Open Tracing Tools: Overview and Critical Comparison

Andrea Janes\textsuperscript{a,b}, Xiaozhou Li\textsuperscript{c,d}, Valentina Lenarduzzi\textsuperscript{d}

\textsuperscript{a}FHV Vorarlberg University of Applied Sciences, Austria
\textsuperscript{b}Free University of Bozen-Bolzano, Italy
\textsuperscript{c}Tampere University, Finland
\textsuperscript{d}University of Oulu, Finland

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.

Method. 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.

Keywords: Open Tracing Tool, Telemetry, Multivocal Literature Review, ChatGPT

1. Introduction

In software engineering, the outcome of the engineering process is invisible \cite{1, 2}. As a consequence, it is difficult to understand progress and to reason about the produced output \cite{3}. 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 \cite{4}. 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 \cite{5, 2}.

Medicine distinguishes between the measuring instruments that need to \textit{look inside} a patient and those that do not. For example, blood pressure can be measured directly via an arterial catheter (called \textit{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 \textit{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 \textit{tracing} and \textit{telemetry}. Tracing, as the word \textit{trace}, means “a mark or line left by something that has passed” \cite{6}, 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—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 \textit{tele} (remote) and the word \textit{metron} (measure) and means to measure something and transmitting the results to a distant station \cite{7}. 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” [8] 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 [9]. 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 [10] to support the topic interpretation.

The key results of the study show 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 a 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 package 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) [9]. The MLR process [9] 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 Figure 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 [11]. 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 [12], 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 [13], we devise the below research questions operationalizing “utility” into four dimensions: ability to measure, popularity, benefits, issues. Consequently, we formulated the following research questions:

RQ1: Which distinctive features do the tools possess?
RQ2: How popular are the identified tools?
RQ3: Which benefits do the identified tools claim to achieve?
RQ4: Which issues do the identified tools introduce?

With RQ1, 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 RQ2, we investigated the popularity of each tool in terms of adoption both in academia and among the developers communities considering when the first version was released on the market. Once characterized each tool, we proceeded with a finer-grained investigation considering benefits (RQ3) and issues (RQ4) 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 Library¹, IEEExplore Digital Library², Science Direct³, Scopus⁴, Google Scholar⁵, CiteseerX⁶, Inspect⁷, and Springer link⁸. 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 Search⁹, Twitter¹⁰, Reddit¹¹ and Medium¹². 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 [11]. 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 [14], 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 observability. 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

---

¹https://dl.acm.org
²https://ieeexplore.ieee.org
³https://www.sciencedirect.com
⁴https://www.scopus.com
⁵https://scholar.google.com
⁶https://citeseer.ist.psu.edu
⁷https://iee.org/Publish/INSPEC/
⁸https://link.springer.com/
⁹https://www.google.com/
¹⁰https://twitter.com/
¹¹https://www.reddit.com/
¹²https://medium.com
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 [9] 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.

| 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/ |
| Sirkit    | https://github.com/sirkitio/sirkit/ |
| Signoz   | https://www.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. Tool analysis

The following steps—based on the research goal and derived questions described in Sect. 1—describe the study context, the data extraction and analysis, and describe its verifiability and replicability following the approach suggested by [15].

3.1. Study context

We considered the eleven open tracing tools retrieved according the process described in Section 2. Table 2 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 (RQ1)**: 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 [16] 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 [17] 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 [17], and if self-hosting is possible.

**Tool popularity (RQ2)**: 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 (**"open tracing" OR "open tracing"**).

- **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 (RQ3 and RQ4)**: 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 covering 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\(^\text{13}\) 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\(^\text{14}\). 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.

### 3.3. Data Analysis

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

**Distinctive features (RQ1)**. 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.

---

\(^{13}\)https://api.stackexchange.com/docs/advanced-search

\(^{14}\)https://www.crummy.com/software/BeautifulSoup/

**Tool popularity (RQ2)**. We use the statistics from both scientific sources and media sources to interpret and compare the popularity of the selected tools.

**Benefits and issues (RQ3 and RQ4)**. Figure 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 RQ2), 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 [18]. 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 by 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, screeninig stop-words, and eliminating extra white spaces, and lemmatization.

