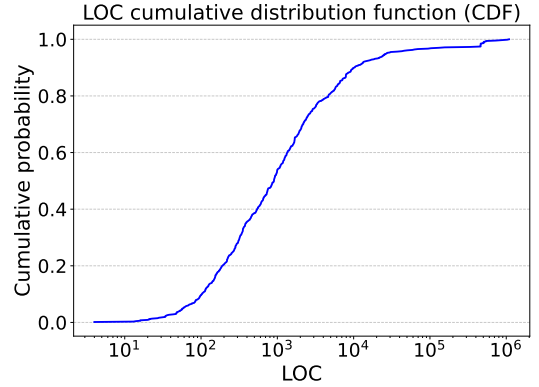


Fig. 3. LOC CDF for all code in the repositories (not only function handlers).

Python, respectively. *serverless-iam-roles-per-function* is for access management in AWS. However, seeing that it is present in just 25 out of 668 projects (3.59%), it seems that the best practices for applying governance are not used consistently (this was also observable in AWSomePy [6]).

C. Code complexity (measured through lines of code, LOC)

We use CLOC [23] to analyze the lines of code (LOC) of the repositories and then analyze the results to find which are the most used programming languages in the dataset (shown in Table II).⁶ Figure 3 shows the cumulative distribution function (CDF) of the lines of code in each repository, with a mean LOC of 20,022. Regarding the programming languages in the repositories (not just the function handlers), the leading one is PHP (46.1%), followed by JavaScript (38.1%), TypeScript (12.8%) and Python (1.4%), where the percentages are calculated as the LOC in that language to the total LOC in OpenLambdaVerse. Rust and Go are more popular than some traditionally popular languages like C#, Java and Ruby, highlighting their recent rise in popularity and adoption.

TABLE III
TOP PROGRAMMING LANGUAGES BY TOTAL CODE SIZE (BYTES).

Language	Total Bytes	Occurrence (%)
PHP	362,887,947	52.36%
JavaScript	247,084,664	35.65%
TypeScript	64,327,098	9.28%
Python	7,814,919	1.13%
Rust	2,773,185	0.40%
Go	1,962,996	0.28%
Cython	904,887	0.13%
ANTLR	857,375	0.12%
C#	830,584	0.12%
Java	772,276	0.11%
C	737,340	0.11%
Other	2,163,226	0.31%
Total	693,116,497	100.00%

TABLE IV
TOP RUNTIMES USED BY THE SERVERLESS FUNCTIONS.

Runtime	Count	Percentage
nodejs	426	73.58%
python	94	16.23%
provided (OS-only)	20	3.45%
java	16	2.76%
go	12	2.07%
ruby	7	1.21%
dotnet	4	0.69%

D. Bytes of code

The GitHub API has an endpoint to list the languages on a repository and the bytes of code written in that language [24]. The aggregates per programming language are in Table III. PHP is not natively supported by AWS Lambda so the high prevalence (52.4%) most likely comes from it being used to implement dynamic pages; however, some small percentage could come from handlers, as the OS-only (provided) runtime can be used to run unsupported languages in AWS Lambda. JavaScript (35.7%) and TypeScript (9.3%) are the most popular AWS Lambda-supported languages in the dataset.

The high-presence of non-Lambda languages (bytes and LOC) makes it evident that there are limitations with an automated analysis of the programming languages present in the repositories, so to get a better picture of the prevalence of languages in serverless functions, it is best to analyze the runtimes used (next Subsection).

E. Runtimes

Table IV lists the most popular function runtimes. Nodejs (JavaScript and TypeScript) is the leading runtime, used in 73.58% of the functions, followed by Python (16.23%). OS-only (provided) runtimes can be used for programming languages not directly supported by Lambda [25], like Go and Rust. The unsupported Go appears in the table (2.07%) because Lambda supported Go 1.x runtime up to Jan 2024 [26].

⁶To be consistent with the analysis of the bytes of code (III.D) done using GitHub’s API, for III.C and III.D, we use the list of all languages known to GitHub, available at: <https://github.com/github-linguist/linguist/blob/main/lib/linguist/languages.yml> and select those with “type: programming”.

