The Journal of Systems & Software 204 (2023) 111793

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

Open tracing tools: Overview and critical comparison☆

Andrea Janes [a], Xiaozhou Li [b,c], Valentina Lenarduzzi [c,*]

[a] FHV Vorarlberg University of Applied Sciences, Austria
[b] Tampere University, Finland
[c] University of Oulu, Finland

ARTICLE INFO

Article history:
Received 21 July 2022
Received in revised form 16 May 2023
Accepted 22 June 2023
Available online 28 June 2023

Keywords:
Open tracing tool
Telemetry
Multivocal literature review
ChatGPT

ABSTRACT

Background: Coping with the rapid growing complexity in contemporary software architecture, tracing has become an increasingly critical practice and been adopted widely by software engineers. By adopting tracing tools, practitioners are able to monitor, debug, and optimize distributed software architectures easily. However, with excessive number of valid candidates, researchers and practitioners have a hard time finding and selecting the suitable tracing tools by systematically considering their features and advantages.
Objective: To such a purpose, this paper aims to provide an overview of popular Open tracing tools via comparison.
Methods: Herein, we first identified 30 tools in an objective, systematic, and reproducible manner adopting the Systematic Multivocal Literature Review protocol. Then, we characterized each tool looking at the 1) measured features, 2) popularity both in peer-reviewed literature and online media, and 3) benefits and issues. We used topic modeling and sentiment analysis to extract and summarize the benefits and issues. Specially, we adopted ChatGPT to support the topic interpretation.
Results: As a result, this paper presents a systematic comparison amongst the selected tracing tools in terms of their features, popularity, benefits and issues.
Conclusion: The result mainly shows that each tracing tool provides a unique combination of features with also different pros and cons. The contribution of this paper is to provide the practitioners better understanding of the tracing tools facilitating their adoption.

1. Introduction

In *software* engineering, the outcome of the engineering process is invisible (Brooks, 1987; Fenton and Pfleeger, 1998). As a consequence, it is difficult to understand progress and to reason about the produced output (British Computer Society and Royal Academy of Engineering (Great Britain), 2004). This is particularly complicated when developing systems that consist of many components. Today's trend of developing systems interacting with components deployed in the cloud or based on microservice architectures only exacerbates this problem.

One way to cope with invisibility is through measurement. Measurement enhances observability as it provides the data needed to understand the internal states of systems and its components (Kalman, 1960). Measurement is defined as the process of assigning numbers or symbols to the attributes of real-world entities to describe them using clearly defined rules (Finkelstein and Leaning, 1984; Fenton and Pfleeger, 1998).

Medicine distinguishes between the measuring instruments that need to *look inside* a patient and those that do not. For example, blood pressure can be measured directly via an arterial catheter (called *invasive*) or by placing a stethoscope on an artery, pumping up a cuff placed around the arm, and reading blood pressure on a special meter called a sphygmomanometer (such an approach is called *non-invasive*). Following the same terminology, measurement of software can be invasive or non-invasive: we can distinguish methods that require to modify the source code of the measured system (e.g., logging relevant events) or methods that consider the observed system a black box and measure how it interacts with the environment.

The two terms often used in software measurement are *tracing* and *telemetry*. Tracing, as the word *trace*, means "a mark or line left by something that has passed" (Merriam-Webster.com Dictionary, 2022b), is often used by developers to log what has happened and to understand if software is working as expected or not. While tracing is also measurement, the term emphasizes that relevant events are logged – together with the time of occurrence

☆ Editor: Dr. Alexander Chatzigeorgiou.
* Corresponding author.
E-mail addresses: andrea.janes@fhv.at (A. Janes), xiaozhou.li@oulu.fi (X. Li), valentina.lenarduzzi@oulu.fi (V. Lenarduzzi).

https://doi.org/10.1016/j.jss.2023.111793

– during the execution of software. In contrast, counting the lines of code of a class is not tracing, it is only measurement.

The second frequent term is telemetry: the word is composed by the Greek adjective *tele* (remote) and the word *metron* (measure) and means to measure something and transmitting the results to a distant station (Merriam-Webster.com Dictionary, 2022a). Such an approach is often needed when a large quantity of data is collected and it cannot be processed in situ; or, if data is collected from several sources (as in a distributed system) and it is necessary to collect the data in one place to obtain the complete picture of what is happening in the system. Another term often used in a distributed context is *distributed tracing*, where a *trace* represents the "whole journey of a request as it moves through all of the services of a distributed system" (Mägi, 2020) and the term *span* describes the part of the trace belonging to one service: a span represents a logical unit of work in completing a user request within *one* service; combining all spans describes one request across all services. Distributed tracing, implicitly, includes telemetry, since the data collected from various points needs to be transmitted for processing to another device.

Tracing is used in a variety of cases, e.g., to locate the cause why a system does not meet performance requirements or where failures occur. It is part of the toolkit used by software engineers to monitor, debug, and optimize distributed software architectures, such as microservices or serverless functions. Researchers and practitioners need to be aware of the tools currently in use and what features they possess. Tool vendors and (potential) tool producers need to understand how popular and adopted their tools are, as well as for through features they distinguish themselves from the competitors.

For this purpose, this paper aims to obtain an overview of tracing tools and to perform a critical comparison among them, focusing on those that are used among researchers and practitioners and are available on the market using an Open Source license. To achieve this objective, we first identified 30 tools in an objective, systematic, and reproducible manner adopting a Systematic Multivocal Literature Review approach (Garousi et al., 2019). Then, we characterized each tool looking at:

- What the tracing tool is able to measure (distinctive features);
- Popularity (both in peer-reviewed literature and online media);
- Advantages (benefits) disadvantages (issues) reported by researchers and practitioners in social media articles using
    1. topic modeling and sentiment analysis techniques for the benefits and issues extraction,
    2. ChatGPT [1] to support the topic interpretation.

The key results of the study shows that, among the considered tools, the features provided vary.

Therein, 10 of them are considered popular based on the volume of social media discussion. These are the ones we took into account for the benefits and issues analysis due to the lack of data from the others. There are six main criteria on which practitioners mainly have opinions. The results show that only very limited number of tools provide all the considered features but none of the tools are perceived positively or negatively in all the aspects.

To sum up, the main contribution of our work is represented by the large analysis up to date on the overview and comparison of the capabilities of Open Tracing Tools that implement Open Tracing API. Specifically, we advanced the current state of the art in four different manners:

1. By providing results from a objective, systematic, and reproducible approach and a detailed replication package3.4 with the data and scripts used to conduct our study and that can be used by the research community and practitioners to replicate and build upon it;
2. By providing a comparison of the distinctive features each tool possessed, which may be used by practitioners as a way to select the most suitable tool(s) based on the specific project needs;
3. By providing an overview about how much each tool is known for the researchers and practitioners considering how long have they been on the market;
4. By investigating the benefits and issues among the considered tools, which can inform tool vendors about the limitations of the current solutions available the market, other than making practitioners aware of how to benefit more from the combined capabilities of the considered tools;

The remainder of this paper is structured as follows. Section 2 presents the process we followed to identify the Open Tracing Tools studied in this paper, while Section 3 describes the empirical study we conducted. Section 4 describes the obtained results, while Section 5 discusses them. Section 6 highlights the threats to validity of this work, Section 7 the related work, and Section 8 draws conclusions and future works.

2. Systematic open tracing tools selection

To identify a list of Open Tracing Tools in an objective, systematic, and reproducible manner, we adopted the approach of a Systematic Multivocal Literature Review (MLR) (Garousi et al., 2019). The MLR process (Garousi et al., 2019) includes both peer reviewed as well as gray literature and the different perspectives between practitioners and academic researchers are taken account in the results. A MLR emphasizes the inclusion of gray literature in the data collection process for topics with a strong interest by practitioners. MLR classifies contributions as *academic literature* in case of peer-reviewed papers and as *gray literature* other types of content like blog posts, white-papers, pod casts, etc.

The process is divided into different phases with the main steps we followed depicted in Fig. 1. We started with the definition of the overall goal of the study and the formulation of research questions. The goal and the research questions determine the selection of the data sources and the search terms. The literature review is executed searching the literature and performing snowballing (Wohlin, 2014). After reviewing the initial set of documents, applying the inclusion/exclusion criteria, and evaluating the quality and credibility of sources, we obtained 41 documents. Based on a defined data extraction scheme, we extracted data useful to answer the defined research questions, and – through data synthesis and interpretation – we obtained the answers to the research questions.

Formulated as a GQM measurement goal (Basili et al., 2014), the objective of this paper can be described as follows: "Analyze the current literature about Open tracing tools for the purpose of characterization, with respect to its distinctive features, its popularity, and benefits and issues, from the point of view of a software developer in the context of a software development organization".

Following the guidelines to formulate questions along the GQM (goal, question, metric) paradigm (Basili et al., 1994), we devise the below research questions operationalizing "utility" into four dimensions: ability to measure, popularity, benefits, issues. Consequently, we formulated the following research questions:

[1] https://chat.openai.com/?model=gpt-4

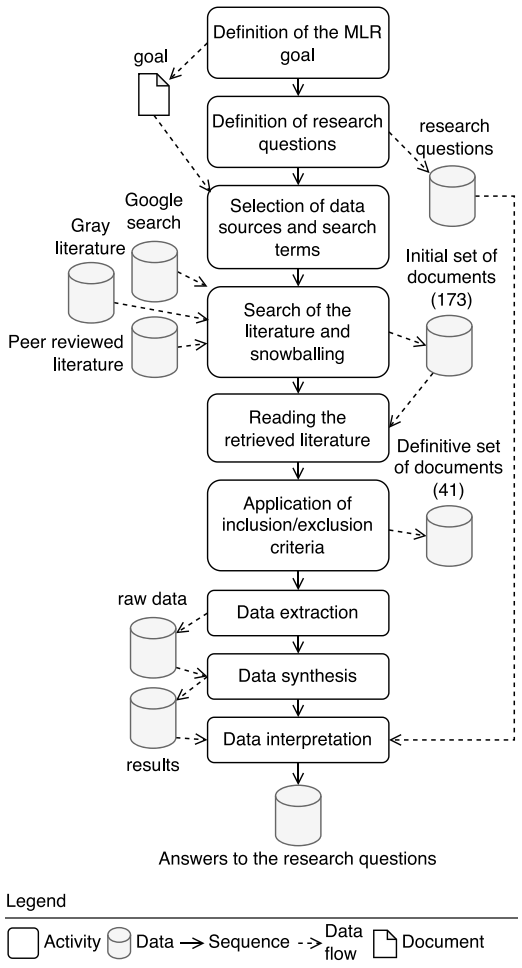

Fig. 1. Overview of the followed MLR process. *Source:* Adapted from Fig. 7 in Garousi et al. (2019).

RQ_1: Which distinctive features do the tools possess?
RQ_2: How popular are the identified tools?
RQ_3: Which benefits do the identified tools claim to achieve?
RQ_4: Which issues do the identified tools introduce?

With **RQ_1**, we aimed at characterizing each tool with respect to their distinctive features to provide a clear comparison of the differences between them. Though many tools provide similar functionalities, each one also provides unique features compared to the others. Then, in **RQ_2**, we investigated the popularity of each tool in terms of adoption both in academia and among the developers communities considering when the firs version was released on the market. Once characterized each tool, we proceeded with a finer-grained investigation considering benefits (**RQ_3**) and issues (**RQ_4**) to understand the unique advantages and drawbacks of each tool so that they can adopt the ones best suiting their needs.

To obtain a high recall and include as many papers as possible, we used the following broad search string to retrieve literature about open tracing tools:

(opentracing OR "open tracing") AND tool*

We used the asterisk character (*) to capture possible term variations such as plurals and verb conjugations. The search terms were applied to all the fields (i.e. title, abstract, and keywords), so as to include as many works as possible.

Peer-reviewed literature search. We considered the papers indexed by several bibliographic sources, namely: ACM digital Library,[2] IEEEXplore Digital Library,[3] Science Direct,[4] Scopus,[5] Google Scholar,[6] CiteseerX,[7] Inspec,[8] and Springer link.[9] The search was conducted in November 2022 and all raw data are presented in the replication package (Section 3.4).

Gray literature search. We adopted the same search terms for retrieving gray literature from online sources as we did for peer-reviewed ones. We performed the search using four search engines: Google Search,[10] Twitter,[11] Reddit.[12] and Medium[13] The search results consisted in books, blog posts, forums, websites, videos, white-paper, frameworks, and podcasts. This search was performed in November 2022.

Snowballing. Snowballing refers to using the reference list of a paper or the citations to the paper to identify additional papers (Wohlin, 2014). We applied backward-snowballing to the academic literature to identify relevant papers from the references of the selected sources. Moreover, we applied backward-snowballing for the gray literature following outgoing links of each selected source.

Application of inclusion and exclusion criteria. Based on guidelines for Systematic Literature Reviews (Kitchenham and Charters, 2007), we defined *inclusion and exclusion* criteria. We included tools that implement Open Tracing APIs because of their importance for tool providers and in particular, for tools aiming at architectural reconstruction and observavility. We excluded tools that could not be downloaded or installed, tools with no documentation on how to install or deploy them, as well as tools without a web site.

Evaluation of the quality and credibility of sources. Differently than peer-reviewed literature, gray literature does not go through a formal review process, and therefore its quality is less controlled. To evaluate the credibility and quality of the selected gray literature sources and to decide whether to include a gray literature source or not, we extended and applied the quality criteria proposed by Garousi et al. (2019) considering the authority of the producer, the applied methodology, objectivity, date, novelty, and impact.

Two authors assessed each source using the aforementioned criteria, with a binary or three-point Likert scale, depending in the criteria itself. In case of disagreement, we discussed the evaluation with the third author that helped to provide the final assessment.

Table 1 lists the outcome of the systematic tool selection, i.e., the tools that we identified through the above described process.

3. Tool analysis

The following steps – based on the research goal and derived questions described in Section 1 – describe the study context, the data extraction and analysis, and describe its verifiability and replicability following the approach suggested by Wohlin et al. (2012).

2 https://dl.acm.org
3 https://ieeexplore.ieee.org
4 https://www.sciencedirect.com
5 https://www.scopus.com
6 https://scholar.google.com
7 https://citeseer.ist.psu.edu
8 https://iee.org/Publish/INSPEC/
9 https://link.springer.com/
10 https://www.google.com/
11 https://twitter.com/
12 https://www.reddit.com/
13 https://medium.com

Table 1
The 30 retrieved tools.

Tool name	Web site
Appdash	https://github.com/sourcegraph/appdash
Appdynamics	https://www.appdynamics.com/
Containiq	https://www.containiq.com/
Datadog	https://www.datadoghq.com/
Dynatrace	https://www.dynatrace.com/
Elasticapm	https://www.elastic.co/
Grafana tempo	https://grafana.com/oss/tempo/
Haystack	https://expediadotcom.github.io/haystack/
Honeycomb.io	https://www.honeycomb.io/
Hypertrace	https://www.hypertrace.org/
Instana	https://www.instana.com/
Jaeger	https://www.jaegertracing.io/
Kamon	https://kamon.io/
Lightstep	https://lightstep.com/
Logit.io	https://logit.io/
Lumigo	https://lumigo.io/
New relic	https://newrelic.com/
Ocelot	https://www.inspectit.rocks/
Opencensus	https://opencensus.io/
Opentelemetry	https://opentelemetry.io/
Sentry	https://sentry.io/welcome/
Skywalking	https://skywalking.apache.org/
Site24 × 7	https://www.site24x7.com/
Signoz	https://signoz.io/
Splunk	https://www.splunk.com/
Stagemonitor	https://www.stagemonitor.org/
Tanzu	https://tanzu.vmware.com/tanzu
Uptrace	https://uptrace.dev/
Victoriametrics	https://victoriametrics.com/
Zipkin	https://zipkin.io/

3.1. Study context

We considered the eleven open tracing tools retrieved according the process described in Section 2. Table 1 shows the selected tools with their respective web site.

3.2. Data extraction

In this section, for each research question, we describe the collected data, i.e., the data we extracted from the retrieved search results.

Distinctive features of each tool (RQ$_1$). We extracted characteristics about the identified tools and grouped them into the following categories:

1. *General information*: the links to the source code repository, the chosen licenses, the adopted programming languages, and the different price types;
2. *Deployment*: the components contained in each tool, also in comparison to the suggested architecture defined by the Open Application Performance Management (OpenAPM) initiative (Novatec Consulting, 2022f) and their supposed deployment;
3. *Usage*: the suggested steps to use the tools, i.e., to setup data collection and to use the collected data;
4. *Data*: the actual data that can be collected with each tool, also compared to what the OpenTelemetry (The OpenTelemetry Authors, 2022) standard suggests: traces, metrics, and logs;
5. *Interoperability*: aspects that are important to guarantee a high degree of operability: the availability of an API, support for OpenTelemetry (The OpenTelemetry Authors, 2022), and if self-hosting is possible.

Tool popularity (RQ$_2$). We evaluated the popularity in terms of how much the tools are mentioned in public online sources. The following sources were investigated:

- *Peer-reviewed literature*: Using the same sources as in Section 2, we investigated the popularity of each tool by applying the following search string on all fields including title, abstract, body, and references: **("tool Name" OR "tool url") AND (*opentracing* OR "open tracing")**. In the case of tools with different names, we considered all variants in the "OR" term. Two authors independently evaluated the relevance of each publication reported by Google Scholar and Scopus, so as to exclude papers not written in English, false positives, or from different domains. In case of disagreement, a third author provided his/her opinion
- *Online media*: Using the same sources as in Section 2, we collected eventual posts, tags, users, groups or websites pertaining to the tools. In particular, we searched the tools' own communities, eventual groups present on LinkedIn and Google groups, as well as the number of appearances in commonly used communities and discussion forums such as: StackOverflow, Reddit, DZone, and Medium.

Benefits and issues (RQ$_3$ and RQ$_4$). To get the different opinions, especially on the advantages and problems of each tool, we extracted the corresponding content of the discussion threads from popular technology forums, including *StackOverflow*, *Medium* and *DZone*. StackOverflow is the largest forum of technology-related questions and answers (Q&As) for developers and tech-enthusiasts. Compared to StackOverflow, DZone is also one of the world's largest online communities for developers but focuses more on new tech-trends, e.g., DevOps, AI and big data, Microservices, etc. Furthermore, similar to DZone, Medium is also a well-known technology forum that provides tutorials and reviewing articles. Comparatively, articles on Medium are more common-reader-friendly and written in a non-technical style. These three platforms are the largest tech communities that can be considered to cover a representative school of opinions described in different styles.