- **Step 2: Filtering**: This step is to filter out the non informative sentences with trained text classifier. By doing so, we shall identify the sentences that contain useful information and screen out those not relevant. Aiming to answer RQ3 and RQ4, the informative sentences shall contain the 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 about 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 topic. By doing so, we shall know regarding each tool which topics are discussed in the social media and how frequent for 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 the detailed reflection on the percentage of positivity and negativity for each topic for each tool.

4. Results

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

4.1. Distinctive features (RQ1)

To answer RQ1, 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 [19]. 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 first 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 [16] (see Figure 3):

![Figure 2: Approach to detect benefits and issues from analyzing social media texts (RQ3 and RQ4)](https://figshare.com/s/8aa40eea5d50ed27d347)
| Tool                  | License            | Programming language      | Repo. | Pricing                                                                 | Created  |
|---------------------|--------------------|--------------------------|-------|-------------------------------------------------------------------------|----------|
| Appdash             | MIT [20]           | Go*, Python*, Ruby*      [20, 21] | [20]  | Free                                                                    | 2014     |
| AppDynamics         | Proprietary, Apache-2.0, GPL-3.0, MIT [22] | Java*, Shell*, NET*, Python*, JavaScript*, Go*, C*/++, PHP*, NodeJS* [22, 23] | [22]  | $0.60/0.60/0.60/0.60 per month, per CPU Core; $0.60 per month, per 1000 tokens; Quote | 2008     |
| Containiq           | Proprietary [24]   | C*/++, Go*, Rust*, Python*, Ruby*, NodeJS* [25] | [24]  | $20 per node, per month OR $3.50 per GB of log data ingested; Quote   | 2021     |
| Datadog             | Proprietary, Apache-2.0, BSD-3-Clause, GPL-2.0, MIT, MPL-2.0 [20] | Go*, Python*, Ruby*, JavaScript*, Node.js*, Java*, .NET* [26, 27] | [26]  | Free; $75/23 Per host, per month                                         | 2010     |
| Dynatrace           | Apache-2.0 [28]    | C*/++, .NET*, Erlang*, Go*, Java*, NodeJS*, Python*, Ruby*, Rust* [28, 29] | [28]  | $237/415/45 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 [30] | Go*, Python*, iOS*, Java*, NodeJS*, PHP*, Ruby*, Gherkin [30, 31] | [30]  | Free; $95/109/125/175 per month                                          | 2012     |
| Grafeas Tempo       | AGPL-3.0-only [32] | Java*, Go*, .NET*, Python*, NodeJS* [32, 33] | [32]  | Free                                                                    | 2020     |
| Haystack            | Apache-2.0 [34]    | Java*, NodeJS*, Python*, Go*, HCL, Shell, Smarty [34, 35] | [34]  | Free                                                                    | 2017     |
| Hypertrace          | Traceable Community License Agreement (1.0)[36] | Java*, Go*, Python*, NodeJS*, .NET* [36, 37] | [36]  | Free                                                                    | 2020     |
| Honeycomb.io        | Apache-2.0, MIT [38] | Go*, Java*, .NET*, NodeJS*, Python*, Ruby*, JavaScript, Python [38, 39] | [38]  | Free; Quote                                                             | 2016     |
| Instana             | Proprietary, Apache-2.0, GPL-2.0, MIT [40] | Shell, JavaScript, Go, Java*, Python*, .NET*, Clojure*, Kotlin*, Python*, PHP*, Scala*, NodeJS*, Ruby* [40, 41] | [40]  | $75/93.80 per host, per month                                           | 2015     |