TABLE V
TOP TOPICS IN THE GITHUB REPOSITORIES.

Topic	Count
serverless	92
aws-lambda	59
aws	42
serverless-framework	36
lambda	36
nodejs	32
typescript	30
cvs-project	17
hacktoberfest	16
dynamodb	15

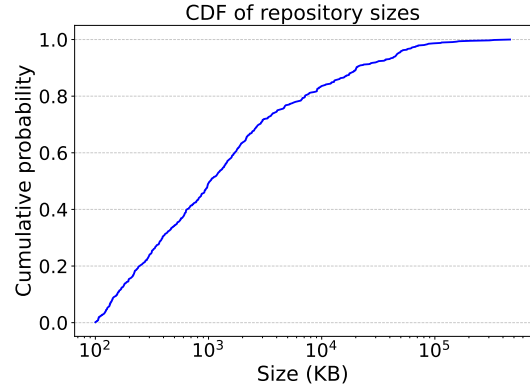


Fig. 4. CDF of repository sizes.

F. Topics

Topics are user-defined tags applied to GitHub repositories. Table V shows the top 11 topics, including unsurprising ones like *serverless*, *aws-lambda*, *aws*, *serverless-framework* and *lambda*. Popular topics also include runtimes or programming languages (*nodejs* and *typescript*) and a specific AWS service (*dynamodb*). *hacktoberfest* is a yearly competition hosted by GitHub; the projects left after filtering are those that are still active, highlighting the impact of this event. *cvs-project* is the Commercial Vehicle Service (CVS) project from the UK’s Driver & Vehicle Standards Agency.

G. Repository sizes

Figure 4 shows the CDF of repository sizes, which range from 100KB to 462MB, with a median of 1.03MB and a standard deviation of 31MB.

H. Repository popularity

The number of contributors, forks, and issues serve as indicators of a project’s popularity and adoption. OpenLambdaVerse repositories average 5 contributors per project, with 34.58% of these repositories driven by a single contributor. The dataset also reveals an average of 7 open issues per project, which translates to notable community support. Although 43.71% of the projects report zero open issues, the collected metadata alone does not suffice to conclude whether the projects initially had any open issues. Furthermore, a significant proportion (38.62%) of these repositories have been forked at least once, another indicator of overall adoption.

TABLE VI
EVENT TRIGGERS USED TO CALL THE SERVERLESS FUNCTIONS.

Event Trigger	Count	Triggers (%)	Repos (%)
http	1547	62.43%	51.05%
httpApi	529	21.35%	23.20%
schedule	199	8.03%	14.37%
websocket	48	1.94%	1.80%
sqs	30	1.21%	3.14%
sns	27	1.09%	2.40%
s3	24	0.97%	2.69%
eventBridge	17	0.69%	0.90%
stream	11	0.44%	1.65%
cloudwatchEvent	9	0.36%	1.05%
cognitoUserPool	6	0.24%	0.45%
cloudwatchLog	4	0.16%	0.45%
alexaSkill	1	0.04%	0.15%
alb	1	0.04%	0.15%

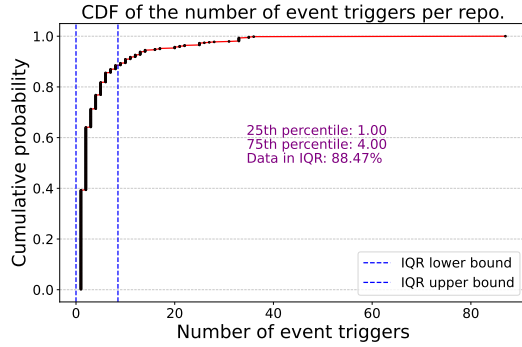


Fig. 5. CDF of the number of event triggers per repository.

I. Event triggers

Serverless code is launched in response to events like HTTP requests, scheduled tasks, and notifications. Table VI lists these triggers and their prevalence in the dataset; *http* and *httpApi* are the most popular, representing 83.78% of the triggers in the dataset, with 73.20% of the repositories using one or the other at least once.⁷ Scheduled events constitute 8.03% of triggers and appear in 14.37% of the repositories. Triggers appearing in 2-4% of the repositories are: *sqs*, *s3* and *sns*.