Due to a different availability of APIs and different crawling policies of these three platforms, we applied different data crawling strategies for each platform accordingly:

- *StackOverflow*. We applied API-based content crawling using the StackExchange API to retrieve the questions and answers regarding each selected tracing tool. Specifically, we used the advanced search API[14] to extract all the questions that contain the name of the tool in either the title or the body of the question, together with the according answers. Please note, due to the daily query limitation of the API, the *pagesize* parameter was set to the maximum (i.e., 100 results shown per query) to minimize the crawling time.
- *Medium*: We adopted a manual crawling approach to respect Medium's policy of not allowing web crawling with tools like BeautifulSoup.[15] First, we searched the name of each tool and obtained all the articles about the tool. For each article, we manually copy/pasted the content into an individual text file named with the tool name and a sequence number.
- *DZone*: We adopted a hybrid crawling approach combining manual search and the use of BeautifulSoup. Different from Medium, Dzone allows crawling with BeautifulSoup within each individual article but not within the list of search results. Hence, we conducted a hybrid crawling strategy by manually collecting all article URLs for each tool and then automatically crawled the article content for each URL using BeautifulSoup.

[14] https://api.stackexchange.com/docs/advanced-search
[15] https://www.crummy.com/software/BeautifulSoup/

3.3. Data analysis

In this Section, we report the data analysis protocol adopted to answer the research questions.

Distinctive features (RQ_1). We inspected the documentation and the website of each tool and mapped the different features. Then, we created a Table that reports which features is available for each tool.

Tool popularity (RQ_2). We use the statistics from both scientific sources and media sources to interpret and compare the popularity of the selected tools.

Benefits and issues (RQ_3 and RQ_4). Fig. 2 depicts the approach overview of how to elicit the benefits and the issues for the users regarding open tracing tools adoption based on the analysis of social media articles. For all the retrieved tools we crawled the gray literature data from the previously identified sources, and compare the total number of articles of all the tools. Due to the difference in their popularity (the answer to RQ_2), the retrieved textual data volume for each tool varies greatly. Sufficient data volume is highly necessary in order to obtain meaningful interpretation of practitioners' collective opinions. According to practitioners' experience, a potential minimum number of articles of 600 is required for Latent Dirichlet Allocation (LDA) modeling on news articles while 5k to 10k for tweets (Naushan, 2020). In terms of the characteristics of the data for this study, we adopt the same threshold to verify the sufficiency and representativeness of the data.

The rest of the process is composed of five main steps:

- *Step 1: Preprocessing*: This step pre-processes the raw text data and prepares them for further analysis. First, we divide texts from the dataset into sentence-level instances since each text can contain multiple topics and various sentiments. Second, we build the bigram and trigram models, which means we identify the common phrases (e.g., *New York* instead of *new* and *york*). Subsequently, for each sentence, a series of text processing activities are required, including transforming text into lower cases, removing non-alpha-numeric symbols, screening stop-words, eliminating extra white spaces, and lemmatization.
- *Step 2: Filtering*: This step is to filter out the noninformative sentences with a trained text classifier. By doing so, we shall identify the sentences that contain useful information and screen out those not relevant. Aiming to answer **RQ_3** and **RQ_4**, the informative sentences shall contain an explicit description on the benefits or issues regarding the tracing tools. For example, "*In fact, with automated instrumentation as part of AppDynamics, metric data is produced consistently and comprehensively across all teams.*" is informative by describing the benefit of using; "*I checked the source code.*" is non-informative and should be filtered out.
- *Step 3: Topic Modeling*: Herein, we detect the main topics of the informative sentences identified from the previous step. Using topic modeling techniques, we shall identify the aspects which the articles are discussing. Especially, we use ChatGPT to support the effective summarization of each topic based on the according keywords.
- *Step 4: Topic Mapping*: In this step, using the topic model built with the informative texts, we can map each piece of text, which is about one particular tool, to one or multiple topics. By doing so, we shall know regarding each tool which topics are discussed in social media and how frequently each topic.
- *Step 5: Opinion Mining*: Finally, using opinion mining techniques on the texts, we can also know the collective sentiment from the social media concerning each topic for each tool. Therefore, the outcome shall provide a detailed reflection on the percentage of positiveness and negativeness for each topic for each tool.

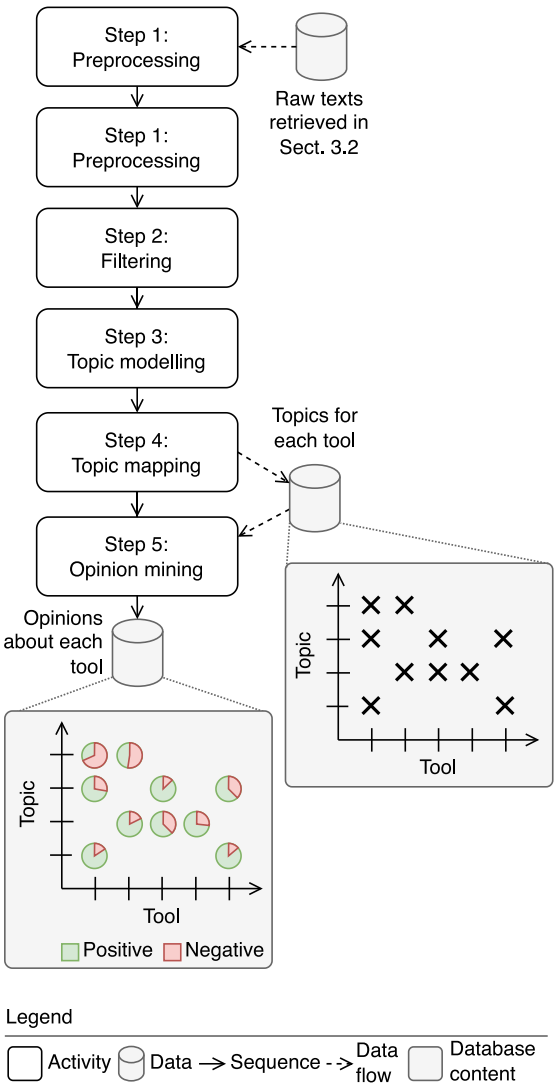

Fig. 2. Approach to detect benefits and issues from analyzing social media texts (RQ_3 and RQ_4).

Based on the approach described above, the benefits of each tool can be obtained analyzing the detected topics in which the tool is discussed positively, based on the collected opinions, which shall answer **RQ_3**. Similarly, the issues of each tool can be summarized by the according topics in which the tool is mentioned negatively, which answers **RQ_4**.

3.4. Verifiability and replicability

To allow our study to be replicated, we have published the complete raw data in the replication package.[16]

4. Results

In this Section, we report the obtained results by applying the steps described in the previous sections.

[16] https://figshare.com/s/8aa40eea5d50ed27d347

4.1. Distinctive features (RQ_1)

To answer **RQ_1**, we studied each tool to extract its distinctive features. The results are as follows.

General information. Table 2, for each tool summarizes the licenses we found in the repositories of the application, the reported programming languages, and the reference to the repository. We use the license identifiers defined by the SPDX workgroup (SPDX Workgroup, 2022). Please note that not all tools are entirely open: AppDynamics, Datadog, Instana, LightStep, and Wavefront have proprietary parts (e.g., the backend) but release agents or libraries (see below) using an Open Source license. The reported programming languages for each tool are only taken from the repositories with an Open Source license. We also investigated the different pricing strategies of all the selected tools (shown in Table 2). Among the 30 retrieved tools, 12 are open-sourced fully and 6 of these 12 provide free version with limited features. Moreover, we reported the age of each tool by means of the year when the firs version was released on the market.

To understand the level of support each tool can provide to a development team, Table 2 lists the supported programming languages, but this is only one aspect. The results described below complete the picture: the identified architectural components listed in Table 3 help developers to understand how a tool needs to be integrated in their context; the type of collected metrics listed in Table 4 illustrate which type of metrics can be collected with the various tools, and Table 5 compares the degree of interoperability each tool offers.

Table 2 lists those programming languages for which we found support in the relative repositories on GitHub and within the documentation. Those programming languages that are starred (*), support *non-invasive* instrumentation, i.e., the automated modification of the code so that tracing information is sent to the Agent. Such non-invasive instrumentation is called in different ways by the producers, e.g., Datadog calls it "Auto Instrumentation".

Deployment. To better understand how each tool is supposed to be used in a tracing scenario, we studied the documentation of each tool to extract the suggested deployment configuration. However, before looking at how the various tools are deployed, it is useful to define the typical components of a tracing tool. We use the terminology defined by the Open Application Performance Management (OpenAPM) initiative (Novatec Consulting, 2022f) (see Fig. 3):

- *Libraries* are used in source code to send data to an agent or directly to the collection component. In some scenarios, agents are able to modify the application automatically so that it sends data to an agent without necessary source code changes (e.g., instrumenting Java byte code).
- *Agents* are responsible for collecting data from a particular context, e.g., an application, the operating system, a mobile app, a database, or a web site. They usually run as part of applications or as an independent process and forward the data to collection components.
- *Storage*: After the data is collected, it needs to be stored. To improve performance, this can occur through a *transport* component that can fulfill routing or caching tasks. *Collectors* receive data from agents or other data sources and persist it to *Storage* components, e.g., a time-series database.
- *Data processing* components elaborate incoming data according to the analysis goals and prepare it for being used; the OpenAPM initiative distinguishes *visualizations* (e.g., in form of charts), *dashboarding*, and *alerting*.

Fig. 3 depicts a generalized deployment scenario of the various components using the terminology of the OpenAPM initiative. The architecture depicted in Fig. 3 also corresponds to the suggested architecture by the OpenTelemetry project (The OpenTelemetry Authors, 2022) (see below).

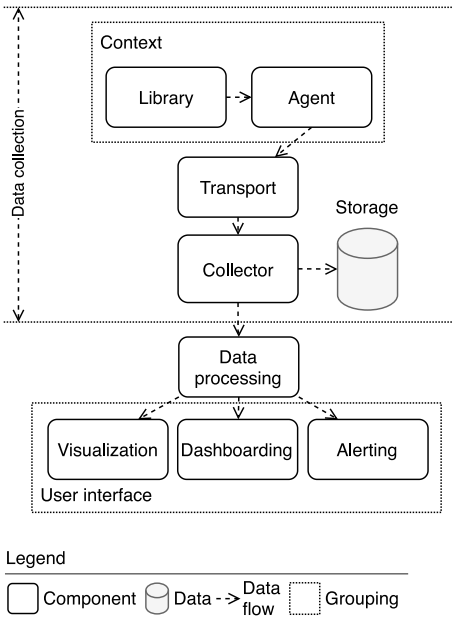

Fig. 3. APM components according to the OpenAPM initiative (Novatec Consulting, 2022f) and their typical communication data flow.

By studying the documentation of all the selected tools, we noted that only 7 out of the 30 tools (i.e., Dynatrace, Jaeger, New Relic, Sentry, Signoz, SkyWalking, and Tanzu) explicitly comply with the architecture described in Fig. 3. For majority of the tools, clear description of the "Transport" features is missing. Meanwhile, a large majority of the tools contain their own UI while some tools, e.g., Ocelot, require visualization tools, e.g., Grafana to display the tracing outcomes. Furthermore, a majority of the tools also provide libraries, agents, collectors, storage and data processing capabilities.

Table 3 lists each retrieved tool, its type, the identified components using the terminology suggested by the OpenAPM initiative, and the source (next to the name of the tool), where we obtained this information. When a tool uses a different term for a component, we mention this below the table. Please note that this table contains the components that are explicitly mentioned in the documentation. The absence of a component, e.g. an explicit transport component, does not mean that such a component does not exist in the platform but rather that this component might be contained in another component, e.g. the agent. Table 3 also shows that the used terminology is not standardized and that different producers and teams call components in different ways.

Usage. Regarding the usage of the tracing tools, we studied their installation and setup requirements, as described in the documentation.

All tools (except Ocelot and StageMonitor, which are Agents) are based on a similar setup, based on the suggested measurement architecture that tracing tools are based on (see Fig. 3). Therefore, using tracing tools always involves the following steps:

1. *Backend installation*: If the tool is installed on premise (as e.g., ElasticAPM (Elastic, 2022b), SkyWalking (Apache Skywalking contributors, 2022g), or Zipkin (Zipkin contributors, 2022d)): installing the tool following the documentation;
2. *Backend setup*: Preparing the backend to receive data: this step might involve creating an account for the organization (also called *tenant* in AppDynamics (AppDynamics,

Table 2
Features about each identified tool (RQ_1).

Tool	License	Programming language	Repo.	Pricing	Created
Appdash	MIT (AppDash, 2023a)	Go*, Python*, Ruby* (AppDash, 2023a,b)	AppDash (2023a)	Free	2014
AppDynamics	Proprietary, Apache-2.0, GPL-3.0, MIT (Appdynamics, 2022a)	Java*, Shell, .NET*, Python*, JavaScript, Go*, C/C++*, PHP*, NodeJS* (Appdynamics, 2022a,b)	Appdynamics (2022a)	$6/60/90/167 per month, per CPU Core; $.06 per month, per 1000 tokens; Quote	2008
Containiq	Proprietary (ContainIQ, 2023c)	C/C++*, Go*, Rust*, Python*, Ruby*, NodeJS* (ContainIQ, 2023e)	ContainIQ (2023c)	$20 per node, per month OR $.50 per GB of log data ingested; Quote	2021
Datadog	Proprietary, Apache-2.0, BSD-3-Clause, GPL-2.0, MIT, MPL-2.0 (DataDog, 2022a)	Go*, Python*, Ruby*, JavaScript, Node.js*, Java*, .NET* (DataDog, 2022a, 2023)	DataDog (2022a)	Free; $15/23 Per host, per month	2010
Dynatrace	Apache-2.0 (Dynatrace, 2023c)	C++*, .NET*, Erlang*, Go*, Java*, NodeJS*, Python*, Ruby*, Rust* (Dynatrace, 2023c,e)	Dynatrace (2023c)	$22/74/+15 per month for 8GB per Host; $11 per month for 10K annual DEM Units; $25 per month for 100k annual DDU; $0.10 per CAU	2005
ElasticAPM	Apache-2.0, BSD-2-Clause, BSD-3-Clause, Elastic-2.0, MIT (Elastic APM-Server contributors, 2022)	Go*, Python*, iOS*, Java*, NodeJS*, PHP*, Ruby*, Gherkin (Elastic APM-Server contributors, 2022; ElasticAPM, 2022)	Elastic APM-Server contributors (2022)	$95/109/125/175 per month	2012
Grafana tempo	AGPL-3.0-only (Tempo, 2023a)	Java*, Go*, .NET*, Python*, NodeJS* (Tempo, 2023a,b)	Tempo (2023a)	Free	2020
Haystack	Apache-2.0 (Haystack, 2023a)	Java*, NodeJS*, Python*, Go*, HCL, Shell, Smarty (Haystack, 2023a,b)	Haystack (2023a)	Free	2017
Hypertrace	Traceable Community License Agreement (1.0)(Hypertrace, 2023a)	Java*, Go*, Python*, NodeJs*, C++*, .NET* (Hypertrace, 2023a,b)	Hypertrace (2023a)	Free	2020
Honeycomb.io	Apache-2.0, MIT (Honeycomb.io, 2023b)	Go*, Java*, .NET*, NodeJS*, Python*, Ruby*, JavaScript, Python (Honeycomb.io, 2023b,d)	Honeycomb.io (2023b)	Free; Quote	2016
Instana	Proprietary, Apache-2.0, GPL-2.0, MIT (Instana, 2022e)	Shell, JavaScript, Go, Java*, Python*, .NET*, Clojure*, Kotlin*, Python*, PHP*, Scala*, NodeJS*, Ruby* (Instana, 2022e,c)	Instana (2022e)	$75/93.80 per host, per month	2015
Jaeger	Apache-2.0 (Jaeger contributors, 2022e)	Go*, Java*, NodeJS*, Python*, C++*, C#* (Jaeger contributors, 2022e; Jaeger, 2023)	Jaeger contributors (2022e)	Free	2016
Kamon	Apache-2.0 (Kamon, 2023b)	Java*, Scala* (Kamon, 2023b,f)	Kamon (2023b)	Free; $89/299 per month; Quote	2017
LightStep	Proprietary, Apache-2.0, BSD-2-Clause, BSD-3-Clause, CC-BY-SA-4.0, MIT (Lightstep, 2022e)	Go*, JavaScript, Python*, Java*, HCL, .NET*, NodeJS* (Lightstep, 2022e, 2023)	Lightstep (2022e)	Free; $100 per active service per month; Quote	2015
Logit.io	MIT (Logit.io, 2023b)	.NET*, Go*, NodeJS*, Python*, Ruby*, JavaScript, Shell (Logit.io, 2023b,a)	Logit.io (2023b)	$0.74 per GB; $5 per million spans per month; $2.80 per 1000 DPM	2013
Lumigo	Apache-2.0(Lumigo, 2023b)	Python*, NodeJS*, Java, Go (Lumigo, 2023b,a)	Lumigo (2023b)	$99/299 per month; Quote	2018
New Relic	Apache-2.0 (Relic, 2023a)	C*, Go*, Java*, .NET*, NodeJS*, PHP*, Python*, Ruby*, JavaScript, Shell (Relic, 2023a,b)	Relic (2023a)	$0.30 per GB Standard Ingest Cost Beyond Free Limits; + 0.50 per GB Data Plus Ingest Cost; +$49 per month Core Users; +Quote	2008
Ocelot	Apache-2.0 (Novatec Consulting, 2022b)	Java*, JavaScript (Novatec Consulting, 2022b; Ocelot, 2023)	Novatec Consulting (2022b)	Free	2018
OpenCensus	Apache-2.0 (OpenCensus, 2023b)	Python*, NodeJS*, Go*, C#*, C++*, Erlang*, Java* (OpenCensus, 2023b,c)	OpenCensus (2023b)	Free	2017
OpenTelemetry	Apache-2.0 (The OpenTelemetry Authors, 2022)	C++*, .NET*, Erlang*, Go*, Java*, JavaScript*, PHP*, Python*, Ruby*, Rust*, Swift* (The OpenTelemetry Authors, 2022; OpenTelemetry, 2023a)	The OpenTelemetry Authors (2022)	Free	2019
Sentry	BSL-1.1 (Sentry, 2023b)	.NET*, JavaScript*, NodeJS*, Python*, PHP*, Rust*, Java*, Go* (Sentry, 2023b,f)	Sentry (2023b)	$0/26/80 per month; Quote	2012
Splunk	Apache-2.0 (Splunk, 2023a)	Python*, Java*, NodeJS*, .NET*, Go*, Ruby*, PHP* (Splunk, 2023a,b)	Splunk (2023a)	$15 per host/month	2003
Signoz	MIT (SigNoz, 2023a)	Java*, Python*, JavaScript*, Go*, PHP*, .NET*, Ruby*, Elixir*, Rust* (SigNoz, 2023a,b)	SigNoz (2023a)	Free; $200 per month; Quote	2020
Site24 × 7	BSD-2-Clause, MIT (Site24x7, 2023d)	Java*, .NET*, Ruby*, PHP*, NodeJS*, Python* (Site24x7, 2023d,b)	Site24x7 (2023d)	€9/39/99/225 per month	2006
SkyWalking	Apache-2.0 (Apache SkyWalking contributors, 2022)	Java*, Python*, NodeJS*, Lua*, JavaScript*, Rust*, PHP* (Apache SkyWalking contributors, 2022; Apache SkyWalking, 2023)	Apache SkyWalking contributors (2022)	Free	2015
StageMonitor	Apache-2.0 (Stagemonitor contributors, 2022a)	Java*, HTML, JavaScript (Stagemonitor contributors, 2022a; Stagemonitor, 2023)	Stagemonitor contributors (2022a)	Free	2013

(continued on next page)

Table 2 (*continued*).