| Jaeger              | Apache-2.0 [42]    | Go*, Java*, NodeJS*, Python*, .NET* [42, 43] | [42]  | Free                                                                    | 2016     |
| Kamon               | Apache-2.0 [44]    | Java*, Scala* [44, 45] | [44]  | Free; $200/250 per month; Quote                                        | 2017     |
| LightStep           | Proprietary, Apache-2.0, BSD-2-Clause, BSD-3-Clause, CC-BY-SA-4.0, MIT [46] | Go*, JavaScript*, Python*, Java*, HCL*.NET*, NodeJS* [46, 47] | [46]  | Free; $100 per active service per month; Quote                          | 2015     |
| Logit.io            | MIT [48]           | .NET*, Go*, NodeJS*, Python*, Ruby*, JavaScript, Shell [48, 49] | [48]  | $0.64 per GB, $5 per million spans per month; $2.80 per 1000 DPM      | 2013     |
| Lumugo              | Apache-2.0 [50]    | Python*, NodeJS*, Java, Go [50, 51] | [50]  | $100/290 per month; Quote                                              | 2016     |
| New Relic           | Apache-2.0 [52]    | C*, Go*, Java*, .NET*, NodeJS*, PHP*, Python*, Ruby*, JavaScript, Shell [52, 53] | [52]  | $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 [54]    | Java*, JavaScript* [54, 55] | [54]  | Free                                                                    | 2016     |
| OpenCensus          | Apache-2.0 [56]    | Python*, NodeJS*, Go*, C*, .NET*, Erlang*, Java* [56, 57] | [56]  | Free                                                                    | 2017     |
| OpenTelemetry       | Apache-2.0 [11]    | C*/++, .NET*, Erlang*, Go*, Java*, JavaScript*, PHP*, Python*, Ruby*, Rust*, Swift* [17, 58] | [11]  | Free                                                                    | 2019     |
| Sentry              | BSL-1.1 [59]       | .NET*, JavaScript*, NodeJS*, PHP*, Rust*, Java*, Go* [59, 60] | [59]  | $0/267.80 per month; Quote                                             | 2012     |
| Splunk              | Apache-2.0 [61]    | Python*, Java*, NodeJS*, .NET*, Go*, Ruby*, PHP* [61, 62] | [61]  | $15/50/250/500 per month; Quote                                        | 2003     |
| Signoz              | MIT [63]           | Java*, Python*, JavaScript*, Go*, PHP*, .NET*, Ruby*, Elixir*, Rust* [63, 64] | [63]  | Free; $200 per month; Quote                                            | 2020     |
| Site2Xs7            | BSD-2-Clause, MIT [65] | Java*, .NET*, Ruby*, PHP*, NodeJS*, Python* [65, 66] | [65]  | Free; $397/390/325 per month; Quote                                    | 2006     |
| SkyWalking          | Apache-2.0 [67]    | Java*, Python*, NodeJS*, Lua*, JavaScript*, Rust*, PHP* [67, 68] | [67]  | Free                                                                    | 2015     |
| StageMonitor        | Apache-2.0 [69]    | Java*, HTML*, JavaScript* [69, 70] | [69]  | Free                                                                    | 2013     |
| Tanizu              | Apache-2.0 [71]    | Java*, .NET*, Go*, .NET*, Python* [71, 72] | [71]  | Free                                                                    | 2019     |
| Uptrace             | BSD-2-Clause, Apache-2.0 [73] | Go*, NodeJS*, .NET*, Ruby*, Python* [73, 74] | [73]  | $40/70/60/50/40/30/25 per month for GB for 50/1000/2500/4250/6000/8000GB with 5$/+100$/+200$/+300$/+400$/+500$/budget | 2021     |
| Victoriametrics     | Apache-2.0 [75]    | Go*, JavaScript* [75, 76] | [75]  | Free; Quote                                                             | 2018     |
| Zipkin              | Apache-2.0 [77]    | Go*, Go*, Java*, JavaScript*, Ruby*, Scala*, PHP* [77, 78] | [77]  | Free                                                                    | 2012     |
• **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** components.