Figure 5 shows the CDF of the number of triggers per application; on the majority of the projects (88.47%), the number of event triggers is between 1 and 4. As the number of triggers increases beyond that mark, we ask ourselves if this represents increased code complexity on function implementations. Figure 6 maps the relationship between the number of event triggers and code complexity (measured in LOC). The visual results and the 0.05 correlation coefficient suggest a negligible correlation between the variables.

J. GitHub security-related metadata

Checking for private vulnerability reporting on a public repository is an additional key piece of metadata related to GitHub repository security [28]. This feature is crucial for constant vulnerability checks and updates. Seeing this

⁷Both *http* and *httpApi* triggers are HTTP/REST triggers coming from API Gateway; the name is due to a version change: v2 (*httpApi*) or v1 (*http*) [27].

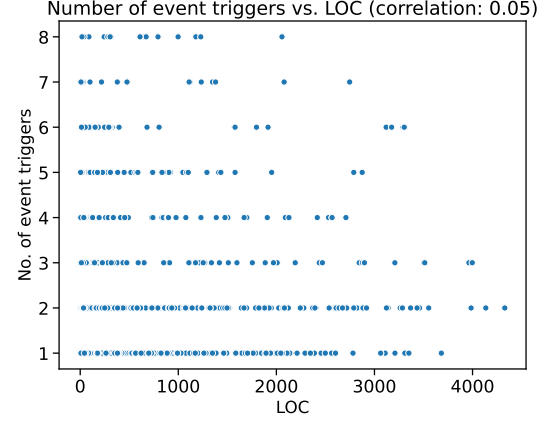


Fig. 6. Number of event triggers vs. LOC

implemented on only 98 repositories (14.67%) reinforces our concerns regarding the lack of proper application of security best practices in serverless applications in the wild.

IV. COMPARISON WITH PRIOR WORK

We discuss how our characterization compares to prior work in the serverless domain. Table VII lists the studies analyzed herein. In addition to those studies, we also refer to the “State of Serverless” report by Datadog [22], which sources data from 20,000+ of companies in Datadog’s customer base and was updated yearly from 2020 through 2023; it is not listed in the table because its source data is not publicly available. We do not include trace-based studies that study/compile serverless calls without looking at the applications themselves (e.g., the “Serverless in the Wild” study [29]).

With the exception of the work by Eismann et al. [4], the other studies in Table VII target the Serverless Framework due to its widespread adoption; this framework was an early product in this domain and it is widely used for deployment of serverless applications in AWS [22], [30]. Unlike the other studies in the table, Eismann et al. did a manual analysis of each of the 89 studied applications and this manual approach allowed them to consider any type of serverless application, at the cost of being the study that covered the smallest number of applications. Wonderless also looked beyond AWS; however, for OpenLambdaVerse we did not consider other platforms as the Serverless Framework deprecated the support of non-AWS providers with the release of version 4 in May 2024 [18].⁸

Given that these studies target different frameworks and cloud providers, we cannot draw sound conclusions from comparing the results of the different characterizations of serverless applications. Nevertheless, we do report how our results differ from prior studies, without making strong claims about what the differences in the results signify.

Runtimes: When compared to the stats in Wonderless, the popularity of the nodejs (73.6% vs 72.2%) and python (16.2 vs 19%) runtimes has remained relatively stable, with

⁸<https://github.com/serverless/serverless/releases?q=4.0&expanded=true>

TABLE VII
SERVERLESS APPLICATIONS DATASETS AND CHARACTERIZATION.