Tool	License	Programming language	Repo.	Pricing	Created
Tanzu	Apache-2.0 (Tanzu, 2023a)	Java*, C++*, Go*, .NET*, Python* Ruby Tanzu (2023a,b)	Tanzu (2023a)	Free	2019
Uptrace	BSD-2-Clause, Apache-2.0 (UpTrace, 2023a)	Go*, NodeJS*, .NET*, Ruby*, Python* (UpTrace, 2023a,d)	UpTrace (2023a)	$0.10/0.09/0.08/0.07/0.06/0.05 per GB for 50/1000/2500/4250 /6000/8000GB with $5+/100+ /200+/300+/400+/500+ budget	2021
Victoriametrics	Apache-2.0 (VictoriaMetrics, 2023c)	Go*, JavaScript* (VictoriaMetrics, 2023c,b)	VictoriaMetrics (2023c)	Free; Quote	2018
Zipkin	Apache-2.0 (Zipkin contributors, 2022b)	C#*, Go*, Java*, JavaScript*, Ruby*, Scala*, PHP* (Zipkin contributors, 2022b; Zipkin, 2023)	Zipkin contributors (2022b)	Free	2012

Table 3
Identified architectural components.

Tool	Libraries	Agent	Transport	Collector	Storage	Data processing	UI
Appdash			×	×	×		×
AppDynamics(AppDynamics, 2022c)	×[a]	×		×[b]	×[b]	×[b]	×[b]
Containiq		×					×
Datadog(Datadog, 2022b)	×	×		×[c]	×[c]	×[c]	×
Dynatrace	×	×	×	×	×	×	×
ElasticAPM(Elastic, 2022a)	×[d]	×		×[e]	×	×	×
Grafana tempo	×[r]	×[r]		×	×		×
Haystack	×	×		×			×
Hypertrace	×[g]	×		×		×	×
Honeycomb.io	×[g]	×		×			×
Instana(Instana, 2022g)	×[f]	×		×[c]	×	×	×
Jaeger(Jaeger contributors, 2022b)	×[g]	×	×	×	×	×	×
Kamon	×	×		×			×
LightStep(Lightstep, 2022f)	×[g]	×[h]		×[i]	×[i]	×[i]	×[i]
Logit.io					×	×	×
Lumigo	×	×		×		×	×
New Relic	×	×	×	×	×	×	×
Ocelot(Novatec Consulting, 2022c)		×					
OpenTelemetry	×	×		×		×	
Sentry	×	×	×	×	×	×	×
Splunk	×	×		×	×	×	×
Signoz	×	×	×	×	×	×	×
Site24 × 7	×	×	×	×			×
SkyWalking(Apache Skywalking contributors, 2022d)	×[j]	×	×	×[k]	×	×[l]	×
StageMonitor(Stagemonitor contributors, 2022b)		×				×[m]	

(*continued on next page*)

Table 3 (*continued*).

Tool	Libraries	Agent	Transport	Collector	Storage	Data processing	UI
Tanzu (VMware, 2022b)	×[a]	×	×[n]	×[o]	×	×	×
Uptrace	×			×	×		×
Victoriametrics	×	×					
Zipkin(Zipkin contributors, 2022a)	×	×[p]		×[q]	×[q]	×[q]	

[a]called *SDK*
[b]called *Controller*
[c]called *Backend*
[d]called *Tracer API*
[e]called *APM Integration*
[f]depending on the technology to monitor, the documentation calls the data collection component *library*, *sensor*, *tracing SDK*, or *collector*
[g]relies on the APIs and SDKs provided by the OpenTelemetry project (The OpenTelemetry Authors, 2022)
[h]called *Microsatellites*
[i]called *Engine*
[j]called *Probes*
[k]called *Receiver cluster*
[l]called *Aggregator cluster*
[m]called *Widget*, only in web applications, for debugging purposes
[n]called *Proxy*
[o]called *Service*
[p]called *Reporter*
[q]called *Server*
[r]relies on Grafana Agent

2022d)), selecting a data collection site to respect privacy regulations (as for, e.g., Datadog (Datadog, 2022c)) and setting up a project. In the case of ElasticAPM, which uses a combination of tools within the backend, these tools have to be configured and connected with each other.

3. *Agent setup and application instrumentation*: All tools require the installation of agents and their configuration (AppDynamics, 2022d; Datadog, 2022c; IBM, 2022a; Lightstep, 2022c; Elastic, 2022b; Jaeger contributors, 2022d; Apache Skywalking contributors, 2022g; Wavefront, 2022a; Zipkin contributors, 2022f). All tools offer a variety of agents that are able to either (a) automatically instrument an application or (b) allow developers to manually instrument it. *Automatic instrumentation* means that the target application is modified in such a way that it logs and transmits the required data to the agent without manual work, *manual instrumentation* means that the developer has to modify the code manually using the provided library to send what is needed to the agent. The agents have to be configured that they send the data to the backend, linking the data to a particular project. Ocelot and StageMonitor are agents and require the configuration of a backend, e.g., InfluxDB.[17]
4. *Data processing*: once the data collection is in place, the various tools (see Fig. 3) allow three different types of

[17] https://www.influxdata.com

Table 4
Metrics collected by the identified tools (RQ_1).

Tool	Traces	Metrics	Logs
Appdash	×(AppDash, 2023a)		×(AppDash, 2023a)
AppDynamics	×(AppDynamics, 2022e)	×(AppDynamics, 2022f)	×(AppDynamics, 2022g)
Containiq	×(ContainIQ, 2023b)	×(ContainIQ, 2023b)	×(ContainIQ, 2023b)
Datadog	×(Datadog, 2022d)	×(Datadog, 2022e)	×(Datadog, 2022f)
Dynatrace	×(Dynatrace, 2023b)	×(Dynatrace, 2023g)	×(Dynatrace, 2023f)
ElasticAPM	×(Elastic, 2022c)	×(Elastic, 2022d)	×(Elastic, 2022e)
Grafana tempo	×(Tempo, 2023c)	×(Tempo, 2023c)	×(Tempo, 2023c)
Honeycomb.io	×(Honeycomb.io, 2023a)	×(Honeycomb.io, 2023e)	×(Honeycomb.io, 2023g)
Hypertrace	×(Hypertrace, 2023d)		
Haystack	×(Haystack, 2023c)	×(Haystack, 2023d)	×(Haystack, 2023e)
Instana	×(Instana, 2022d)	×(Instana, 2022b)	×(Instana, 2022f)
Jaeger	×(Jaeger contributors, 2022f)	×(Jaeger contributors, 2022g)	×(Jaeger contributors, 2022h)
Kamon	×(Kamon, 2023h)	×(Kamon, 2023e)	×(Kamon, 2023c)
LightStep	×(Lightstep, 2022a)	×(Lightstep, 2022d)	×(Lightstep, 2022b)
Logit.io	×(Logit.io, 2023d)	×(Logit.io, 2023e)	×(Logit.io, 2023f)
Lumigo	×(Lumigo, 2023f)	×(Lumigo, 2023d)	×(Lumigo, 2023c)
New Relic	×(Relic, 2023g)	×(Relic, 2023f)	×(Relic, 2023e)
Ocelot	×(Novatec Consulting, 2022g)	×(Novatec Consulting, 2022e)	×(Novatec Consulting, 2022d)
Opencensus	×(OpenCensus, 2023e)	×(OpenCensus, 2023d)	
OpenTelemetry	×(OpenTelemetry, 2023d)	×(OpenTelemetry, 2023c)	×(OpenTelemetry, 2023b)
Sentry	×(Sentry, 2023h)	×(Sentry, 2023d)	×(Sentry, 2023c)
Splunk	×(Splunk, 2023e)	×(Splunk, 2023c)	
SkyWalking	×(Apache Skywalking contributors, 2022e)	×(Apache Skywalking contributors, 2022b)	×(Apache Skywalking contributors, 2022a)
Site24 × 7	×(Site24x7, 2023c)	×(Site24x7, 2023e)	×(Site24x7, 2023f)
Signoz	×(SigNoz, 2023e)		×(SigNoz, 2023c)
StageMonitor	×(Stagemonitor contributors, 2022a)	×(Stagemonitor contributors, 2022d)	×(Stagemonitor contributors, 2022c)
Tanzu	×(VMware, 2022c)	×(VMware, 2022a)	×(Wavefront, 2022b)
Uptrace	×(UpTrace, 2023c)	×(UpTrace, 2023f)	×(UpTrace, 2023e)
Victoriametrics	×(VictoriaMetrics, 2023f)	×(VictoriaMetrics, 2023d)	
Zipkin	×(Zipkin contributors, 2022b)		

data processing: (a) exploratory data analysis querying the collected data or visualizing it in charts (b) pre-defining frequently needed queries and charts and storing and presenting them in form of dashboards (c) pre-defining queries and defining thresholds to obtain alerts if certain conditions are met.

Data. As mentioned in the introduction, distributed tracing aims to track requests as they flow through the services of a distributed system. Therefore, foremost, distributed tracing tools collect data about textittraces, i.e., how a request traverses different services. In addition, tracing tools often also collect (The OpenTelemetry Authors, 2022) *metrics* and *logs*: metrics are measurements that describe the state of the observed system, e.g., the `memory` utilization at timestamp `2022-07-03T18:53:55Z` of microservice `1`. Logs are messages that developers emit with their code to inform about important events, e.g., that the event `ItemDeleted` was initiated by user `7` and occurred with the timestamp `2022-07-03T18:53:55Z`.

Table 4 reports which of the three aspects – tracing, metrics, and logs – are collected by the analyzed tools. All tools collect traces, which is obvious as we are looking at tracing tools, and all tools except Appdash, Hypertrace, Signoz and Zipkin allow the additional collection of metrics. Regarding log data, only Hypertrace, Opencensus, Splunk, Victoriametrics and Zipkin do not provide log data. These additional data is linked to the component in which the current trace was recorded and can be helpful when observing a trace. Next to each cross we provide the point in the documentation describing the presence of a particular data collection capability.

Interoperability. To evaluate interoperability, we looked at three aspects: the presence of a documented API, the support for OpenTelemetry (The OpenTelemetry Authors, 2022), and if it is possible to self-host the tool, i.e., to install everything locally.

OpenTelemetry is a "vendor-neutral open-source Observability framework for instrumenting, generating, collecting, and exporting telemetry data such as traces, metrics, logs (The OpenTelemetry Authors, 2022)". We found that it is supported by all tools except StageMonitor.

The results are reported in Table 5. The gray crosses indicate that self-hosting is implicitly possible because the entire tool is provided with an OpenSource license. Please note, that it might be complex to perform a local installation, but technically, it is possible.

From the point of view of interoperability, also the data provided in Table 2 can be of relevance: this table describes the used licenses of the tool and the used programming languages.

4.2. Tool popularity (RQ_2)

Tool popularity based on peer-reviewed literature. For each tool, we searched the Peer-reviewed publications that mentioned it. We found that only three Open Tracing Tools have been cited by more than 10 papers: Zipkin 29 times, Jaeger 18 times, and LightStep 10 times. The other considered tools have been cited less than 10 times.

Tool popularity based on Online Media. Shown in Fig. 4, the search results from three different technology-based social media platforms, i.e., StackOverflow, Medium, and DZone on each tool reflect their varied popularity.

First of all, Splunk, among the 30 tools, is the most popular considering the collective volume of the textual data from the three sources. The second and third most popular tools are Haystack and Sentry, which have similar levels of data volume compared to Splunk. New Relic and Datadog are also considerably popular and rank at 4th and 5th. However, the tools from 6th to 10th, i.e., Zipkin, Jaeger, OpenTelemetry, Dynatrace, and AppDynamics have only no more than 1/3 of data volume compared to the top ones. Furthermore, the bottom 20 tools have very

Table 5
Features about interoperability (RQ_1).

Tool	API	OpenTelemetry support	Self-hosting
Appdash		×(AppDash, 2023a)	
AppDynamics	×(AppDynamics, 2022h)	×(AppDynamics, 2022i)	×(AppDynamics, 2022j)
Containiq		×(ContainIQ, 2023a)	×(ContainIQ, 2023d)
Datadog	×(Datadog, 2022g)	×(Datadog, 2022h)	
Dynatrace	×(Dynatrace, 2023a)	×(Dynatrace, 2023h)	×(Dynatrace, 2023d)
ElasticAPM	×(Elastic, 2022f)	×(Elastic, 2022g)	×
Grafana tempo	×(Tempo, 2023e)	×(Tempo, 2023d)	
Honeycomb.io	×(Honeycomb.io, 2023c)	×(Honeycomb.io, 2023f)	
Hypertrace		×(Hypertrace, 2023c)	
Haystack		×(Haystack, 2023f)	
Instana	×(Instana, 2022a)	×(IBM, 2022b)	×(IBM, 2022c)
Jaeger	×(Jaeger contributors, 2022a)	×(Jaeger contributors, 2022c)	×
Kamon	×(Kamon, 2023a)	×(Kamon, 2023g)	
LightStep	×(Lightstep, 2022a)	×(Lightstep, 2022d)	×(Lightstep, 2022b)
Logit.io	×(Logit.io, 2023c)	×(Kamon, 2023d)	
Lumigo		×(Lumigo, 2023e)	
New Relic	×(Relic, 2023c)	×(Relic, 2023d)	
Ocelot		×(Novatec Consulting, 2022a)	×
Opencensus	×(OpenCensus, 2023a)	×	
OpenTelemetry	×	×	
Sentry	×(Sentry, 2023a)	×(Sentry, 2023e)	×(Sentry, 2023g)
Splunk	×(Splunk, 2023d)	×(Sentry, 2023h)	
SkyWalking	×(Apache Skywalking contributors, 2022f)	×[b] (Apache Skywalking contributors, 2022c)	×
Site24 × 7	×(Site24x7, 2023a)	×[c] (Site24x7, 2023g)	
Signoz		×(SigNoz, 2023d)	
StageMonitor		×(Stagemonitor contributors, 2022a)	×
Tanzu	×(Wavefront, 2022d)	×(Wavefront, 2022c)	
Uptrace		×(UpTrace, 2023b)	
Victoriametrics	×(VictoriaMetrics, 2023a)	×[d] (VictoriaMetrics, 2023e)	
Zipkin	×(Zipkin contributors, 2022e)	×[a]	×

[a]OpenTelemetry data can be exported to Zipkin Zipkin contributors (2022c).
[b]Metrics can be reported to the OpenTelemetry receiver or imported using OpenTelemetry exporter; Traces and logs are not supported.
[c]Site24 × 7's support for OpenTelemetry is currently in development.
[d]OpenTelemetry support is requested within an issue.

limited social media coverage and have at best about 300 articles in all three sources. There are 14 tools having no more than 100 articles/posts in total.

In Fig. 4, we reported the social media content distribution for the 30 retrieved tools. As we can see, 10 out of 30 take up 90.2% of the articles of all the tools (Fig. 4). However, when checking the obtained data volume for each tool, we found only 10 tools have more than 600 data points (Naushan, 2020). The discussion frequency (number of questions or answers each month) of each tool on Stack Overflow shows that only the selected tools have been drawing noticeable attention (shown in Fig. 5). Unfortunately, 6 tools (i.e., ContainIQ, Hypertrace, Logit.io, Lumigo, OpenCensus and Sentry.io) did not provide available data points. The reason is likely due to the lack of discussion and their limited popularity.

4.3. Benefit and issues (RQ_3 and RQ_4)

According to the social media content distribution described in Fig. 4, we proceeded to answer to RQ_3 and RQ_4 only for the tools with sufficient data (as explained in Section 3.3). The final list is reported in Table 6.

Following the approach described in Section 3, here we describe the obtained results for each step.

Step 1: Preprocessing. We pre-processed the crawled textual data by retaining only the natural language sentences. Herein we eliminated unnecessary content, such as the source code, URLs, publishing date and author info, etc. For Medium articles, we started eliminating the heading of the article that includes the publishing date and author info by splitting the string at the common last character of the part "min read" and selecting the later part. Subsequently, we used the sentence tokenizer from the Natural Language Toolkit (NLTK)[18] to obtain the list of sentences from each article. As the tokenizer does not identify the source code or URLs, we eliminated them by selecting only the sentences ending with a period, an exclamation mark, or a question mark.

First, we crawled data from social media, including, StackOverflow (16 223 questions and 17 811 answers), Medium (2 028 articles), and Dzone (1 623 posts). We used *langdetect* Python package.[19] to filter the non-English texts and obtained 37 685

Table 6
The 10 tools considered for RQ_3 and RQ_4.