Figure 3 depicts a generalized deployment scenario of the various components using the terminology of the OpenAPM initiative. The architecture depicted in Figure 3 also corresponds to the suggested architecture by the OpenTelemetry project [17] (see below).

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 Figure 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 Figure 3). Therefore, using tracing tools always involves the following steps:

1. **Backend installation**: If the tool is installed on premise (as e.g., ElasticAPM [90], SkyWalking [91], or Zipkin [92]): 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 [93]), selecting a data collection site to respect privacy regulations (as for, e.g., Datadog [94]) 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 [93, 94, 95, 96, 90, 97, 91, 98, 99]. 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\(^\text{16}\).

4. **Data processing**: once the data collection is in place, the various tools (see Figure 3) allow three different types of 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 \([17]\) **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 \([17]\), 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 \([17]\).” 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 Figure 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 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 Figure 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 (Figure 4). However, when checking the obtained data volume for each tool, we found only 10 tools have more than 600 data points [18]. 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 Figure 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 (RQ3 and RQ4)

According to the social media content distribution described in Figure 4, we proceeded to answer to RQ3 and RQ4 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

| Tool                  | Traces | Metrics | Logs |
|-----------------------|--------|---------|------|
| Appdash               | ✗      | ✗       | ✗    |
| AppDynamics           | ✗      | ✗       | ✗    |
| Containing            | ✗      | ✗       | ✗    |
| Datadog               | ✗      | ✗       | ✗    |
| Dynatrace             | ✗      | ✗       | ✗    |
| ElasticAPM            | ✗      | ✗       | ✗    |
| Grafana tempo         | ✗      | ✗       | ✗    |
| Honeycomb.io          | ✗      | ✗       | ✗    |
| Hypertrace            | ✗      | ✗       | ✗    |
| Haystack              | ✗      | ✗       | ✗    |
| Instana               | ✗      | ✗       | ✗    |
| Jaeger                | ✗      | ✗       | ✗    |
| Kamon                 | ✗      | ✗       | ✗    |
| LightStep             | ✗      | ✗       | ✗    |
| LogIt.io              | ✗      | ✗       | ✗    |
| Lumigo                | ✗      | ✗       | ✗    |
| New Relic             | ✗      | ✗       | ✗    |
| Ocelot                | ✗      | ✗       | ✗    |
| OpenTelemetry         | ✗      | ✗       | ✗    |
| Sentry                | ✗      | ✗       | ✗    |
| Splunk                | ✗      | ✗       | ✗    |
| SkyWalking            | ✗      | ✗       | ✗    |
| StackTrace            | ✗      | ✗       | ✗    |
| Signoz                | ✗      | ✗       | ✗    |
| StageMonitor          | ✗      | ✗       | ✗    |
| Tanzu                 | ✗      | ✗       | ✗    |
| Uptrace               | ✗      | ✗       | ✗    |
| Victoriametrics       | ✗      | ✗       | ✗    |
| Zipkin                | ✗      | ✗       | ✗    |

Table 4: Metrics collected by the identified tools (RQ2)

| Tool                  | API      | OpenTelemetry support | Self-hosting |
|-----------------------|----------|-----------------------|--------------|
| Appdash               | ✗        | ✗                     | ✗            |
| AppDynamics           | ✗        | ✗                     | ✗            |
| ContainIQ             | ✗        | ✗                     | ✗            |
| Datadog               | ✗        | ✗                     | ✗            |
| Dynatrace             | ✗        | ✗                     | ✗            |
| ElasticAPM            | ✗        | ✗                     | ✗            |
| Grafana tempo         | ✗        | ✗                     | ✗            |
| Honeycomb.io          | ✗        | ✗                     | ✗            |
| Hypertrace            | ✗        | ✗                     | ✗            |
| Haystack              | ✗        | ✗                     | ✗            |
| Instana               | ✗        | ✗                     | ✗            |
| Jaeger                | ✗        | ✗                     | ✗            |
| Kamon                 | ✗        | ✗                     | ✗            |
| LightStep             | ✗        | ✗                     | ✗            |
| Logit.io              | ✗        | ✗                     | ✗            |
| Lumigo                | ✗        | ✗                     | ✗            |
| New Relic             | ✗        | ✗                     | ✗            |
| Ocelot                | ✗        | ✗                     | ✗            |
| OpenTelemetry         | ✗        | ✗                     | ✗            |
| Sentry                | ✗        | ✗                     | ✗            |
| Splunk                | ✗        | ✗                     | ✗            |
| SkyWalking            | ✗        | ✗                     | ✗            |
| StackTrace            | ✗        | ✗                     | ✗            |
| Signoz                | ✗        | ✗                     | ✗            |
| StageMonitor          | ✗        | ✗                     | ✗            |
| Tanzu                 | ✗        | ✗                     | ✗            |
| Uptrace               | ✗        | ✗                     | ✗            |
| Victoriametrics       | ✗        | ✗                     | ✗            |
| Zipkin                | ✗        | ✗                     | ✗            |