Study & Collection year	#applications	Dataset description	Dataset URL
Wonderless [3] 2020	1,877	Repo URLs and cloned repositories (GitHub, Serverless Framework: AWS, Azure, Google, OpenWhisk, Cloudflare, Kubeless).	https://zenodo.org/records/4451387
Eismann et al. [4] . . . 2021	89	Tabulated analysis of serverless applications sourced from GitHub, academic literature, and industrial literature; AWS, Azure, IBM, Google & private.	https://zenodo.org/records/5185055
AWSomePy [6] 2023	145	Dataset summary (metadata) and 145 cloned repositories (GitHub, Python, Serverless Framework: AWS).	https://zenodo.org/records/7838077
OpenLambdaVerse . . . 2025	668	Tabulated metadata and cloned repositories (GitHub, Serverless Framework: AWS).	https://zenodo.org/records/16533581

nodejs increasing slightly in popularity and python decreasing in popularity. These two runtimes are also reportedly the most common runtimes in the Eismann study (both at 42%) and the Datadog study ($\sim 43\%$ nodejs and $\sim 28\%$ python), though the prevalence percentages differ due varied application inclusion methodologies. In sum, all studies found that the most popular runtimes for serverless functions are nodejs and python, followed by java at a distant third place.

In OpenLambdaVerse and Wonderless, the other runtimes have popularities of less than 3.5%, and have remained stable in the 5 years that have passed. However, both Eismann and Datadog report Java runtimes having a higher prevalence (12% and 10%); this language also comes in third in popularity in OpenLambdaVerse and Wonderless, but at a lower popularity ($\sim 2.8\%$). The difference in the Datadog numbers are particularly important given that they also report them for AWS only, so different provider support cannot explain them. We believe this difference (higher popularity of Java in the Datadog study) may come from either inclusion of private repositories or the fact that they also included repositories with serverless applications managed by Terraform, which they report to be the preferred AWS Lambda deployment tool among larger organizations (which in turn may favor Java).

When comparing OpenLambdaVerse versus Wonderless we observe a $\sim 15\%$ decline in usage of Python and Go runtimes, the latter likely as a result of AWS deprecating this runtime in Jan 2024 [26]. The only runtime with a significant increase in adoption is “provided” ($\Delta 8\%$) which may have seen increased adoption to support Go as this language is now unsupported by Lambda. Datadog also found that custom runtimes are the fastest-growing type of functions and posit this may be due to an increased interest in serverless containers.

Plugins: AWSomePy is the only other study to include plug-in usage characterization. We find the top three plug-ins to be serverless-offline (40% prevalence), serverless-webpack (14% prevalence), and serverless-dotenv-plugin (13% prevalence). These results differ from the AWSomePy results: serverless-python-requirements (65%), serverless-pseudo-parameters (17%), and serverless-domain-manager (10%), presumably because AWSomePy only has Python applications. In OpenLambdaVerse, serverless-python-requirements appears in 5th place, with 10% prevalence.

Complexity (LOC): Wonderless and AWSomePy also report on the lines-of-code (LOC) per language and per repository, respectively. However, our Table II is not comparable to Table 1 in the Wonderless paper [3] because their analysis is

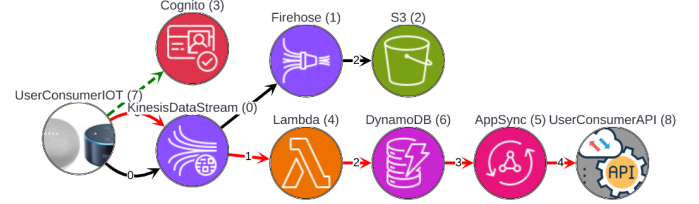


Fig. 7. Sample IoT architecture from the Cloudscape dataset [31]: TechConnect (tech-powered sports with IoT machine learning).

done only for the handlers; thus, we reported a comparison between their Table 1 and our Table IV under the Runtimes heading of this Section. With respect to the lines of code, OpenLambdaVerse has average LOC per repository of 20,022, with 53.82% of the repositories having 1K LOC or less, and 9.75% having 100 LOC or less. In AWSomePy the repositories are smaller on average (4,468 LOC), but have similar LOC for the 10 and 55 percentiles (the only percentiles specifically reported in [6]): 55% of the repositories have less than 1K LOC, and 10% are under 100 LOC.