Tool name	Web site
AppDynamics	https://www.AppDynamics.com
Datadog	https://www.Datadoghq.com/
Dynatrace	https://www.Dynatrace.com/
Haystack	https://www.Haystackteam.com/
Jaeger	https://www.Jaegertracing.io
New Relic	https://opensource.newrelic.com/
OpenTelemetry	https://opentelemetry.io/
Sentry	https://Sentry.io/
Splunk	https://dev.Splunk.com/
Zipkin	https://zipkin.io

[18] http://www.nltk.org/
[19] https://pypi.org/project/langdetect/

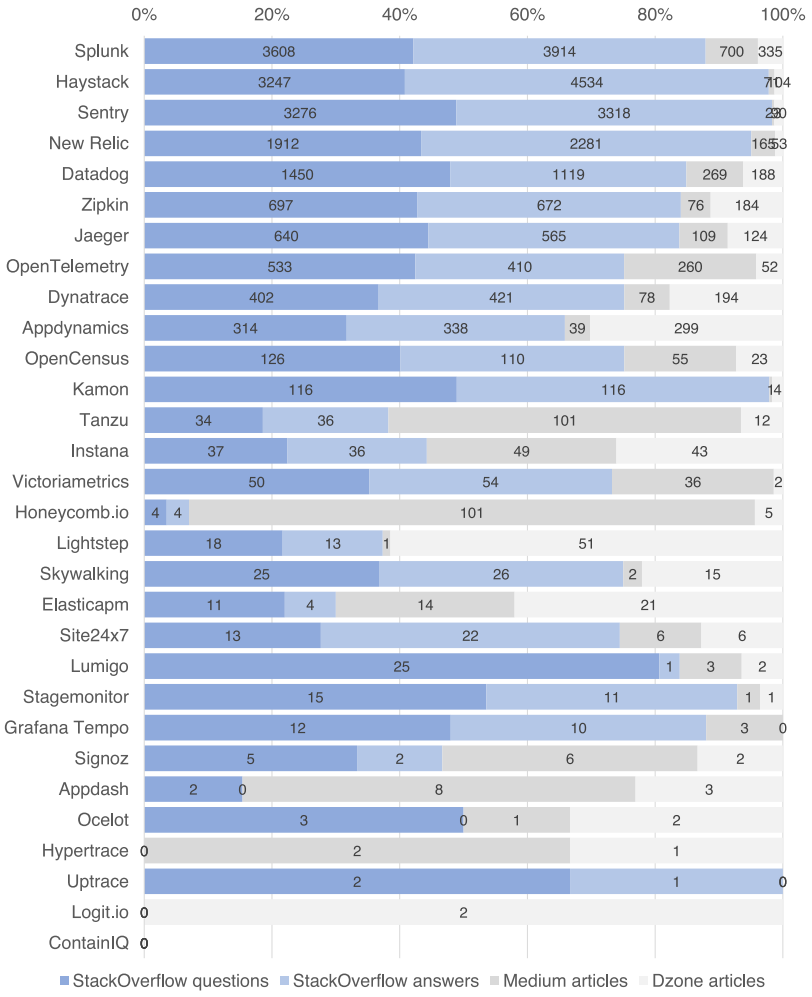

Fig. 4. Social media content distribution (RQ_2).

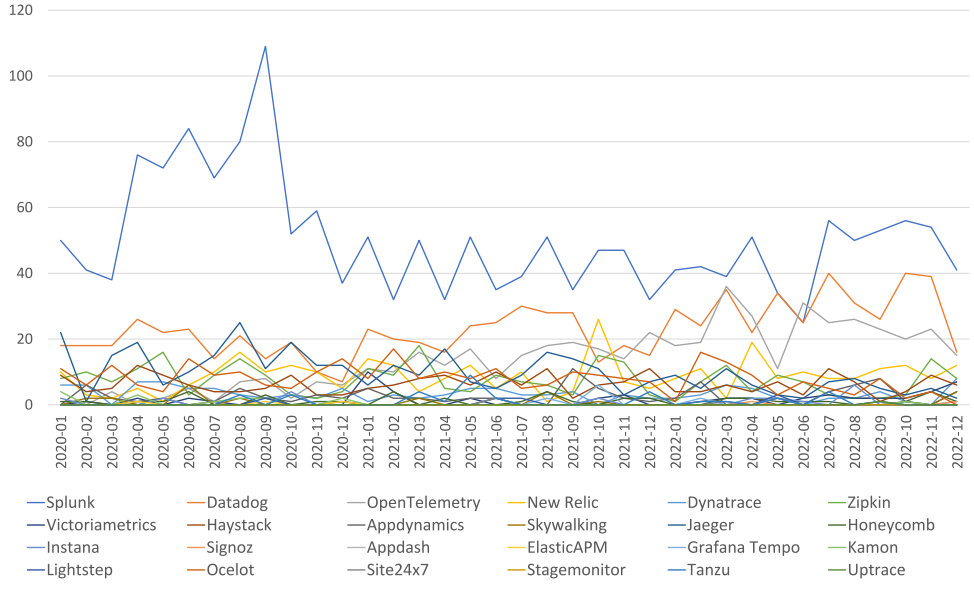

Fig. 5. The latest discussion frequency on stack overflow for each tool.

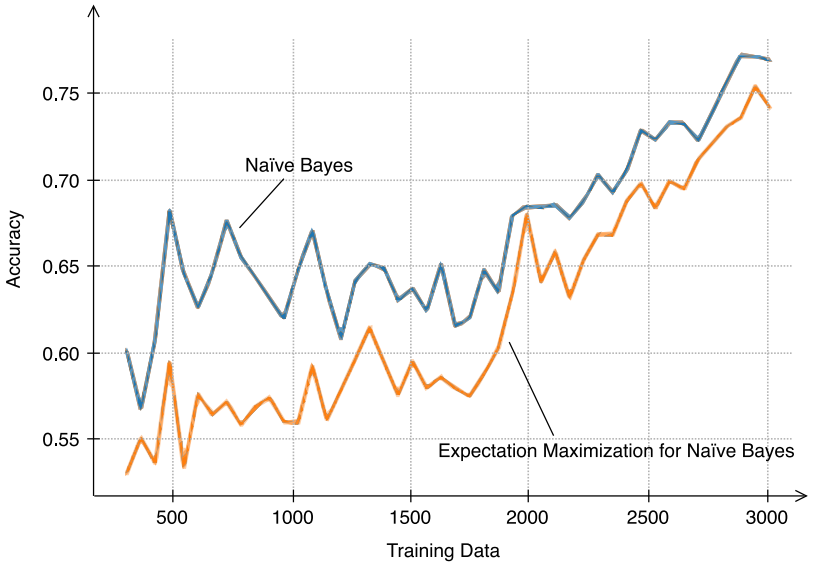

Fig. 6. Testing informative text classifier accuracy (RQ_3 and RQ_4).

text data points, including 17 572 StackOverflow questions and 16 079 answers, 1 790 Medium articles and 1 623 Dzone posts. Furthermore, we filter the source code and html markdowns from each text using adapted *html2text* package.[20] We obtained 338 070 sentences using the NLTK sentence tokenizer (StackOverflow questions: 110 524, answers: 65 038, Medium: 65 632, Dzone: 106 876).

Step 2: Filtering. Herein, we identified the informative sentences using a Naïve Bayes (NB) classifier and the Expectation Maximization for Naïve Bayes (EMNB) classifier (Nigam et al., 2000). The selection shall be based on the accuracy comparison of these two classifiers with the obtained dataset. First, we manually labeled a sufficient number of training data including 50% informative sentences and 50% half non-informative ones. The selection criteria of informative sentences are that the sentence must explicitly present: (1) the benefits/features of the tools or (2) the issues of the tools. With an increasing number of training and testing data, the two classifiers shall be respectively trained and compared with the F1-score using a 5-fold cross validation.

In this study, from the 338 070 sentences obtained previously, we manually selected 3 000 training data, including 1 500 informative sentences and 1 500 non-informative ones. To evaluate the performance of informativeness filtering, with a series of experiments, we compared the results of the NB algorithm and the EMNB algorithm with 3 000 training data. We inspected the accuracy comparison of the two classifiers with different amounts of data starting from 200 to 3 000 with an incremental step of 20. The test data ratio is set as default (0.25). Fig. 6 shows that with the given training data, NB performs better than EMNB with the accuracy can reach as high as 0.76. Thus, we adopted the NB classifier for filtering the informative sentences. Using the classifier trained by the 3 000 training data, we obtained 158 219 informative sentences.

Step 3: Topic Modeling. To detect the topics of a set of text using an LDA topic modeling approach (Blei et al., 2003), a number of preprocessing steps are required, which include: removing punctuation, removing extra space, restoring the word to its root form (lemmatization), remove stopwords, and build the bigram and trigram models.

Furthermore, to find the best topic number for each review subset, we conducted a series of experiments for each set testing with the topic numbers ranging from 2 to 40. We used topic coherence to represent the quality of the topic models. Topic coherence measures the degree of semantic similarity between high scoring words in the topic. A high coherence score for a topic model indicates the detected topics are more interpretable. Thus, by finding the highest topic coherence score, we can decide the most fitting topic number. Herein, we use c_v coherence measure, which is based on a sliding window, one-set segmentation of the top words, and an indirect confirmation measure that uses normalized pointwise mutual information (NPMI) and the cosine similarity (Syed and Spruit, 2017). Note that we pick the model that has the highest c_v value before flattening out or a major drop, in order to prevent the model from over-fitting.

Shown in Fig. 7, we built topic models using LDA with the number of topics from 2 to 20 for text data. A clear turning point from the local highest value is at 6. It is possible the coherence score reaches even higher when selecting topic numbers larger than 20. However, such a phenomenon is caused by the over-fitting models and shall be ignored. Thus, the topic number was determined as 6.

Therefore, with the LDA topic model, we detected the 6 topics as follows: based on the allocated keywords in probability order, together with the overall term frequency as a reference, two domain experts synthesized their interpretation of the topics. The extracted topics are: *Usability*, *Development*, *Architecture*, *Tracing*, *Measurement*, *Deployment & Integration*.

The list of topics and the according lists of indicator keywords are shown in Table 7.

Step 4: Topic Mapping. With the obtained LDA topic model, we then mapped each of the informative sentences to one of the topics to which it was most likely related. Shown in Fig. 8, the numbers of topic-related sentences from the articles on each tool are summarized. Compared with Fig. 4, the number of informative sentences for each tool correlated to that of the article crawled proportionally. To be noted, we obtained much fewer informative sentences on Haystack than Splunk although they have a similar amount of article-level data points. The reason is that *"needle in a Haystack"* is a classic algorithm problem metaphor for "checking a string contains another string", which has drawn heated discussion in StackOverflow. Meanwhile, "Haystack" can also be linked to the modular search for Django.[21] All such texts should

[20] https://pypi.org/project/html2text/

[21] http://Haystacksearch.org/

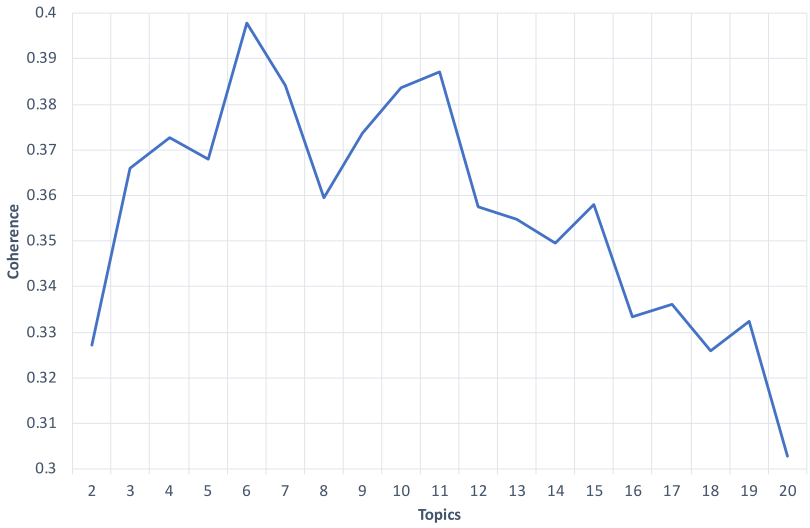

Fig. 7. Topic number detection (RQ_3 and RQ_4).

Table 7
Topic interpretation with indicator keywords (RQ_3 and RQ_4).

Topic	Indicator keywords
Usability	'user', 'support', 'developer', 'security', 'build', 'performance', 'production', 'design', 'use', 'need', etc.
Development	'spring', 'team', 'software', 'development', 'implement', 'issue', 'develop', 'time', 'handle', 'process', etc.
Architecture	'architecture', 'distribute', 'scale', 'observability', 'business', 'customer', 'solution', 'pattern', 'release', 'feature', etc.
Tracing	'trace', 'source', 'code', 'framework', 'message', 'open', 'follow', 'alert', 'spring_boot', 'transaction', etc.
Measurement	'microservice', 'application', 'request', 'log', 'use', 'event', 'metric', 'server', 'kubernete', 'api', etc.
Deployment & Integration	'application', 'service', 'monitor', 'cloud', 'deploy', 'deployment', 'infrastructure', 'integration', 'manage', 'environment', etc.

have been classified as "non-informative". Displayed in Fig. 8 *Deployment & Integration* is the most dominant topic for Datadog and New Relic. For some other tools, e.g., Haystack, Dynatrace and AppDynamics, *Usability* topic is concerned the most.

Step 5: Opinion Mining. Using the VADER method (Gilbert and Hutto, 2014), we can assess the sentiment of each informative sentence and furthermore the overall sentiment of each tool in terms of each topic. Herein, we take into account the percentage of positive, neutral, and negative sentences without considering the according sentiment strength. The percentage sentences in different sentiments for each tool on each topic is shown in Fig. 9. To determine the benefits and issues in terms of each extracted aspect, i.e., topic, we compared each set of sentiment percentages to the average sentiment percentage of all sentences.

Shown in Table 8, the percentage of the different sentiments for each tool was used as the reference. Therefore, we determined each topic for each tool being either a benefit, an issue, or a neutral opinion according to the following criteria.

- If the percentage of positive sentences is higher than average and the percentage of negative ones lower than average, the topic is considered as a benefit for the tool.
- If the percentage of positive sentences is lower than average and the percentage of negative ones higher than average, the topic is considered as an issue for the tool.
- For any other circumstances, the topic is considered disputed.

Table 8
Topic sentiment average percentage for each tool (RQ_3 and RQ_4).

Tool	Positive	Neutral	Negative
AppDynamics	47.4%	36.6%	16.0%
Datadog	43.3%	42.2%	14.6%
Dynatrace	45.7%	39.7%	14.6%
Haystack	38.0%	40.3%	21.7%
Jaeger	40.8%	46.6%	12.6%
New Relic	32.5%	47.4%	20.1%
OpenTelemetry	41.9%	46.2%	11.9%
Sentry	30.8%	36.6%	32.6%
Splunk	44.7%	40.1%	15.2%
Zipkin	41.1%	45.8%	13.1%

Table 9
Benefits and issues for each tool regarding topics (RQ_3 and RQ_4).

Criteria	AppDynamics	Datadog	Dynatrace	Haystack	Jaeger	New Relic	OpenTelemetry	Sentry	Splunk	Zipkin
Architecture	+	+	+	+	+	+	+	+	+	+
Deployment & Integration				+	+			+		
Development	−		−	−	−	−		−		
Measurement	−	−	−	−	−	−	−	−	−	−
Tracing		−	−			−		−		
Usability		+		+	+	+	+	+		+

+ Benefit.
− Issue.

Thus, according to the criteria, the according benefits and issues for each tool in terms of the topics can be summarized as Table 9, which answers RQ_3 and RQ_4.

We want to point out, that the "benefits" and "issues" obtained through the analysis of sentiments and discussion topics are complementary to the distinctive features of each tool (**RQ**$_1$): the *actual* benefit or issue of using a tool in a given environment needs to be decided to know the context and the requirements of a given project. The benefits and issues identified in this section complement the picture and want to advise the practitioner that

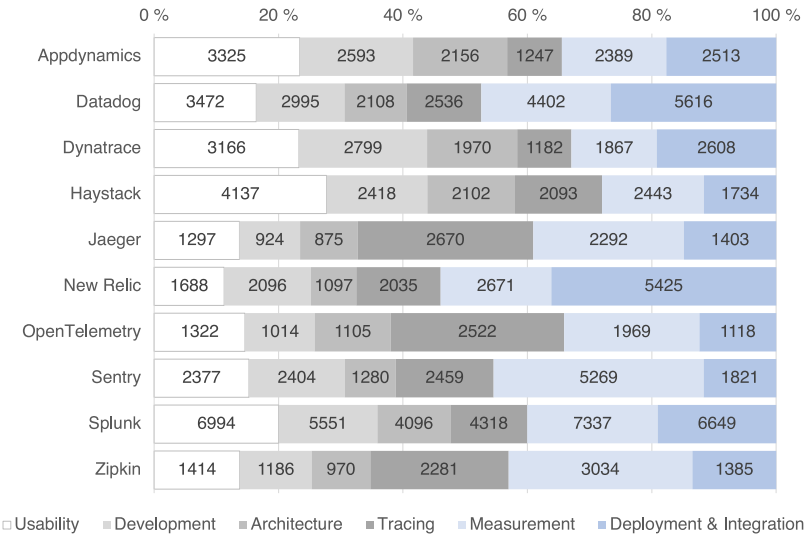

Fig. 8. Topic frequency for each tool (RQ_3 and RQ_4).

Table 10
Sub-topic numbers for each topic of each tool.

Criteria	AppDynamics	Datadog	Dynatrace	Haystack	Jaeger	New Relic	OpenTelemetry	Sentry	Splunk	Zipkin
Architecture	4	6	2	5	4	2	4	8	7	2
Deployment & Integration	3	3	6	6	2	4	5	2	3	6
Development	7	5	2	2	4	4	4	8	2	7
Measurement	3	3	6	4	4	2	6	2	3	3
Tracing	3	9	5	4	4	3	3	3	4	2
Usability	9	4	3	4	3	4	4	8	3	6

intends to use a given tool, to put particular care into the aspects we identified.

To further investigate the benefits and issues of each selected tool with details, we continued to use LDA topic modeling to extract the latent topics in each of the six criteria. For each subset of texts for each criterion of each tool (i.e., 6 topics × 10 tools = 60 subsets), We adopted the same "topic-modeling-and-mapping" procedure as previously described. By conducting the same experiments to find the best topic number for each subset, we found the topic numbers (shown in Table 10 and trained the topic model for each subset.

Same as the previous topic modeling step, we obtained a set of keywords sorted by the relevance for each of the 60 sub-topic. In addition, we used ChatGPT (OpenAI, 2023) to support the effective interpretation and summarization of the topics based on these keywords. ChatGPT is an artificial intelligence chatbot developed by OpenAI, which uses foundational large language models (LLMs) and is fine-tuned via supervised and reinforcement learning techniques. Though a newly emerging technique, ChatGPT has quickly gained overwhelmingly world-wide attention from both industry and academia. Specifically, regarding the facilitation of text summarization, many early-stage studies have investigated the use of ChatGPT for such tasks (Yang et al., 2023; Luo et al., 2023).

Herein, for this topic extraction task, we adopted the newly released GPT-4 model.[22] Compared to the legacy GPT-3.5 model, the new model has much higher reasoning capacity and conciseness. We initiate the topic extraction by entering a series of structured requests formatted as follows.

"Extract a short <MAIN TOPIC>-related topic for each of these lines of keywords.

<1st LINE OF KEYWORDS>
...
<nth LINE OF KEYWORDS>"

The replies received were also structured, corresponding to the requests above as follows.

"<1st TOPIC>: <Explanation>
...
<nth TOPIC>: <Explanation>"

For the purpose of validation, the first author and third author compare the AI-extracted topics and the original keyword lists. The sub-topics of each of the subsets are summarized in Tables 11 and 12.

4.3.1. RQ_3. What benefits are achieved by adopting Open Tracing Tools?

The results, summarized in Table 9, show that all the selected tools provide benefits in terms of *Architecture* based on the collective opinions of practitioners. On the other hand, 7 of the 10 tools, i.e., Datadog, Haystack, Jaeger, New Relic, OpenTelemetry, Sentry, and Zipkin, are positively received in terms of *Usability*. Haystack, Jaeger, and Sentry also receive positive feedback regarding *Deployment & Integration*. However, in terms of *Development*, *Tracing*, and *Measurement*, the opinions are more neutral or negative for all tools. To be emphasized, it does not mean none of these tools has any benefits for these aspects. It shows that the practitioners reflect more on their issues than the benefits regarding these aspects.