Table 5: Features about interoperability (RQ1)

1. OpenTelemetry data can be exported to Zipkin [220].
2. Metrics can be reported to the OpenTelemetry receiver or imported using OpenTelemetry exporter; Traces and logs are not supported.
3. Site24x7’s support for OpenTelemetry is currently in development.
4. OpenTelemetry support is requested within an issue.
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)\textsuperscript{17} 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\textsuperscript{18} to filter the non-English texts and obtained 37,685 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\textsuperscript{19}. 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 [221]. 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). Figure 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 [222], a number of preprocessing steps are required, which include: removing punctuation, removing extra space, restoring the word to its

\textsuperscript{17}http://www.nltk.org/

\textsuperscript{18}https://pypi.org/project/langdetect/

\textsuperscript{19}https://pypi.org/project/html2text/
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 [223]. 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 Figure 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 mapped each informative sentence to one of the topics to which it was most likely related. Shown in Figure 8, the numbers of topic-related sentences from the articles on each tool are summarized. Compared with Figure 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\footnote{http://Haystacksearch.org/}. All such texts should have been classified as "non-informative". Displayed in Figure 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 [224], 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 Figure 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.

**Table 8: Topic sentiment average percentage for each tool (RQ3 and RQ4)**

| Tool            | Positive | Neutral | Negative |
|-----------------|----------|---------|----------|
| AppDynamics     | 47.4%    | 36.6%   | 16.0%    |
| Datadog         | 43.3%    | 42.2%   | 14.6%    |
| Dynatrace       | 45.9%    | 29.9%   | 24.2%    |
| Haystack        | 38.0%    | 40.6%   | 21.4%    |
| Jaeger          | 40.8%    | 40.6%   | 12.6%    |
| New Relic       | 32.3%    | 47.4%   | 20.1%    |
| OpenTelemetry   | 41.9%    | 40.2%   | 11.9%    |
| Sentry          | 30.8%    | 36.6%   | 32.6%    |
| Splunk          | 44.7%    | 40.1%   | 15.2%    |
| Zipkin          | 41.1%    | 45.8%   | 13.1%    |

- 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.

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 RQ3 and RQ4.

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 (RQ1): 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 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.

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                   | 3           | 2       | 2         | 4        | 3      | 3         | 6             | 6      | 6      | 3      |
| Measurement                   | 3           | 3       | 6         | 4        | 4      | 2         | 6             | 2      | 3      | 3      |
| Tracing                       | 3           | 9       | 5         | 4        | 4      | 3         | 3             | 4      | 2      |        |
| Usability                     | 9           | 4       | 3         | 4        | 3      | 4         | 8             | 3      | 6      | 6      |

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 [10] 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 [225, 226].

Herein, for this topic extraction task, we adopted the newly released GPT-4 model21. 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>

...”

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

"<1st 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 Table 11 and Table 12.

4.3.1. RQ3, What benefits are achieved by adopting Open Tracking 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 Table 11 and 12.

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 Table 11 and Table 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 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.

21https://openai.com/product/gpt-4
### Table 11: Sub-Topics for Each Topic of Each Tool (Part 1)