Event triggers: Only the Eismann (E) study looks at what types of triggers are used to trigger serverless functions. In their study and in OpenLambdaVerse (O), the three most popular triggers are HTTP, Cloud events (sum of triggers coming from other service events like SQS, SNS and S3), and scheduled. However, the specific numbers differ: HTTP (O: 86%, E: 48%), scheduled (O: 16%, E: 13%), cloud events (O: 12%, E: 41%). These differences may not to represent changes in development practices but rather a result of different methodologies in selecting and analyzing the applications used by Eismann et al. (i.e., no automated scraping or analysis).

V. DISCUSSION

We found that serverless functions are frequently present in large projects that involve other, non-serverless components (e.g., PHP dynamic web pages), with functions written predominantly in node.js (JavaScript and Typescript) and Python, and triggered primarily by http and scheduled events. We interpret this as an indicator that serverless functions are frequently used as part of larger, non-fully serverless projects, due to their simplicity and ease of implementation and deployment.

Regarding the types of triggers used to call the Lambdas, we observe that IoT-specific triggers are missing, a surprising observation given that Lambda is used in a significant number of IoT/Edge applications: A recent study from our group

found that Lambda is present in 60% of Edge architectures applications in a recent AWS-based dataset [32]. However, the applications in that dataset may trigger Lambda through other mechanisms like http, stream (Kinesis), websockets, and indirectly through other cloud services. For an example of this, see Fig. 1 in [32] (reproduced here for clarity, in Fig. 7). In addition, it is possible that IoT applications on GitHub are, in a significant number of cases, closed-source or unlicensed and thus excluded by our filtering steps, or that these are preferring other IaC frameworks like Terraform [22] or even other providers like Cloudflare.

The low rates of implementation of GitHub’s private vulnerability reporting feature causes concern for the disclosure on critical vulnerabilities on these applications. The owners of the repositories should implement tools that allow them to address security concerns privately before these become available to the public. Further, not studied in this paper is the trustworthiness of plugins and libraries used in the projects, a growing concern in projects that depend on external packages [33]; these security plugins should always be thoroughly evaluated for potential vulnerabilities before adoption.

VI. THREATS TO VALIDITY

While we sought to be systematic in our data collection process and analysis, this study has limitations that could impact the generalizability and interpretation of our findings. We categorize these threats into internal and external validity.

A. External Validity

Scope and Representativeness of the Dataset: Our analysis is based on publicly available repositories. However, a substantial portion of software development, particularly for proprietary and enterprise-grade applications, occurs within private repositories. The inaccessibility of this data limits the depth and breadth of our analysis, potentially introducing a bias towards practices prevalent in open-source projects. Therefore, the trends and characteristics identified in this study may not fully represent the complete landscape of serverless function usage, especially in production environments within organizations that heavily utilize private repositories.

Technology Stack Specificity (Serverless Framework and AWS focus): A significant threat to the generalizability of our findings stems from our specific focus on repositories utilizing the Serverless Framework and targeting AWS Lambda. The serverless ecosystem is diverse, containing diverse frameworks (including open source options like OpenWhisk) and cloud providers (e.g., Azure Functions, Google Cloud Functions, Cloudflare Workers), though AWS remains the provider with the largest serverless adoption [22]. Our reliance on the Serverless Framework for identifying serverless projects and AWS as the primary cloud provider means that our results might not be directly transferable to projects built with other frameworks or deployed on different cloud platforms; for AWS, this framework remains the most popular infrastructure-as-code tool for managing AWS Lambda [22] functions, but Terraform is more popular for highly-complex architectures [22] so not

considering Terraform biases our results. Furthermore, new projects are increasingly preferring the AWS native options like CDK, SAM, and Chalice [22], and not including these in our study also limits the generalizability of our results. Different frameworks offer varying features, conventions, and levels of abstraction, which could influence how serverless applications are structured, secured, and deployed. Similarly, each cloud provider has unique services, best practices, and security models. While AWS Lambda is a prominent serverless platform, a broader comparison including other frameworks or open-source platforms would enhance the generalizability of our findings.

B. Internal Validity

Criteria and Limitations of Repository Filtering: The accuracy of our findings depends on filters used to identify repositories that employ serverless functions (and that are not toy projects, demos, etc.). To minimize this bias, we based our criteria in practices used by the mining software repositories community (and cited sources where appropriate), but this automated process might inadvertently exclude relevant repositories or include irrelevant ones.