Furthermore, by adopting another round of topic modeling, we further investigated each tool's collectively positive and negative opinions regarding each main topic. By doing so, we can intuitively compare each tool's benefits in more detail. The benefits of each tool summarized by the practitioners' collective opinions are shown in the green texts of Tables 11 and 12.

22 https://openai.com/product/gpt-4

Fig. 9. Topic sentiment for each tool (RQ_3 and RQ_4).

Architecture. Shown in the Table 9, all tools are considered comparatively positively received, though some tools also received proportionally negative opinions on certain aspects. As shown in Tables 11 and 12, AppDynamics is the only tool that has no significant issues. Especially, considering the keywords of the sub-topics, AppDynamics has the benefits regarding architecture patterns and addressing the challenges of scaling, security, and adoption in microservice-based business applications by leveraging design principles, cloud technologies, and containerization, as well as enhancing user experiences of mobile apps. Datadog is considered to perform well for microservice architecture in general and especially in tackling challenges and responsibilities in building distributed software systems with DevOps and monitoring tools. Comparatively, Jaeger is received more positively in terms of the balancing cost and performance in managing resources and complexity in large-scale enterprise deployments and also the enhanced observability in a distributed microservice architecture. Similarly, OpenTelemetry and Zipkin also have benefits in building distributed observability solutions. Sentry

Table 11
Sub-topics for each topic of each tool (Part 1).

	Usability	Development	Architecture	Tracing	Measurement	Deployment&Integration
AppDynamics	1) Tool production; 2) User performance; 3) App development pipeline; 4) Downtime reduction; 5) Security in application architecture; 6) Flexible data configuration; 7) Microservice implementation; 8) Performance testing; 9) Scalability and reliability;	1) Microservice development; 2) Software testing processes; 3) Performance troubleshooting; 4) DevSecOps evolution; 5) Monitoring tools in DevOps; 6) Service integration and communication; 7) Reducing load times and costs;	1) Microservice architecture patterns; 2) Mobile app customer experience; 3) Cloud-based product architecture; 4) Observability in business applications;	1) Distributed tracing in microservices; 2) Alerting and logging; 3) Application performance and transaction tracing;	1) API and microservice metrics; 2) Measuring database and transaction performance; 3) Application performance monitoring;	1) Cloud and container deployment; 2) Application monitoring and integration; 3) Performance testing and continuous integration;
Datadog	1) Tool usability testing; 2) Performance monitoring and optimization; 3) Data pipeline management; Security and scalability in development environments;	1) DevOps and Team Collaboration; 2) Performance Issue Management; 3) Optimizing Kubernetes Resource Usage; 4) Automation in Deployment Processes; 5) Cloud-native Security Testing;	1) Microservices Architecture; 2) Latency Optimization in Distributed Tracing; 3) API Reliability and Observability; 4) Scalability and Observability in Customer-focused Services; 5) Containerization and Enterprise Growth; 6) Resource Management in High-Usage Environments	1) Integrating Tracing Frameworks; 2) Distributed Tracing 3) Incident Notification and Response; 4) Alert Configuration and Threshold Management; 5) Tracing in Containerized Environments; 6) Open-source Event Tracing with Prometheus; 7) Chaos Engineering and Observability; 8) User Request Tracing and Performance Testing; 9) Error Tracing and Debugging in Kubernetes;	1) Kubernetes Metrics and Logging; 2) Measuring Service Performance in Microservices; 3) Request-based Event Measurement;	1) Cloud Deployment and Performance Monitoring; 2) Chaos Engineering in Multi-layer Monitoring; 3) Kubernetes Deployment and Datadog Integration;
Dynatrace	1) Tool use and user performance 2) Performance management in software development 3) Security and continuous development	1) DevOps culture and collaboration 2) Proactive troubleshooting in software development	1) Cloud and open platform architectures 2) High-performance architecture for business applications	1) Opentelemetry in observability 2) Distributed tracing for optimizing response time 3) Automated alerting and incident management 4) Microservices and transaction tracing 5) Open-source tracing tools for performance analysis	1) Performance measurement in microservices 2) Improving API performance with test metrics 3) Monitoring and optimizing end-to-end service request time 4) Cross-platform performance measurement 5) Log analysis and event rate limiting in distributed systems 6) Enhancing app performance with load balancing and error management	1) Deployment and monitoring of enterprise applications 2) Comparing application performance management (APM) tools 3) Cloud-native application deployment and integration 4) Load and performance testing in continuous integration 5) Dynatrace for telemetry and reporting 6) Deployment and monitoring with Dynatrace
Haystack	1) Enhancing Customer Experience 2) Solving Usability Problems 3) Cybersecurity in Usability 4) Implementing Search Indexes	1) Addressing Memory Leaks and Performance 2) Improving Error Handling and Search Functionality	1) Scalability in Cloud Architecture 2) Balancing Performance and Security in Enterprise Architecture 3) Enhancing Search Capabilities in Modern Applications 4) Customer-centric Architectures for AI-powered Applications 5) Embracing Microservices in Mobile Application Development	1) Visualizing User Interactions 2) Tracing Code Execution and Error Handling 3) Integrating Elasticsearch and Haystack for Enhanced Search 4) Monitoring Email and Alert Response Times	1) Measuring API Performance 2) Evaluating Database Search Efficiency 3) Analyzing Server Performance and Error Impact 4) Monitoring Microservices and Application Metrics	1) Custom Service Deployment Challenges 2) Integrating Elasticsearch with Django 3) Optimizing Query Performance 4) Validator Integration and Performance 5) Deploying and Integrating Search Functionality 6) Monitoring and Scaling Deployment in Hadoop Environments
Jaeger	1) Enhancing user and developer experience with microservices 2) Security and performance in application development 3) Efficient software development	1) Memory management and CPU control in development environments 2) Streamlining the software development lifecycle 3) Building scalable, cloud-native applications 4) Addressing challenges in software development	1) Balancing cost and performance in cloud-native architecture 2) Evaluating integration strategies and vendor solutions in architectural design 3) Achieving high availability and scalability in distributed data services 4) Enhancing observability in distributed microservice architectures	1) Implementing Jaeger for performance monitoring and tracing 2) Enhancing request tracking in applications 3) Leveraging open-source tracing tools for distributed systems 4) Identifying and addressing errors in microservices	1) Analyzing end-to-end communication in service-based architectures 2) Evaluating application performance with log events and metrics 3) Measuring Kubernetes-based microservices performance 4) Assessing distributed service communication in microservice architectures	1) Streamlining cloud-native application deployment 2) Integrating and deploying Jaeger with Kubernetes for distributed tracing

has benefits on adaptable data architecture in terms of developing flexible data models and storage solutions to accommodate changing customer needs and market demands while ensuring seamless access and integration for developers and users, community-driven frameworks for business applications towards developing scalable and adaptable business applications with a focus on observability, user experience, and seamless online integration, and automated build and release processes. Splunk is also received positively for the development process for complex systems, leveraging agile methodologies and cross-functional teams to effectively manage complex software projects and rapidly adapt to changing industry requirements, as well as the optimization of resource estimation in software development via employing data-driven models and analytics to improve project management, technology infrastructure, and resource allocation for on-time, cost-effective software delivery.

Deployment & Integration. Regarding deployment and integration, many tools, e.g., AppDynamics, DataDog, Dynatrace, receive positive opinions on cloud and container deployment

Table 12
Sub-topics for each topic of each tool (Part 2).

	Usability	Development	Architecture	Tracing	Measurement	Deployment&Integration
New Relic	1) Enhancing tool performance in applications 2) Simplifying error resolution 3) Streamlining problem-solving with New Relic 4) Optimizing user experience in real-time production environments	1) Enhancing site performance and error handling 2) Monitoring memory and CPU usage for server optimization 3) Streamlining software development with New Relic 4) Identifying and resolving performance issues in applications	1) Optimizing app architecture for high scalability 2) Implementing modern architecture with New Relic	1) Enhancing function tracing with metrics 2) Leveraging transaction tracing for error detection 3) Analyzing page load times and tracing issues with New Relic	1) Monitoring application performance with New Relic metrics 2) Identifying and resolving error causes through measurement	1) Enhancing application performance with New Relic and Azure integration 2) Streamlining large-scale deployments with New Relic 3) Ensuring smooth deployment through containerization and team collaboration 4) Integrating New Relic agents for better monitoring and alerting
OpenTelemetry	1) Environment support and health 2) Application performance and observability 3) End-user management in cloud architecture 4) Open-source tools for resource optimization	1) Cloud infrastructure development 2) Adapting to changing needs in development 3) Troubleshooting in microservice architecture 4) Optimizing time and cost in DevOps	1) Navigating change in technology and deployment 2) Scaling microservice architecture 3) Leveraging OpenTelemetry for enhanced observability 4) Building distributed observability solutions	1) Enhancing user experience with tracing 2) Adopting open-source tracing tools 3) Integrating Jaeger and OpenTelemetry in data collection	1) Monitoring distributed microservices 2) Utilizing OpenTelemetry exporters for data collection 3) Error management and Opentelemetry metrics 4) Comprehensive metric collection in observability 5) Enhancing database performance with tracing 6) Optimizing database usage in microservices	1) Zero-downtime deployment with Jaeger and OpenTelemetry 2) Open-source tools for application testing and deployment 3) Streamlining cloud-native application management 4) Monitoring AWS services with custom metrics 5) Adopting new monitoring platforms for enhanced observability
Sentry	1) Environment and Usability 2) Developer Tools 3) Error Resolution 4) API Integration 5) Data Management 6) Security Testing 7) Performance Optimization 8) Scalable Security Solutions	1) Efficient App Development 2) Scrum and Automation 3) Error Handling in React Applications 4) Memory Management and Debugging 5) Device-Optimized Development 6) Scalable Infrastructure 7) Collaborative Development and Security 8) Continuous Integration and DevOps	1) Adaptable Data Architecture 2) Scalable and Resilient Table Architecture 3) Community-driven Frameworks for Business Applications 4) Secure Data Management and Analytics 5) Cloud-based App Development 6) Technology Selection and Tracking 7) Performance-driven Solution Development 8) Automated Build and Release Processes	1) Real-time Function Tracing 2) Error Monitoring and Communication 3) Open-source Tracing and Logging	1) Error Measurement and Monitoring 2) Performance Metrics in API Services	1) Seamless Application Deployment and Integration 2) Scalable Data Storage and Management
Splunk	1) Streamlining Continuous Deployment 2) Enhancing User Experience 3) Balancing Performance and Security	1) Optimizing Application Performance 2) Advancing Agile DevOps	1) Big Data and High-Performance Architecture 2) Enhancing Observability in Software Architectures 3) Handling Massive Data Sets 4) Scalable Enterprise DevOps 5) Agile Development for Complex Systems 6) Building Robust and Adaptable Applications 7) Optimizing Resource Estimation in Software Development	1) Automated Alert and Notification Systems 2) Pinpointing Errors in Complex Systems 3) Open Source Tracing in DevOps 4) Real-time Data Visualization and Monitoring	1) Evaluating Test Efficiency 2) Monitoring and Analysis of Microservices 3) Analyzing API and Service Performance	1) Optimizing Test Performance in Deployment 2) Adapting to Emerging Technologies and Custom Workloads 3) Streamlining Cloud-Based Application Deployment and Integration
Zipkin	1) Ensuring a healthy build pipeline 2) Organizational data ownership 3) Scalability and reliability in service architecture 4) Defining and measuring user metrics 5) Microservice support tools 6) Managing production aspects in service environments	1) Centralized monitoring tools 2) Exploring Spring framework and graph databases 3) Spring Boot for fast microservice development 4) Troubleshooting distributed microservices 5) Optimizing server and application performance 6) Automating development and deployment in the cloud 7) Cross-functional team collaboration in software development	1) Observability in distributed microservice architecture 2) Scaling data-driven applications	1) Error handling and tracing in service requests 2) Distributed tracing with Jaeger in Spring Boot applications	1) Monitoring and measuring application performance 2) Measuring microservice communication performance 3) Assessing microservice scalability and resource usage	1) Cloud-based microservice deployment and monitoring 2) Integrating RESTful services in public and private cloud environments 3) Simplifying cloud application deployment with Spring Boot and JHipster 4) Testing and monitoring microservices in production 5) Enhancing service deployment with Istio and Envoy sidecars 6) Kubernetes and Docker for streamlined microservice deployment

in general. Especially, according to the topic keywords, AppDynamics leans more towards the benefit of managing secure and efficient deployment of microservices in cloud-based and containerized infrastructure while Datadog focues on Streamlining application testing, deployment, and integration in scalable cloud environments to optimize the user experience. Dynatrace has the benefit of automating and managing containerized services with Kubernetes and continuous integration tools for secure, scalable, and efficient infrastructure management. Haystack has benefits in deploying and integrating search functionality. Jaeger, New Relic, and Splunk also have the benefits of streamlining deployment where Jaeger and New Relic are more praised for leveraging Kubernetes, containerization, and serverless architectures for efficient infrastructure management while New Relic for utilizing data from test reports, instance requests, and customer cases. On the other hand, OpenTelemetry is received more positively regarding leveraging Kubernetes, cloud vendors, and open-source resources for managing services and environments, while enhancing integration and traceability in development workflows, as well as Leveraging telemetry data and diverse software tools

for comprehensive infrastructure analysis while integrating with existing services and pipelines for seamless access and operation.

Development. Though shown in Table 9, many tools are received negatively in terms of development, which does not mean there are no positive aspects at all. For example, several tools have benefits regarding DevOps and collaboration (Dynatrace and DataDog), DevSecOps evolution (AppDynamics), and Advancing Agile DevOps (Splunk). Specifically, based on the topic keywords, many specific aspects of DevOps are considered the benefits of these tools, including addressing security challenges, infrastructure management and cost optimization, automation tools, and processes etc. Sentry and Zipkin also have the benefit of collaborative development and security as well as scalability. Furthermore, for Zipkin, its benefits also include centralized monitoring tools, specifically concerning leveraging hardware and software solutions for monitoring traffic, alerting, and simplifying the management of logs and operational data in Azure-based applications, as well as the use of the Spring Boot platform to create efficient and scalable applications, with a focus on reactive data management and extending software design goals.

Measurement. Similarly, several benefits can also be found for each tool in the obtained sub-topics, though all tools are perceived relatively negatively regarding measurement. Surprisingly, for nearly all tools, monitoring and measuring the performance of microservice architecture via application metrics, in general, is still positively perceived. Especially, OpenTelemetry also has benefits in the comprehensive metric collection in observability. Haystack and Sentry also have the benefits of measuring API performance.

Tracing. Regarding tracing, several tools have the benefits of distributed tracing (e.g., AppDynamics and Zipkin) and tracing for distributed systems (e.g., Jaeger). For these tools, using open-source tracing frameworks for monitoring and visualizing distributed microservices and enhancing observability and visualization are the benefits. On the other hand, real-time function tracing and real-time data visualization and monitoring are the benefits of Sentry and Splunk. Specifically, they enable leveraging free tools and alerts to trace function calls and state changes in real-time, optimizing query execution and exception handling, as well as interactive visualizations and tracking system performance in real-time enabling proactive incident response.

Usability. Regarding usability aspect, many tools are perceived more positively (shown in Table 9. For AppDynamics, the benefits include tool production (enhancing developer productivity through feature-rich development platforms) downtime reduction (minimizing downtime, comparing and track system health), and scalability and reliability. Several tools have benefits regarding security, including AppDynamics, Datadog, Dynatrace, Haystack, Jaeger, Sentry and Splunk. The reason for security topics appearing under the usability aspect is likely due to the fact they are commonly mentioned simultaneously. On the other hand, DataDog has the benefit of tool usability testing (evaluating the effectiveness of feature design and user experience in application development) while Dynatrace also has the benefit of performance management (ensuring and improving user experience through end-to-end testing and performance monitoring in a DevOps business environment). Similarly, New Relic, Sentry, and Splunk are also perceived positively regarding tool performance optimization (implementing new usability features in monitoring tools to support end-user teams and improve customer experience through effective design and browser alerts) and management as well as the balancing between performance and security (leveraging tools and best practices in DevOps and risk management to maintain application security without sacrificing performance or customer satisfaction).

4.3.2. RQ_4. Do Open Tracing Tools introduce any issues?

Similarly, the issues introduced by each of the open tracing tools are also shown in Table 9. We identify the issues from the tools that received more negative opinions than positive ones. Regarding *Development*, AppDynamics, Dynatrace, Jaeger, Haystack, New Relic, and Sentry are perceived more negatively than the average. For *Tracing* perspective, only Datadog, Dynatrace, New Relic, and Sentry are considered. Furthermore, all tools are perceived slightly negatively regarding *Measurement*. Similarly, by adopting another round of topic modeling, we can further compare each tool's issues with more details. The issues of each tool summarized by the practitioners' collective opinions are shown in the red texts of Tables 11 and 12. Apparently, as shown in the tables, it is likely there are still sub-topics that are perceived positively in those aspects.

Architecture. Datadog is poorly perceived regarding latency optimization in distributed tracing and resource management in high-usage environments. Jaeger has issues regarding evaluating integration strategies and vendor solutions and achieving high availability and scalability in distributed data services. New Relic and Zipkin also suffer from scalability-related issues regarding app architecture optimization and data-driven applications. For OpenTelemetry, the issue lies in navigating change in technology and deployment, that is, exploring the impact of release and deployment on observability, developer productivity, and key technology advancements while supporting user experience and tech leadership in API development. For Sentry, secure data management and analytics is the issue, that is, utilizing modern, open-source tools and APIs to develop secure, distributed data management systems with powerful analytics capabilities, ensuring efficient and safe handling of large-scale data across different platforms. Splunk's issues lie in big data and high-performance as well as robustness and adaptability.

Deployment & Integration . Regarding deployment and integration, surprisingly, all tools have issues in deployment, integration and monitoring in general. Specifically, Jaeger has an issue regarding deployment with Kubernetes for distributed tracing. New Relic has issue in agents for monitoring and alerting. OpenTelemetry falls short regarding zero-downtime deployment when Sentry is perceived negatively regarding seamless application deployment. Zipkin has issues in streamlined microservice deployment and also in testing and monitoring microservices in production.

Development. Regarding the development aspect, AppDynamics is perceived negatively on microservice development (navigating the complexities of microservice applications to improve team productivity, implement AI-powered operations, and maintain a scalable and efficient development environment), performance troubleshooting (monitoring memory usage, CPU, and server metrics to identify, diagnose, and resolve performance issues), service integration and communication (efficient service integration, coordinated communication, and proactive issue resolution in complex engineering environments), and reducing load times and costs. Datadog has issues regarding Kubernetes resource optimization and deployment process automation. Dynatrace and Zipkin have issues in proactive troubleshooting when Haystack and New Relic are in error handling. Jaeger and Sentry are perceived negatively regarding memory management. OpenTelemetry, Splunk and Zipkin also suffer from performance optimization.

Measurement. In terms of measurement, AppDynamics suffers from negative opinions on API and microservice metrics, specifically about monitoring and aggregating API requests, user interactions, and error logs to optimize microservice performance, ensure reliability, and manage resource usage. Datadog has issues regarding request-based event measurement, that is, analyzing response times, errors, and resource usage in Azure and

Prometheus to optimize endpoint performance and user experience. Dynatrace is perceived negatively mainly on API performance improvement with test metrics and log analysis and event rate limiting in distributed systems. Haystack, New Relic, and Sentry have issues in error management through measurement. Splunk also has issues in analyzing API and service performance, specifically, using real-time data, event logs, and query analysis to monitor and improve the response times and reliability of APIs and other service endpoints in complex systems. Zipkin has issues in monitoring and measuring application performance, analyzing logs and metrics at various levels to identify errors, and tracing events.

Tracing. Regarding the tracing aspect, error monitoring, and handling is a common issue for many tools, e.g., Datadog, Haystack, Jaeger, Sentry, and Zipkin. Meanwhile, both Datadog and Splunk have issues with incident notifications and automated alerts. AppDynamics has issues in performance and transaction tracing, i.e., capturing and analyzing transaction data to identify slow response times, execution bottlenecks, and other performance issues in business applications. Dynatrace and New Relic have issues in response time optimization and analysis.

Usability. Several tools have issues in app development and data pipeline management, including AppDynamics, Datadog, and Zipkin. The issues lie in the following aspects respectively: streamlining app development and release processes to create efficient, secure, and scalable applications using modern coding patterns and technologies; enhancing the usability of data pipelines with access controls, pattern compliance, and organization-wide sharing; complete audits and checks for crucial application deployment plans, with a focus on security and continuous production control. AppDynamics also has issues with flexible data configuration and microservice adoption towards system performance enhancement in general. Dynatrace, Haystack, Jaeger and Splunk are criticized for usability and user experience problems in general. New Relic has issues in simplifying error resolution and streamlining problem-solving. OpenTelemetry has issues in environment support and health, specifically about optimizing traceability in microservices for better team collaboration, ensuring security, and addressing problems through policy changes in the production pipeline, while managing costs and time. Sentry has issues with error resolution, specifically proactive error detection, and simplified solutions.

5. Discussion

The analysis of the results revealed interesting insights that let us distill a number of lessons and/or implications both for researchers and practitioners.

How to select a tool? Based on the comparative results, we cannot conclude that any of these tracing tool candidates is clearly better than the others. According to the results obtained in our study, and specifically for $\mathbf{RQ}_1$, different Open tracing tools provide different features that suit users and organizations with different preferences. For example, users who value complete control over their data will more likely choose a self-hosted solution; users who prefer a commercial solution with commercial support, will choose a tool like AppDynamics or Datadog. Most of the time, the availability of an agent or library for the programming language used within the team will be a important criteria to choose a specific tool.

Please note that questions about run-time behavior like throughput, resource usage, and performance impact are also of concern for engineers, but were not discussed in this paper. The actual run-time behavior of a given tool depends on the adopted programming language, the context in which the observed software is deployed, and on the level of granularity with which data is collected. We therefore suggest to *first* select tools based on the required functionality and *then* to validate in the concrete context, which tool satisfies more requirements.

We compared the 30 selected open tracing tools in details regarding licensing, programming languages, deployment, usage, the collected data, and interoperability. We hope that these results will ease the effort of the teams in selecting a tool according to their needs.

Which tools are best for what? The outcome of opinion mining from the gray literature ($\mathbf{RQ}_3$ and $\mathbf{RQ}_4$) shows the tools' quality reflected by the practitioners in the six key aspects for the 10 most popular tools. Therein, the results show that none of these tools are perceived positively in all aspects. Especially, the practitioners reflect more positive opinions on the *architecture* perspective of all these tools; however, more negative on the *measurement* perspective. To be emphasized, such results only indicate, for example, there are more negative opinions on tools' measurement than positive. It does not mean the deficit in these tools' measurement quality. It is depending on the teams' main interests and their criticality criteria that a selected tool can help them the most with its provided benefits. To be noted, it is highly likely there are certain amount of irrelevant data being taken into account due to the limitation in the data extraction strategy. The filtering procedure shall help to eliminate the influence yet the outcome can still contain distraction. Nonetheless, including more data sources, such as, forum posts, blogs, and tweets, and more facilitation from human expert shall enrich the opinion pool and further reduce the influence of irrelevant data.

Which tools are popular? We observed that the communities behind the analyzed tools differ significantly. In particular, Zipkin and Jaeger are the most cited tools in peer-reviewed publications, while Splunk, Haystack, Sentry, New Relic, Datadog, Zipkin, Jaeger, OpeneTelemetry, Dynatrace and AppDynamics are the most discussed tools in the selected online media ($\mathbf{RQ}_2$). Moreover, the community behind Splunk is the most responsive in term of questions answered in the online media channels investing in supporting the community by creating posts and tags. The other nine popular tools also have reasonably active supporting communities responding to technical issues. To be noted, there is a clear gap in terms of community discussion between the top 10 popular tools and the other tools. However, it certainly does not reflect their quality or usefulness to the customers. The difference in popularity may result from marketing strategies, investment in promotion, personal preferences, or simply the psychology of conformity.

To be noted, the different expectation levels of the tool users together with their various requirements can lead to the inevitable difference in their opinions. As the expectation of information system users is "a set of beliefs held by the targeted users of an information system associated with the eventual performance and with their performance using the system (Szajna and Scamell, 1993)". The initial expectation is formed towards a particular product or service (in our case, tracing tools) by the practitioners. After using the tool, they form the perception of the quality. When such perceived quality is assessed according to their expectation, the extent to which the expectation is confirmed is determined (Bhattacherjee, 2001). The satisfaction is formed based on the expectation and the corresponding confirmation. In this case, we assume the collective expectation of the practitioners towards each tracing tool is equal, because 1) the main feature of the tools can be seen as identical and 2) we collected statistically representative amount of data to even the deviation. Therefore, we assume the collective perceived sentiment of the practitioners towards the tools can be directly compared regardless of the expectation factor.

What is still missing? It is difficult to identify the benefits and issues in details regarding each identified aspect using the

topic modeling and sentiment analysis method. Surveys and interviews of the practitioners or domain expects shall facilitate the further investigation. On the other hand, it is also worthwhile to investigate the in-depth reasons for the high popularity of some tools, e.g., Datadog and AppDynamics. Furthermore, from the perspective of software evolution, it is also interesting to investigate the changes of these tools over time in terms of the provided features, user perceived benefits and issues, and their popularity.

6. Threats to validity

Our paper might suffer from threats related to the inaccuracy of the data extraction, a possible incomplete set of results due to limitation of the search terms, bibliographic sources and gray literature search engine, and possible subjectivity related to the definition and the application of the exclusion/inclusion criteria. In the this section, we discuss these threats and the strategies we adopted to mitigate them, based on the standard checklist for validity threats proposed in Wohlin et al. (2012).

Construct validity. Construct validities are concerned with issues that to what extent the object of study truly represents theory behind the study (Wohlin et al., 2012). The RQs and the classification schema adopted might suffer of this threat. To limit this threat, the authors reviewed independently and then discussed collaboratively RQs and the related classification schema.

To be noted, such the NLP-based approach of detecting the benefits and issues of open tracing tools ($\mathbf{RQ_3}$–$\mathbf{RQ_4}$) may fall shorts due to the potential data abundance. For some particular tools, e.g., Ocelot, SkyWalking and StageMonitor, the related text data is not sufficient compared to that of the other tools, which shall likely result in the lack of reliability in the related conclusion. On the other hand, the selected data sources, such as, Dzone and Medium, aim to introduce and promote emerging or prevailing technologies rather than to criticize them. As a result, the number of texts with negative sentiment is far lower than positive or neutral ones. It is recommended to include more data sources that covers the opinions from the forum end users who provide more unbiased comments and feedback. Furthermore, due to the limitation of LDA topic modeling on short texts, the performance of the approach can be further enhanced by adopting other techniques, such as the Biterm topic model (Cheng et al., 2014), which shall be included in the future work.

Internal Validity. The source selection approach adopted in this work is described in Section 2. In order enable the replicability of our work, we carefully identified and reported bibliographic sources adopted to identify the peer-review literature, search engines, adopted for the gray literature, search strings as well as inclusion and exclusion criteria. Possible issues in the selection process are related to the selection of search terms that could have lead to a non complete set of results. To mitigate this risk, we applied a broad search string. This was possible because of the novelty of the topic. To overcome the limitation of the search engines, we queried the academic literature from eight bibliographic sources, while we included the gray literature from Google, Medium Search, Twitter Search and Reddit Search. Additionally, we applied a snowballing process to include all the possible sources. The application of inclusion and exclusion can be affected by researchers' opinion and experience. To mitigate this threat, all the sources were evaluated by at least two authors independently.

Conclusion validity. Conclusion validity is related to the reliability of the conclusions drawn from the results (Wohlin et al., 2012). To ensure the reliability of our treatments, the terminology adopted in the schema has been reviewed by the authors to avoid ambiguities. All primary sources were reviewed by at least two authors to mitigate bias in data extraction and each disagreement was resolved by consensus, involving the third author.

External Validity. External validity is related to the generalizability of the results of our multivocal literature review. In this study, we map the literature on Open Tracing Tools, considering both the academic and the gray literature. However, we cannot claim to have screened all the possible literature, since some documents might have not been properly indexed, or possibly copyrighted or, even not freely available.

7. Related work

Among the literature, we identified only one study that – as a primary research goal – compares tools to understand their suitability in different contexts: Li et al. (2022) conduct an industrial survey regarding the different adoption strategies of distributed tracing tools. Covering ten different tools and ten different companies, the study finds that the companies' tracing and analysis pipelines are similar and that companies choose different tools based on different concerns and focuses caused by their company size.

More often, researchers compare tracing tools in their state-of-the-art section to point out a research gap and then propose their own approach. For example, Bento et al. (2021) propose using tracing data to extract service metrics, dependency graphs and work-flows with the objective of detecting anomalous services and operation patterns. Therein, the authors reflect on advantages and disadvantages of the tools Jaeger and Zipkin and point out their lack of automated analysis and processing functionality. Along the same lines, Song et al. (2019) propose ASTracer, a tracing tool for the Apache Hadoop[23] distributed file system. The authors point out the shortcomings of tracing tools like Zipkin, Jaeger and Htrace,[24] e.g., not considering the execution of different call trees or not being able to adapt its sampling rate during execution.

Some researchers discuss the use of a varying sampling rate when collecting data so that a tool can collect more data when a problem arises and fewer data otherwise. For example, Berg et al. (2021) propose Snicket, a distributed tracing system, in which database-style queries are used to express the analysis the developer wants to perform on the trace data. This query is then used to generate microservice extensions that intercept the needed data. The authors compare tracing tools such as Dapper (Sigelman et al., 2010), Jaeger, and Canopy (Kaldor et al., 2017), and point out that they may miss important unusual trace information with a uniform, up-front decided sampling rate. They also mention LightStep, which prioritizes latest unusual traces with dynamic tracing and sampling. Also Las-Casas et al. (2019) propose Sifter, a general-purpose distributed tracing framework that is able to adapt the sampling rate in case of anomalous and outlier executions. Sifter integrates with X-Trace (Fonseca et al., 2007), Jaeger and Zipkin to obtain tracing samples.

Another frequent type of paper is when researchers use tracing tools to obtain traces, which are then used in their research. For example, Gorige et al. (2020) propose a privacy risk detection framework based on distributed tracing, to identify privacy and security risks in microservices. They use Jaeger to collect the required data. Iurman et al. (2021) propose a unified solution combining in-band telemetry (an approach to collect data about the network state without affecting network performance) and Application Performance Management. In their

[23] https://hadoop.apache.org
[24] http://htrace.org

solution, Jaeger and other tracing tools can be used to collect application level information. Avritzer et al. (2021b) propose PPTAM, a set of tools for performance testing and performance-based application monitoring. They also extend their approach in Avritzer et al. (2021a) to detect performance anti-patterns.

In summary, within the identified literature, we observe that tracing tools are mentioned to (a) develop innovative tracing approaches or (b) to enhance existing tools, e.g., allowing an adaptive sampling rate. Moreover, often (c) researchers use the collected tracing data to obtain other research goals, e.g., identifying anti-patterns. Rarely, in fact we identified only one, a general-purpose comparison of various tools available on the market is the research goal. Such information can be found often in gray literature, such as blogs or online articles, e.g., Barker (2018), where Barker compares Zipkin, Jaeger, and Appdash.[25] Therefore, to obtain an overview over the available tools, a study has to (1) systematically collect the tools discussed in the literature and (2) consider both white and gray literature. This research gap is addressed in this paper.

8. Conclusion

In this work, we compared 30 Open Tracing Tools identified by adopting the Systematic Multivocal Literature Review process. For each tool, we investigated the measured features, the popularity both in peer-reviewed literature and online media, and derived benefits and issues. Specially, we adopted topic modeling and sentiment analysis for topic extraction and analysis with ChatGPT to support effective topic interpretation.

The achieved results provided interesting insights among the eleven tools investigated in this study. The deepest comparison we conducted did not allow us to clearly identify a "silver bullet" tool for any usage. Each tool has different implications under different conditions. Moreover, it is difficult to identify the benefits and issues in detail regarding each identified aspect using the topic modeling and sentiment analysis method.

CRediT authorship contribution statement

Andrea Janes: Study design, Data analysis, Writing. **Xiaozhou Li:** Data collection, Data analysis, Writing. **Valentina Lenarduzzi:** Supervision, Writing, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

Acknowledgments

The research presented in this article has been partially funded by : the "Software Rejuvenation" project funded by Ulla Tuominen Foundation by Academy of Finland project "MUFANO/349488".

25 https://github.com/sourcegraph/appdash

References

Apache SkyWalking, 2023. Agent. https://skywalking.apache.org/docs/#Agent. (Accessed on 14 January 2023).
Apache SkyWalking contributors, 2022. Github repository of Apache SkyWalking. https://github.com/apache/skywalking. (Accessed on 07 July 2022).
Apache Skywalking contributors, 2022a. Log collection and analysis. https://skywalking.apache.org/docs/main/latest/en/setup/backend/log-analyzer/. (Accessed on 07 July 2022).
Apache Skywalking contributors, 2022b. Meter receiver. https://skywalking.apache.org/docs/main/latest/en/setup/backend/backend-meter. (Accessed on 07 July 2022).
Apache Skywalking contributors, 2022c. OpenTelemetry receiver. https://skywalking.apache.org/docs/main/latest/en/setup/backend/opentelemetry-receiver. (Accessed on 07 July 2022).
Apache Skywalking contributors, 2022d. Overview. https://skywalking.apache.org/docs/main/v9.0.0/en/concepts-and-designs/overview. (Accessed on 07 July 2022).
Apache Skywalking contributors, 2022e. Overview . https://skywalking.apache.org/docs/main/latest/en/concepts-and-designs/overview/. (Accessed on 07 July 2022).
Apache Skywalking contributors, 2022f. Protocol documentation. https://skywalking.apache.org/docs/skywalking-banyandb/latest/api-reference. (Accessed on 07 July 2022).
Apache Skywalking contributors, 2022g. SkyWalking 9.x showcase. https://skywalking.apache.org/docs/skywalking-showcase/latest/readme/. (Accessed on 07 July 2022).
AppDash, 2023a. Github repository of AppDash. https://github.com/sourcegraph/appdash. (Accessed on 11 January 2023).
AppDash, 2023b. Language support - Github repository of AppDash. https://github.com/sourcegraph/appdash#language-support. (Accessed on 11 January 2023).
Appdynamics, 2022a. Github repository of . https://github.com/Appdynamics. (Accessed on 07 July 2022).
Appdynamics, 2022b. Agent installation by type. https://docs.appdynamics.com/appd/22.x/latest/en/application-monitoring/install-app-server-agents#id-.InstallAppServerAgentsv22.1-AgentInstallationbyType. (Accessed on 07 July 2022).
AppDynamics, 2022c. Architecture. https://docs.appdynamics.com/display/PRO14S/Architecture. (Accessed on 07 July 2022).
AppDynamics, 2022d. Getting started. https://docs.appdynamics.com/appd/22.x/latest/en/appdynamics-essentials/getting-started. (Accessed on 07 July 2022).
AppDynamics, 2022e. What is distributed tracing? https://www.appdynamics.com/topics/distributed-tracing. (Accessed on 07 July 2022).
AppDynamics, 2022f. Extensions and custom metrics. https://docs.appdynamics.com/appd/21.x/21.7/en/infrastructure-visibility/machine-agent/extensions-and-custom-metrics. (Accessed on 07 July 2022).
AppDynamics, 2022g. Log analytics. https://www.appdynamics.com/product/how-it-works/application-analytics/log-analytics. (Accessed on 07 July 2022).
AppDynamics, 2022h. AppDynamics APIs. https://docs.appdynamics.com/appd/21.x/21.7/en/extend-appdynamics/appdynamics-apis. (Accessed on 07 July 2022).
AppDynamics, 2022i. AppDynamics for OpenTelemetry. https://www.appdynamics.com/product/opentelemetry. (Accessed on 07 July 2022).
AppDynamics, 2022j. Get started with AppDynamics on-premise. https://docs.appdynamics.com/display/PRO39/Get+Started+with+AppDynamics+On-Premise. (Accessed on 07 July 2022).
Avritzer, A., Britto, R., Trubiani, C., Russo, B., Janes, A., Camilli, M., van Hoorn, A., Heinrich, R., Rapp, M., Henß, J., 2021a. A multivariate characterization and detection of software performance antipatterns. In: Proceedings of the ACM/SPEC International Conference on Performance Engineering. ICPE '21, Association for Computing Machinery, New York, NY, USA, pp. 61–72.
Avritzer, A., Camilli, M., Janes, A., Russo, B., Jahič, J., Hoorn, A.v., Britto, R., Trubiani, C., 2021b. PPTAMλ: What, where, and how of cross-domain scalability assessment. In: 2021 IEEE 18th International Conference on Software Architecture Companion. ICSA-C, pp. 62–69.
Barker, D., 2018. 3 open source distributed tracing tools. https://opensource.com/article/18/9/distributed-tracing-tools. (Accessed on 07 July 2022).
Basili, V.R., Caldiera, G., Rombach, H.D., 1994. The Goal Question Metric Approach. In: Encyclopedia of Software Engineering, John Wiley & Sons.
Basili, V.R., Trendowicz, A., Kowalczyk, M., Heidrich, J., Seaman, C., Münch, J., Rombach, D., 2014. Aligning organizations through measurement: The GQM+Strategies approach. The Fraunhofer IESE Series on Software and Systems Engineering, Springer International Publishing.
Bento, A., Correia, J., Filipe, R., Araujo, F., Cardoso, J., 2021. Automated analysis of distributed tracing: Challenges and research directions. J. Grid Comput. 19 (1), 1–15.
Berg, J., Ruffy, F., Nguyen, K., Yang, N., Kim, T., Sivaraman, A., Netravali, R., Narayana, S., 2021. Snicket: Query-driven distributed tracing. In: ACM Workshop on Hot Topics in Networks. pp. 206–212.