Interpretation of Security Practices from Code Analysis: While our analysis highlighted concerns regarding the low usage of security plugins within the mined repositories, we acknowledge a problem with this approach: Security features in cloud applications can be managed directly through the cloud provider’s web interface. Therefore, an absence of explicit security configurations within the code repository does not necessarily imply a lack of security best practices in the deployed application (though we can argue that not committing those configurations to the repositories is prone to manual error and is a security issue on itself).

VII. CONCLUSIONS

We presented OpenLambdaVerse, a dataset with 668 repositories obtained by using a scraping methodology based on best practices from the mining software repositories community and the original methodology proposed for the Wonderless dataset. Our characterization of the dataset goes significantly further than that of Wonderless and presents an up-to-date view of serverless applications in the wild. We believe the dataset and characterization can be used by practitioners and academics to better understand how serverless workloads are evolving. Furthermore, as we are releasing our dataset and code, others can build upon our work and continue to provide a more in depth understanding of the evolving serverless landscape.

ACKNOWLEDGMENTS

We want to thank our anonymous reviewers for their helpful suggestions that lead to an improved final version of this paper.

REFERENCES

- [1] S. Kounev, N. Herbst, C. L. Abad, A. Iosup, I. Foster, P. Shenoy, O. Rana, and A. A. Chien, "Serverless computing: What it is, and what it is not?" *Commun. ACM*, vol. 66, no. 9, p. 80–92, Aug. 2023.
- [2] "Serverless Framework," <https://www.serverless.com/>, accessed on May 8, 2025.
- [3] N. Eskandani and G. Salvaneschi, "The Wonderless Dataset for Serverless Computing," in *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*, 2021, pp. 565–569.
- [4] S. Eismann, J. Scheuner, E. v. Eyk, M. Schwinger, J. Grohmann, N. Herbst, C. L. Abad, and A. Iosup, "The state of serverless applications: Collection, characterization, and community consensus," *IEEE Transactions on Software Engineering*, vol. 48, no. 10, 2022.
- [5] S. Bhatnagar, Z. Li, Z. Song, and E. Tilevich, "Os³: The art and the practice of searching for open-source serverless functions," in *IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, 2023.
- [6] G. Raffa, J. Alis, D. O'Keeffe, and S. Dash, "AWSomePy: A Dataset and Characterization of Serverless Applications," in *Workshop on Serverless Systems, Applications and Methodologies (SESAME)*, 2023.
- [7] "Wonderless," <https://zenodo.org/records/4451387>, accessed on May 5, 2025.
- [8] "Wonderless implementation," <https://github.com/prg-grp/wonderless/tree/master/impl>, accessed on May 5, 2025.
- [9] "serverless_dataset," https://github.com/edgeumd/serverless_dataset, accessed on May 5, 2025.
- [10] "AWSomePy dataset," <https://doi.org/10.5281/zenodo.7838076>, accessed on May 5, 2025.
- [11] GitHub. (2022) GitHub REST API Rate Limit Documentation. [Online]. Available: <https://docs.github.com/en/free-pro-team@latest/rest/rate-limit/rate-limit?apiVersion=2022-11-28>
- [12] Y. Sens, H. Knopp, S. Peldszus, and T. Berger, "A Large-Scale Study of Model Integration in ML-Enabled Software Systems," in *IEEE/ACM International Conference on Software Engineering (ICSE)*, May 2025.
- [13] H. He, B. Vasilescu, and C. Kästner, "Pinning is futile: You need more than local dependency versioning to defend against supply chain attacks," in *ACM Foundations of Software Engineering (FSE)*, 2025.
- [14] S. Baltes, J. Knack, D. Anastasiou, R. Tymann, and S. Diehl, "(no) influence of continuous integration on the commit activity in github projects," in *ACM SIGSOFT International Workshop on Software Analytics*, 2018.