Bhattacherjee, A., 2001. Understanding information systems continuance: An expectation-confirmation model. MIS Q. 351–370.
Blei, D.M., Ng, A.Y., Jordan, M.I., 2003. Latent dirichlet allocation. J. Mach. Learn. Res. 3, 993–1022.
British Computer Society and Royal Academy of Engineering (Great Britain), 2004. The Challenges of Complex IT Projects: The Report of a Working Group from the Royal Academy of Engineering and the British Computer Society. Royal Academy of Engineering.
Brooks, 1987. No silver bullet essence and accidents of software engineering. Computer 20 (4), 10–19.
Cheng, X., Yan, X., Lan, Y., Guo, J., 2014. Btm: Topic modeling over short texts. IEEE Trans. Knowl. Data Eng. 26 (12), 2928–2941.
ContainIQ, 2023a. Automatic instrumentation and OpenTelemetry | Tutorial. https://www.containiq.com/post/auto-instrumentation-and-opentelemetry. (Accessed on 12 January 2023).
ContainIQ, 2023b. ContainIQ overview. https://docs.containiq.com/containiq-overview. (Accessed on 12 January 2023).
ContainIQ, 2023c. Github repository of ContainIQ. https://github.com/containiq/containiq-deployment. (Accessed on 11 January 2023).
ContainIQ, 2023d. Requirements. https://docs.containiq.com/requirements. (Accessed on 12 January 2023).
ContainIQ, 2023e. Using ContainIQ. https://docs.containiq.com/using-containiq#osSa5. (Accessed on 11 January 2023).
DataDog, 2022a. Github repository of DataDog. https://github.com/DataDog. (Accessed on 07 July 2022).
Datadog, 2022b. Set up datadog APM. https://docs.datadoghq.com/tracing/setup_overview. (Accessed on 07 July 2022).
Datadog, 2022c. Getting started. https://docs.datadoghq.com/getting_started/. (Accessed on 07 July 2022).
Datadog, 2022d. Distributed tracing overview. https://www.datadoghq.com/knowledge-center/distributed-tracing/. (Accessed on 07 July 2022).
Datadog, 2022e. Metrics. https://docs.datadoghq.com/metrics/. (Accessed on 07 July 2022).
Datadog, 2022f. Visualize key log metrics. https://www.datadoghq.com/dg/logs/log-metrics/. (Accessed on 07 July 2022).
Datadog, 2022g. API reference. https://docs.datadoghq.com/api/latest. (Accessed on 07 July 2022).
Datadog, 2022h. OpenTelemetry and OpenTracing. https://docs.datadoghq.com/tracing/setup_overview/open_standards/. (Accessed on 07 July 2022).
DataDog, 2023. Auto instrumentation. https://www.datadoghq.com/auto-instrumentation/. (Accessed on 12 January 2023).
Dynatrace, 2023a. Applications API. https://www.dynatrace.com/support/help/dynatrace-api/environment-api/topology-and-smartscape/applications-api. (Accessed on 12 January 2023).
Dynatrace, 2023b. Distributed traces. https://www.dynatrace.com/support/help/how-to-use-dynatrace/diagnostics/diagnostic-distributed-traces. (Accessed on 12 January 2023).
Dynatrace, 2023c. Github repository of Dynatrace. https://github.com/Dynatrace. (Accessed on 11 January 2023).
Dynatrace, 2023d. Hosted self-monitoring environment. https://www.dynatrace.com/support/help/setup-and-configuration/dynatrace-managed/self-monitoring/hosted-self-monitoring. (Accessed on 12 January 2023).
Dynatrace, 2023e. Instrument your service with OpenTelemetry. https://www.dynatrace.com/support/help/extend-dynatrace/opentelemetry/opentelemetry-traces/opentelemetry-ingest#instrument-service. (Accessed on 11 January 2023).
Dynatrace, 2023f. Log management and analytics. https://www.dynatrace.com/support/help/how-to-use-dynatrace/log-management-and-analytics. (Accessed on 12 January 2023).
Dynatrace, 2023g. Metrics. https://www.dynatrace.com/support/help/how-to-use-dynatrace/metrics. (Accessed on 12 January 2023).
Dynatrace, 2023h. Send data to dynatrace with OpenTelemetry. https://www.dynatrace.com/support/help/extend-dynatrace/opentelemetry. (Accessed on 12 January 2023).
Elastic, 2022a. Components and documentation. https://www.elastic.co/guide/en/apm/guide/current/apm-components.html. (Accessed on 07 July 2022).
Elastic, 2022b. Quick start. https://www.elastic.co/guide/en/apm/guide/current/apm-quick-start.html. (Accessed on 07 July 2022).
Elastic, 2022c. Distributed tracing. https://www.elastic.co/guide/en/apm/guide/current/apm-distributed-tracing.html. (Accessed on 07 July 2022).
Elastic, 2022d. Metrics. https://www.elastic.co/guide/en/apm/guide/current/data-model-metrics.html. (Accessed on 07 July 2022).
Elastic, 2022e. How to easily correlate logs and APM traces for better observability. https://www.elastic.co/blog/how-to-easily-correlate-logs-apm-traces-for-better-observability-elastic-stack. (Accessed on 07 July 2022).
Elastic, 2022f. API. https://www.elastic.co/guide/en/apm/guide/current/api.html. (Accessed on 07 July 2022).
Elastic, 2022g. OpenTelemetry integration. https://www.elastic.co/guide/en/apm/guide/current/open-telemetry.html. (Accessed on 07 July 2022).
Elastic APM-Server contributors, 2022. Github repository of the elastic APM-server. https://github.com/elastic/apm-server. (Accessed on 07 July 2022).
ElasticAPM, 2022. Instrument applications with APM. https://www.elastic.co/guide/en/observability/8.6/instrument-apps.html#instrument-apps. (Accessed on 07 July 2022).
Fenton, N.E., Pfleeger, S.L., 1998. Software Metrics: A Rigorous and Practical Approach, second ed. PWS Publishing Co., Usa.
Finkelstein, L., Leaning, M., 1984. A review of the fundamental concepts of measurement. Measurement 2 (1), 25–34.
Fonseca, R., Porter, G., Katz, R.H., Shenker, S., 2007. X-Trace: A pervasive network tracing framework. In: 4th USENIX Symposium on Networked Systems Design & Implementation. NSDI 07, USENIX Association, Cambridge, MA, pp. 271–284.
Garousi, V., Felderer, M., Mäntylä, M.V., 2019. Guidelines for including grey literature and conducting multivocal literature reviews in software engineering. Inf. Softw. Technol. 106, 101–121.
Gilbert, C., Hutto, E., 2014. Vader: A parsimonious rule-based model for sentiment analysis of social media text. In: Eighth International Conference on Weblogs and Social Media, Vol. 81. ICWSM-14, p. 82.
Gorige, D., Al-Masri, E., Kanzhelev, S., Fattah, H., 2020. Privacy-risk detection in microservices composition using distributed tracing. In: 2020 IEEE Eurasia Conference on IOT, Communication and Engineering. ECICE, Ieee, pp. 250–253.
Haystack, 2023a. Github repository of haystack. https://github.com/ExpediaDotCom/haystack. (Accessed on 11 January 2023).
Haystack, 2023b. Clients deployment. https://expediadotcom.github.io/haystack/docs/deployment/deployment_clients.html. (Accessed on 11 January 2023).
Haystack, 2023c. Traces. https://expediadotcom.github.io/haystack/docs/subsystems/subsystems_traces.html. (Accessed on 12 January 2023).
Haystack, 2023d. Metrics. https://expediadotcom.github.io/haystack/docs/about/clients.html#metrics. (Accessed on 12 January 2023).
Haystack, 2023e. Introduction. https://expediadotcom.github.io/haystack/docs/about/introduction.html#the-problem. (Accessed on 12 January 2023).
Haystack, 2023f. Haystack client. https://expediadotcom.github.io/haystack/docs/about/clients.html#haystack-agent. (Accessed on 12 January 2023).
Honeycomb.io, 2023a. Explore distributed trace data. https://docs.honeycomb.io/working-with-your-data/tracing/. (Accessed on 12 January 2023).
Honeycomb.io, 2023b. Github repository of Honeycomb.io. https://github.com/honeycombio. (Accessed on 11 January 2023).
Honeycomb.io, 2023c. Honeycomb via API. https://docs.honeycomb.io/api/. (Accessed on 12 January 2023).
Honeycomb.io, 2023d. Instrument your code. https://docs.honeycomb.io/getting-data-in/#instrument-your-code. (Accessed on 11 January 2023).
Honeycomb.io, 2023e. Metrics overview. https://docs.honeycomb.io/getting-data-in/metrics/. (Accessed on 12 January 2023).
Honeycomb.io, 2023f. OpenTelemetry. https://docs.honeycomb.io/getting-data-in/opentelemetry-overview/. (Accessed on 12 January 2023).
Honeycomb.io, 2023g. OpenTelemetry logs. https://docs.honeycomb.io/getting-data-in/logs/opentelemetry/. (Accessed on 12 January 2023).
Hypertrace, 2023a. Github repository of Hypertrace. https://github.com/hypertrace/hypertrace. (Accessed on 11 January 2023).
Hypertrace, 2023b. Instrumentation. https://docs.hypertrace.org/instrumentation/. (Accessed on 11 January 2023).
Hypertrace, 2023c. OpenTelemetry collector architecture. https://docs.hypertrace.org/architecture/data-collection/. (Accessed on 12 January 2023).
Hypertrace, 2023d. UI and platform overview - traces. https://docs.hypertrace.org/platform-ui/#traces. (Accessed on 12 January 2023).
IBM, 2022a. Getting started with Instana. https://www.ibm.com/docs/en/instana-observability/current?topic=references-getting-started-instana. (Accessed on 07 July 2022).
IBM, 2022b. OpenTelemetry. https://www.ibm.com/docs/en/obi/current?topic=apis-opentelemetry. (Accessed on 07 July 2022).
IBM, 2022c. Self-hosted Instana backend on docker (on-premises). https://www.ibm.com/docs/en/obi/current?topic=instana-self-hosted-backend-docker-premises. (Accessed on 07 July 2022).
Instana, 2022a. Agent REST API. https://instana.github.io/openapi/#section/Agent-REST-API. (Accessed on 07 July 2022).
Instana, 2022b. Application metrics. https://instana.github.io/openapi/#tag/Application-Metrics. (Accessed on 07 July 2022).
Instana, 2022c. Automatic instrumentation. https://www.ibm.com/docs/en/instana-observability/current?topic=references-tracing-in-instana#automatic-instrumentation. (Accessed on 07 July 2022).
Instana, 2022d. Distributed tracing for microservice applications. https://www.instana.com/automatic-distributed-tracing-and-analysis/. (Accessed on 07 July 2022).
Instana, 2022e. Github repository of Instana. https://github.com/instana. (Accessed on 07 July 2022).
Instana, 2022f. LogDNA. https://www.instana.com/supported-technologies/logdna/. (Accessed on 07 July 2022).

Instana, 2022g. Setting up and managing Instana. https://www.ibm.com/docs/en/obi/current?topic=setting-up-managing-instana. (Accessed on 07 July 2022).
Iurman, J., Brockners, F., Donnet, B., 2021. Towardss cross-layer telemetry. In: Applied Networking Research Workshop. pp. 15–21.
Jaeger, 2023. Client library features. https://www.jaegertracing.io/docs/1.41/client-features/. (Accessed on 11 January 2023).
Jaeger contributors, 2022a. APIs. https://www.jaegertracing.io/docs/1.36/apis/. (Accessed on 07 July 2022).
Jaeger contributors, 2022b. Architecture. https://www.jaegertracing.io/docs/1.36/architecture/. (Accessed on 07 July 2022).
Jaeger contributors, 2022c. Deprecating jaeger clients. https://www.jaegertracing.io/docs/1.35/client-libraries/#deprecating-jaeger-clients. (Accessed on 07 July 2022).
Jaeger contributors, 2022d. Getting started. https://www.jaegertracing.io/docs/1.36/getting-started/. (Accessed on 07 July 2022).
Jaeger contributors, 2022e. Github repository of Jaeger. https://github.com/jaegertracing/jaeger. (Accessed on 07 July 2022).
Jaeger contributors, 2022f. Introduction. https://www.jaegertracing.io/docs/1.36/. (Accessed on 07 July 2022).
Jaeger contributors, 2022g. Introduction. https://www.jaegertracing.io/docs/1.36/. (Accessed on 07 July 2022).
Jaeger contributors, 2022h. Troubleshooting. https://www.jaegertracing.io/docs/1.36/troubleshooting/. (Accessed on 07 July 2022).
Kaldor, J., Mace, J., Bejda, M., Gao, E., Kuropatwa, W., O'Neill, J., Ong, K.W., Schaller, B., Shan, P., Viscomi, B., Venkataraman, V., Veeraraghavan, K., Song, Y.J., 2017. Canopy: An end-to-end performance tracing and analysis system. In: Symposium on Operating Systems Principles. Sosp '17, Association for Computing Machinery, New York, NY, USA, pp. 34–50.
Kalman, R., 1960. On the general theory of control systems. IFAC Proc. Vol. 1 (1), 491–502, 1st International IFAC Congress on Automatic and Remote Control, Moscow, USSR, 1960.
Kamon, 2023a. Core APIs. https://kamon.io/docs/latest/core/#core-apis. (Accessed on 12 January 2023).
Kamon, 2023b. Github repository of Kamon. https://github.com/kamon-io/Kamon. (Accessed on 11 January 2023).
Kamon, 2023c. Logging trace ID and context information. https://kamon.io/docs/latest/guides/how-to/log-trace-id-and-context-info/#logging-trace-id-and-context-information. (Accessed on 12 January 2023).
Kamon, 2023d. Logit.io FOR opentelemetry. https://logit.io/platform/observability/opentelemetry-otel/. (Accessed on 12 January 2023).
Kamon, 2023e. Metrics. https://kamon.io/docs/latest/core/metrics/#metrics. (Accessed on 12 January 2023).
Kamon, 2023f. Monitoring Akka HTTP applications with Kamon. https://kamon.io/docs/latest/guides/installation/akka-http/. (Accessed on 11 January 2023).
Kamon, 2023g. Service overview. https://kamon.io/docs/latest/apm/services/service-details/#service-overview. (Accessed on 12 January 2023).
Kamon, 2023h. Trace details. https://kamon.io/docs/latest/apm/traces/trace-details/#trace-details. (Accessed on 12 January 2023).
Kitchenham, B., Charters, S., 2007. Guidelines for Performing Systematic Literature Reviews in Software Engineering. Technical Report Ebse 2007-001, Keele University and Durham University Joint Report.
Las-Casas, P., Papakerashvili, G., Anand, V., Mace, J., 2019. Sifter: Scalable sampling for distributed traces, without feature engineering. In: Symposium on Cloud Computing. pp. 312–324.
Li, B., Peng, X., Xiang, Q., Wang, H., Xie, T., Sun, J., Liu, X., 2022. Enjoy your observability: an industrial survey of microservice tracing and analysis. Empir. Softw. Eng. 27 (1), 1–28.
Lightstep, 2022a. Distributed tracing: A complete guide. https://lightstep.com/distributed-tracing. (Accessed on 07 July 2022).
Lightstep, 2022b. Find correlated areas of latency and errors. https://docs.lightstep.com/docs/find-correlated-areas-of-latency. (Accessed on 07 July 2022).
Lightstep, 2022c. Get started. https://docs.lightstep.com/get-started. (Accessed on 07 July 2022).
Lightstep, 2022d. Get started with lightstep observability. https://docs.lightstep.com/docs/welcome-to-lightstep. (Accessed on 07 July 2022).
Lightstep, 2022e. Github repository of lightstep. https://github.com/lightstep. (Accessed on 07 July 2022).
Lightstep, 2022f. Lightstep architecture explained. https://lightstep.com/blog/lightstep-xpm-architecture-explained. (Accessed on 07 July 2022).
Lightstep, 2023. Quick start: Tracing instrumentation. https://docs.lightstep.com/otel/quick-start-instrumentation. (Accessed on 14 January 2023).
Logit.io, 2023a. All sources - Languages & Libraries. https://logit.io/sources/search/. (Accessed on 11 January 2023).
Logit.io, 2023b. Github repository of Logit.io. https://github.com/logit-io. (Accessed on 11 January 2023).
Logit.io, 2023c. How do I use the HTTP/s logit ingestion API? https://help.logit.io/en/articles/3402534-how-do-i-use-the-http-s-logit-ingestion-api. (Accessed on 12 January 2023).
Logit.io, 2023d. Monitoring solutions from Logit.io. https://logit.io/solutions/monitoring/. (Accessed on 12 January 2023).
Logit.io, 2023e. The best solutions for handling metrics. https://logit.io/platform/metrics/. (Accessed on 12 January 2023).
Logit.io, 2023f. The best solutions for logging. https://logit.io/platform/logging/. (Accessed on 12 January 2023).
Lumigo, 2023a. AWS lambda auto-tracing. https://docs.lumigo.io/docs/auto-instrumentation. (Accessed on 11 January 2023).
Lumigo, 2023b. Github repository of Lumigo. https://github.com/lumigo-io. (Accessed on 11 January 2023).
Lumigo, 2023c. Logs. https://docs.lumigo.io/docs/logs. (Accessed on 12 January 2023).
Lumigo, 2023d. Metrics. https://docs.lumigo.io/docs/function-drilldown-view#metrics. (Accessed on 12 January 2023).
Lumigo, 2023e. OpenTelemetry support. https://docs.lumigo.io/docs/opentelemetry. (Accessed on 12 January 2023).
Lumigo, 2023f. Trace details. https://docs.lumigo.io/docs/trace-details. (Accessed on 12 January 2023).
Luo, Z., Xie, Q., Ananiadou, S., 2023. Chatgpt as a factual inconsistency evaluator for abstractive text summarization. arXiv preprint arXiv:2303.15621.
Mägi, I., 2020. Distributed tracing for dummies. https://plumbr.io/blog/monitoring/distributed-tracing-for-dummies. (Accessed on 07 July 2022).
Merriam-Webster.com Dictionary, 2022a. Telemeter. https://www.merriam-webster.com/dictionary/telemeter. (Accessed on 07 July 2022).
Merriam-Webster.com Dictionary, 2022b. Trace. https://www.merriam-webster.com/dictionary/trace. (Accessed on 07 July 2022).
Naushan, H., 2020. Topic modeling with latent Dirichlet allocation. https://towardssdatascience.com/topic-modeling-with-latent-dirichlet-allocation-e7ff75290f8. (Accessed on 29 December 2022).
Nigam, K., McCallum, A.K., Thrun, S., Mitchell, T., 2000. Text classification from labeled and unlabeled documents using EM. Mach. Learn. 39 (2–3), 103–134.
Novatec Consulting, 2022a. Breaking changes in 2.0.0. https://inspectit.github.io/inspectit-ocelot/docs/breaking-changes/Breaking%20Changes#breaking-changes-in-200. (Accessed on 07 July 2022).
Novatec Consulting, 2022b. Github repository of inspectIT Ocelot. https://github.com/inspectIT/inspectit-ocelot. (Accessed on 07 July 2022).
Novatec Consulting, 2022c. Hello world. https://inspectit.github.io/inspectit-ocelot/docs/doc1.(Accessed on 07 July 2022).
Novatec Consulting, 2022d. Log correlation. https://inspectit.github.io/inspectit-ocelot/docs/tracing/log-correlation. (Accessed on 07 July 2022).
Novatec Consulting, 2022e. Metrics recorders. https://inspectit.github.io/inspectit-ocelot/docs/metrics/metric-recorders. (Accessed on 07 July 2022).
Novatec Consulting, 2022f. OpenAPM. https://openapm.io. (Accessed on 07 July 2022).
Novatec Consulting, 2022g. Tracing. https://inspectit.github.io/inspectit-ocelot/docs/tracing/tracing. (Accessed on 07 July 2022).
Ocelot, 2023. Instrumentation. https://inspectit.github.io/inspectit-ocelot/docs/instrumentation/instrumentation. (Accessed on 14 January 2023).
OpenAI, 2023. Chatgpt. https://chat.openai.com/?model=gpt-4. (Accessed on 4 May 2023).
OpenCensus, 2023a. API documentation. https://opencensus.io/api/python/trace/api/index.html. (Accessed on 12 January 2023).
OpenCensus, 2023b. Github repository of OpenCensus. https://github.com/census-instrumentation. (Accessed on 11 January 2023).
OpenCensus, 2023c. Github repository of OpenCensus. https://github.com/census-instrumentation. (Accessed on 11 January 2023).
OpenCensus, 2023d. Stats/metrics. https://opencensus.io/stats/. (Accessed on 12 January 2023).
OpenCensus, 2023e. Tracing. https://opencensus.io/tracing/. (Accessed on 12 January 2023).
OpenTelemetry, 2023a. Instrumentation. https://opentelemetry.io/docs/instrumentation/. (Accessed on 11 January 2023).
OpenTelemetry, 2023b. Logs. https://opentelemetry.io/docs/concepts/signals/logs/. (Accessed on 12 January 2023).
OpenTelemetry, 2023c. Metrics. https://opentelemetry.io/docs/concepts/signals/metrics/. (Accessed on 12 January 2023).
OpenTelemetry, 2023d. Traces. https://opentelemetry.io/docs/concepts/signals/traces/. (Accessed on 12 January 2023).
Relic, N., 2023a. Github repository of New Relic. https://github.com/newrelic. (Accessed on 11 January 2023).
Relic, N., 2023b. Introduction to APM. https://docs.newrelic.com/docs/apm/new-relic-apm/getting-started/introduction-apm/. (Accessed on 11 January 2023).
Relic, N., 2023c. Introduction to new relic APIs. https://docs.newrelic.com/docs/apis/intro-apis/introduction-new-relic-apis/. (Accessed on 12 January 2023).
Relic, N., 2023d. Introduction to OpenTelemetry with new relic. https://docs.newrelic.com/docs/more-integrations/open-source-telemetry-integrations/opentelemetry/opentelemetry-introduction/. (Accessed on 12 January 2023).
Relic, N., 2023e. Introduction to the log API. https://docs.newrelic.com/docs/logs/log-api/introduction-log-api/. (Accessed on 12 January 2023).
Relic, N., 2023f. Introduction to the metric API. https://docs.newrelic.com/docs/data-apis/ingest-apis/metric-api/introduction-metric-api/. (Accessed on 12 January 2023).

Relic, N., 2023g. Introduction to the trace API. https://docs.newrelic.com/docs/distributed-tracing/trace-api/introduction-trace-api/. (Accessed on 12 January 2023).
Sentry, 2023a. API reference. https://docs.sentry.io/api/. (Accessed on 12 January 2023).
Sentry, 2023b. Github repository of Sentry. https://github.com/getsentry/sentry. (Accessed on 11 January 2023).
Sentry, 2023c. Logging. https://docs.sentry.io/product/relay/options#logging. (Accessed on 12 January 2023).
Sentry, 2023d. Metrics. https://docs.sentry.io/product/performance/transaction-summary/#metrics. (Accessed on 12 January 2023).
Sentry, 2023e. OpenTelemetry support. https://docs.sentry.io/platforms/python/guides/aws-lambda/performance/instrumentation/opentelemetry/. (Accessed on 12 January 2023).
Sentry, 2023f. Platforms. https://docs.sentry.io/platforms/. (Accessed on 11 January 2023).
Sentry, 2023g. Self-hosted support. https://docs.sentry.io/platforms/dotnet/migration/#self-hosted-support. (Accessed on 12 January 2023).
Sentry, 2023h. Tracing. https://docs.sentry.io/product/sentry-basics/tracing/. (Accessed on 12 January 2023).
Sigelman, B.H., Barroso, L.A., Burrows, M., Stephenson, P., Plakal, M., Beaver, D., Jaspan, S., Shanbhag, C., 2010. Dapper, a Large-Scale Distributed Systems Tracing Infrastructure. Technical Report, Google, Inc..
SigNoz, 2023a. Github repository of SigNoz. https://github.com/SigNoz/signoz. (Accessed on 11 January 2023).
SigNoz, 2023b. Instrument your application. https://signoz.io/docs/instrumentation/. (Accessed on 11 January 2023).
SigNoz, 2023c. Logs management overview. https://signoz.io/docs/userguide/logs/. (Accessed on 12 January 2023).
SigNoz, 2023d. OpenTelemetry operator usage. https://signoz.io/docs/tutorial/opentelemetry-operator-usage/. (Accessed on 12 January 2023).
SigNoz, 2023e. View traces. https://signoz.io/docs/userguide/traces/. (Accessed on 12 January 2023).
Site24x7, 2023a. API documentation. https://www.site24x7.com/help/api/. (Accessed on 12 January 2023).
Site24x7, 2023b. Application performance monitoring for Site24x7. https://www.site24x7.com/help/apm.html. (Accessed on 11 January 2023).
Site24x7, 2023c. Distributed tracing. https://www.site24x7.com/help/apm/distributed-tracing.html. (Accessed on 12 January 2023).
Site24x7, 2023d. Github repository of Site24x7. https://github.com/site24x7. (Accessed on 11 January 2023).
Site24x7, 2023e. Metric profile. https://www.site24x7.com/help/aws/metric-profile.html. (Accessed on 12 January 2023).
Site24x7, 2023f. Upload logs from logstash to Site24x7 AppLogs. https://www.site24x7.com/help/log-collectors/logstash.html. (Accessed on 12 January 2023).
Site24x7, 2023g. What is OpenTelemetry: A guide to understanding OpenTelemetry and the way forward. https://www.site24x7.com/blog/what-is-opentelemetry. (Accessed on 12 January 2023).
Song, Y., Li, Y., Wu, S., Yang, H., Li, W., 2019. ASTracer: An efficient tracing tool for HDFS with adaptive sampling. In: IFIP International Conference on Network and Parallel Computing. Springer, pp. 107–119.
SPDX Workgroup, 2022. SPDX license list. https://spdx.org/licenses/. (Accessed on 07 July 2022).
Splunk, 2023a. Github repository of Splunk. https://github.com/splunk. (Accessed on 11 January 2023).
Splunk, 2023b. Instrument back-end applications to send spans to Splunk APM. https://docs.splunk.com/Observability/gdi/get-data-in/application/application.html#nav-Instrument-back-end-services-and-applications. (Accessed on 11 January 2023).
Splunk, 2023c. Metrics metadata. https://dev.splunk.com/observability/reference/api/metrics_metadata/latest. (Accessed on 12 January 2023).
Splunk, 2023d. Reference documentation. https://dev.splunk.com/observability/reference. (Accessed on 12 January 2023).
Splunk, 2023e. Send APM traces. https://dev.splunk.com/observability/docs/apm/send_traces/. (Accessed on 12 January 2023).
Stagemonitor, 2023. Installation. https://github.com/stagemonitor/stagemonitor/wiki/Installation. (Accessed on 14 January 2023).
Stagemonitor contributors, 2022a. Github repository of stagemonitor. https://github.com/stagemonitor/stagemonitor. (Accessed on 07 July 2022).
Stagemonitor contributors, 2022b. Installation. https://github.com/stagemonitor/stagemonitor/wiki/Installation. (Accessed on 07 July 2022).
Stagemonitor contributors, 2022c. Logging dashboard. https://github.com/stagemonitor/stagemonitor/wiki/Logging-Dashboard. (Accessed on 07 July 2022).
Stagemonitor contributors, 2022d. Track your own metrics. https://github.com/stagemonitor/stagemonitor/wiki/Track-your-own-metrics. (Accessed on 07 July 2022).
Syed, S., Spruit, M., 2017. Full-text or abstract? Examining topic coherence scores using latent dirichlet allocation. In: 2017 IEEE International Conference on Data Science and Advanced Analytics. DSAA, Ieee, pp. 165–174.
Szajna, B., Scamell, R.W., 1993. The effects of information system user expectations on their performance and perceptions. Mis Q. 493–516.
Tanzu, 2023a. Github repository of Tanzu. https://github.com/vmware-tanzu. (Accessed on 11 January 2023).
Tanzu, 2023b. Instrumenting your app for tracing. https://docs.wavefront.com/tracing_instrumenting_frameworks.html. (Accessed on 11 January 2023).
Tempo, G., 2023a. Github repository of grafana tempo. https://github.com/grafana/tempo. (Accessed on 11 January 2023).
Tempo, G., 2023b. Instrumentation references. https://grafana.com/docs/tempo/latest/getting-started/instrumentation/. (Accessed on 11 January 2023).
Tempo, G., 2023c. OpenTelemetry. https://grafana.com/docs/?plcmt=footer. (Accessed on 12 January 2023).
Tempo, G., 2023d. OpenTelemetry. https://grafana.com/docs/opentelemetry/?plcmt=footer. (Accessed on 12 January 2023).
Tempo, G., 2023e. Tempo API. https://grafana.com/docs/tempo/latest/api_docs/. (Accessed on 12 January 2023).
The OpenTelemetry Authors, 2022. OpenTelemetry. https://opentelemetry.io/docs/. (Accessed on 07 July 2022).
UpTrace, 2023a. Github repository of UpTrace. https://github.com/uptrace. (Accessed on 11 January 2023).
UpTrace, 2023b. OpenTelemetry. https://uptrace.dev/opentelemetry/. (Accessed on 12 January 2023).
UpTrace, 2023c. OpenTelemetry distributed tracing. https://uptrace.dev/opentelemetry/distributed-tracing.html. (Accessed on 12 January 2023).
UpTrace, 2023d. OpenTelemetry instrumentations. https://uptrace.dev/opentelemetry/instrumentations/. (Accessed on 11 January 2023).
UpTrace, 2023e. OpenTelemetry logs. https://uptrace.dev/opentelemetry/logs.html. (Accessed on 12 January 2023).
UpTrace, 2023f. OpenTelemetry metrics. https://uptrace.dev/opentelemetry/metrics.html. (Accessed on 12 January 2023).
VictoriaMetrics, 2023a. API docs. https://docs.victoriametrics.com/operator/api.html. (Accessed on 12 January 2023).
VictoriaMetrics, 2023b. CodeQL support for VictoriaMetrics. https://github.com/VictoriaMetrics/VictoriaMetrics/blob/b4e6460d2f92f712c3338e4d065e521592d862da/.github/workflows/codeql-analysis.yml. (Accessed on 12 January 2023).
VictoriaMetrics, 2023c. Github repository of VictoriaMetrics. https://github.com/VictoriaMetrics/VictoriaMetrics. (Accessed on 11 January 2023).
VictoriaMetrics, 2023d. Metrics explorer. https://docs.victoriametrics.com/Single-server-VictoriaMetrics.html#metrics-explorer. (Accessed on 12 January 2023).
VictoriaMetrics, 2023e. OTLP support in VictoriaMetrics? #2424. https://github.com/VictoriaMetrics/VictoriaMetrics/issues/2424. (Accessed on 12 January 2023).
VictoriaMetrics, 2023f. Query tracing. https://docs.victoriametrics.com/#query-tracing. (Accessed on 12 January 2023).
VMware, 2022a. App metrics for vmware Tanzu. https://docs.vmware.com/en/App-Metrics-for-VMware-Tanzu/2.1/app-metrics/GUID-index.html. (Accessed on 07 July 2022).
VMware, 2022b. Enterprise grade on day 1. https://tanzu.vmware.com/content/vmware-tanzu-observability-features/enterprise-grade-on-day-1. (Accessed on 07 July 2022).
VMware, 2022c. Microservices observability with distributed tracing. https://tanzu.vmware.com/content/vmware-tanzu-observability-features/microservices-observability-with-distributed-tracing. (Accessed on 07 July 2022).
Wavefront, 2022a. Getting started pages. https://docs.wavefront.com/label_getting%20started.html. (Accessed on 07 July 2022).
Wavefront, 2022b. Log data integration. https://docs.wavefront.com/log.html. (Accessed on 07 July 2022).
Wavefront, 2022c. Send OpenTelemetry data. https://docs.wavefront.com/opentelemetry_tracing.html. (Accessed on 07 July 2022).
Wavefront, 2022d. Wavefront REST API. https://docs.wavefront.com/wavefront_api.html. (Accessed on 07 July 2022).
Wohlin, C., 2014. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In: International Conference on Evaluation and Assessment in Software Engineering. Ease '14, Association for Computing Machinery, New York, NY, USA, pp. 1–10.
Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., 2012. Experimentation in Software Engineering. pp. 1–236.
Yang, X., Li, Y., Zhang, X., Chen, H., Cheng, W., 2023. Exploring the limits of chatgpt for query or aspect-based text summarization. arXiv preprint arXiv:2302.08081.
Zipkin, 2023. Tracers and instrumentation. https://zipkin.io/pages/tracers_instrumentation.html. (Accessed on 14 January 2023).
Zipkin contributors, 2022a. Architecture. https://zipkin.io/pages/architecture.html. (Accessed on 07 July 2022).
Zipkin contributors, 2022b. Github repository of zipkin. https://github.com/openzipkin/zipkin. (Accessed on 07 July 2022).

Zipkin contributors, 2022c. Opentelemetry zipkin exporter. https://opentelemetry.github.io/opentelemetry-python/exporter/zipkin/zipkin.html. (Accessed on 07 July 2022).
Zipkin contributors, 2022d. Quickstart. https://zipkin.io/pages/quickstart.html. (Accessed on 07 July 2022).
Zipkin contributors, 2022e. Server extensions and choices. https://zipkin.io/pages/extensions_choices.html. (Accessed on 07 July 2022).
Zipkin contributors, 2022f. Tracers and instrumentation. https://zipkin.io/pages/tracers_instrumentation.html. (Accessed on 07 July 2022).

Andrea Janes is senior lecturer at FHV Vorarlberg University of Applied Sciences (Austria). His research activity is related to the area of software maintenance and development. In particular, his research involves the identification of cost-efficient software production techniques, quality assurance methodologies, as well as the application of foundational aspects of software engineering methods such as testing and software process improvement. He received the master's in computer science from the Technical University of Vienna, Austria and the doctorate in computer science (with distinction) from the University of Klagenfurt (Austria). He was a visiting researcher at the Research Center Hagenberg (Austria) and the Tampere University (Finland). He was an assistant professor at the Free University of BolzanoBozen (Italy). He served as a program committee member of various international conferences and as a reviewer for various international journals (e.g., TSE, EMSE, JSS, and IST) in the field of software engineering. He has been Doctoral Symposium co-chair of PROFES 2022, short papers and poster co-chair of EASE 2023, program co-chair of PROFES 2023, Journal First and Special Issue chair of QUATIC 2023, and industrial papers co-chair of ESOCC 2023 and SANER 2024. He organized several workshops and events for practitioners focused on the application of research in industry. Since 2017, he is also involved in technology transfer within Smart Data Factory, a group within the NOI technology park with the goal of technology transfer within the local industry.

Xiaozhou Li is a postdoctoral researcher in Empirical Software Engineering in Software, Systems and Services (M3S) Research Group at Faculty of Information Technology and Electrical Engineering (ITEE), University of Oulu, Finland. Li received his Ph.D. in computer science from Tampere University. His research interests include microservice degradation, microservice organization network, open source software quality, software maintenance and evolution, user review opinion mining, and computational game studies.

Valentina Lenarduzzi is an Assistant Professor (tenure track) in Empirical Software Engineering in Software, Systems and Services (M3S) Research Group at Faculty of Information Technology and Electrical Engineering (ITEE), University of Oulu (Finland). Her research activities are related to contemporary software development practices and methodologies, including data analysis in software engineering, software quality, software maintenance and evolution, focusing on Technical Debt as well as code and architectural smells. She got the Ph.D. in Computer Science in 2015 and was a postdoctoral researcher at the Free University of Bozen-Bolzano, (Italy), at the Tampere University (Finland), and LUT University (Finland). Moreover, she was visiting researcher at the University of Kaiserslautern (TUK) and the Fraunhofer Institute for Experimental Software Engineering IESE (Germany). She is Finnish consortium leader in the COST Action EUGAIN European Network For Gender Balance in Informatics. She served as a program committee member of various international conferences (e.g., ICSE, ICSME, ESEM), and for various international journals (e.g., TSE, EMSE, JSS, IST) in the field of software engineering. She has been program co-chair of OSS 2021, TechDebt 2022, SEAA2023, PROFES 2023, and QUATIC2023. Moreover, she served in different organization roles (i.e. short papers, emerging results, and publicity chair) in several conferences such as ESEM 2021, SANER 2022, PROFES 2022, EASE2023, ESEM2023, ESOCC 2023, and SANER 2024. Dr. Lenarduzzi is recognized by the Journal of Systems and Software (JSS) as one of the most active SE researcher in top-quality journals in the period 2013 to 2020.