- [15] J. Coelho, M. Valente, L. Silva, and E. Shihab, "Identifying unmaintained projects in github," in *ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, 2018.
- [16] W. Lu, E. Kasaadah, S. Karim, M. Germonprez, and S. Gogins, "Open source software lifecycle classification: Developing wrangling techniques for complex sociotechnical systems," *arXiv preprint arXiv:2504.16670*, 2025.
- [17] S. Flint, J. Chauhan, and R. Dyer, "Pitfalls and guidelines for using time-based git data," *Empirical Software Eng.*, vol. 27, no. 7, 2022.
- [18] I. Serverless, "Upgrading to serverless framework v4: Deprecation of non-aws providers," 2025, online, last accessed on Jul. 27, 2025. [Online]. Available: <https://www.serverless.com/framework/docs/guides/upgrading-v4#deprecation-of-non-aws-providers>
- [19] I. GitHub, "Rate limits for the REST API," 2025, online, last accessed on Jul. 30, 2025. [Online]. Available: <https://docs.github.com/en/rest/rate-limit/rate-limit?apiVersion=2022-11-28>
- [20] —, "REST API endpoints for search," 2025, online, last accessed on Jul. 30, 2025. [Online]. Available: <https://docs.github.com/en/rest/search/search?apiVersion=2022-11-28>
- [21] G. Blog, "Changes to the code search API," 2023, online, last accessed on Jul. 30, 2025. [Online]. Available: <https://github.blog/changelog/2023-03-10-changes-to-the-code-search-api/>
- [22] Datadog, "The state of serverless," 2023, online, last accessed on Jul. 27, 2025. [Online]. Available: <https://www.datadoghq.com/state-of-serverless/>
- [23] A. Danial, "cloc (GitHub Repository)," <https://github.com/AIDanial/cloc>, accessed on August 28, 2023.
- [24] "GitHub: List repository languages," <https://docs.github.com/en/rest/repos/repos?apiVersion=2022-11-28#list-repository-languages>, accessed on May 5, 2025.
- [25] AWS, "When to use Lambda's OS-only runtimes," 2025, online, last accessed on Jul. 28, 2025. [Online]. Available: <https://docs.aws.amazon.com/lambda/latest/dg/runtimes-provided.html>
- [26] —, "Lambda runtimes," 2025, online, last accessed on Jul. 30, 2025. [Online]. Available: <https://docs.aws.amazon.com/lambda/latest/dg/lambda-runtimes.html>
- [27] I. Serverless, "Http api (api gateway v2)," 2025, online, last accessed on Jul. 28, 2025. [Online]. Available: <https://www.serverless.com/framework/docs/providers/aws/events/http-api>
- [28] "GitHub: Check if private vulnerability reporting is enabled for a repository," <https://docs.github.com/en/rest/repos/repos?apiVersion=2022-11-28#check-if-private-vulnerability-reporting-is-enabled-for-a-repository>, accessed on May 5, 2025.
- [29] M. Shahrad, R. Fonseca, I. Goiri, G. Chaudhry, P. Batum, J. Cooke, E. Laureano, C. Tresness, M. Russinovich, and R. Bianchini, "Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider," in *USENIX Annual Technical Conference (USENIX ATC)*, 2020.
- [30] K. Kritikos and P. Skrzypek, "A review of serverless frameworks," in *IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)*, 2018.
- [31] S. Satija, C. Ye, R. Kosgi, A. Jain, R. Kankaria, Y. Chen, A. C. Arpacı-Dusseau, R. H. Arpacı-Dusseau, and K. Srinivasan, "Cloudscape: A study of storage services in modern cloud architectures," in *23rd USENIX Conference on File and Storage Technologies (FAST 25)*, 2025.
- [32] S. Santillan and C. Abad, "An analysis of hpc and edge architectures in the cloud," in *Proceedings of the IEEE International Conference on Cloud Engineering (IC2E, to appear)*, 2nd Workshop on Accelerated HPC in the Cloud-Edge Continuum, 2025.
- [33] M. Alfadel, D. E. Costa, E. Shihab, and B. Adams, "On the discoverability of npm vulnerabilities in node.js projects," *ACM Trans. Softw. Eng. Methodol.*, vol. 32, no. 4, May 2023.