| Topic | Sub-Topics |
|-------|------------|
| **Datadog** | 1) Distributed tracing in microservice architectures 2) Observability in business application performance | 3) Automated alerting and incident notification on performance issues 4) Addressing errors in distributed service architectures 5) Enhancing request-based event tracking 6) Performance testing and analyzing event rate and event time | 7) Chaos Engineering and transaction testing 8) Monitoring and reporting | 9) Preventing and optimizing transaction costs 10) Measuring and analyzing API and database performance | 11) Application monitoring and performance | 12) Microservices and distributed tracing in microservice architectures | 13) Distributed tracing for performance analysis 14) Improving error handling in cloud-native architectures | 15) Monitoring and scaling in continuous integration; |
| New Relic | Sentry |  |
|----------|--------|---|
| **1) Enhancing tool performance in applications** | **1) Streamlining problem-solving with New Relic** | **1) Environment support and management in cloud architecture** |
| **2) Simplifying error resolution** | **2) Managing user experience in real-time production environments** | **4) Open-source tools for resource optimization** |
| **3) Streamlining software development with New Relic** | **3) Identifying and resolving performance issues in applications** | **5) Microservice architecture with New Relic** |
| **4) Optimizing user experience in real-time production environments** | **4) Implementing modern architecture with New Relic** | **6) Defining and measuring user experience in real-time production environments** |
| **5) Enhancing site performance and error handling** | **5) Troubleshooting in microservice architecture** | **6) Automating application performance with New Relic and Azure integration** |
| **6) Monitoring memory and CPU usage** | **6) Optimizing and measuring microservice architecture** | **7) Streamlining large-scale deployments with New Relic** |
| **7) Analyzing page load times and tracing issues with New Relic** | **7) Measuring microservice performance with New Relic metrics** | **8) Ensuring smooth deployment through containerization and team collaboration** |

**Table 12: Sub-Topics for Each Topic of Each Tool (Part 2)**

| Usability | Development | Architecture | Tracing | Measurement | Deployment & Integration |
|----------|-------------|--------------|---------|-------------|--------------------------|
| **1) Enhancing tool performance in applications** | **1) Enhancing site performance and error handling** | **1) Optimizing app architecture for high scalability** | **1) Enhancing function tracing with metrics** | **1) Enhancing application performance with New Relic and Azure integration** |
| **2) Simplifying error resolution** | **2) Monitoring memory and CPU usage** | **2) Implementing modern architecture with New Relic** | **2) Leveraging transaction tracing for error detection** | **2) Streamlining large-scale deployments with New Relic** |
| **3) Streamlining problem-solving with New Relic** | **3) Identifying and resolving performance issues in applications** | **3) Analyzing page load times and tracing issues with New Relic** | **3) Analyzing page load times and tracing issues with New Relic** | **3) Ensuring smooth deployment through containerization and team collaboration** |
| **4) Optimizing user experience in real-time production environments** | **4) Streamlining software development with New Relic** | **4) Defining and measuring user experience in real-time production environments** | **4) Integrating New Relic agents for better monitoring and alerting** | **4) Streamlining large-scale deployments with New Relic** |

**Environment and Usability**

| Environment and Usability | Developer Tools |  |
|---------------------------|-----------------|---|
| **1) Environment support and management in cloud architecture** | **1) Cloud infrastructure development** | **1) Navigating change in technology and deployment** |
| **2) Application performance and observability** | **2) Adapting to changing needs in development** | **2) Scaling microservice architecture** |
| **3) End-user management in cloud architecture** | **3) Troubleshooting in microservice architecture** | **3) Leveraging OpenTelemetry for enhanced observability** |
| **4) Open-source tools for resource optimization** | **4) Optimizing time and cost in DevOps** | **4) Building distributed observability solutions** |

**Optimization**

| Optimization |  |
|--------------|---|
| **1) Optimizing application performance** | **1) Big Data and High-Performance Architecture** |
| **2) Advancing Agile DevOps** | **2) Enhancing observability in Software Architectures** |
| **3) Handling in React Applications** | **3) Handling Massive Data Sets** |
| **4) Memory Management and Debugging** | **4) Scalable Enterprise DevOps** |
| **5) Device-Optimized Development** | **5) Agile Development for Complex Systems** |
| **6) Scalable Infrastructure** | **6) Building Robust and Adaptable Applications** |
| **7) Collaborative Development and Security** | **7) Optimizing Resource Estimation in Software Development** |
| **8) Continuous Integration and DevOps** | **8) Automated Build and Release Processes** |

**Deployment & Integration**

| Deployment & Integration |  |
|--------------------------|---|
| **1) Optimizing test performance in deployment** | **1) Adaptable Data Architecture** |
| **2) Adapting to emerging technologies** | **2) Scalable and Resilient Table Architecture** |
| **3) Streamlining cloud-native application management** | **3) Community-driven Frameworks for Business Applications** |
| **4) Monitoring AWS services with custom metrics** | **4) Secure Data Management and Analytics** |
| **5) Adopting new monitoring platforms for enhanced observability** | **5) Cloud-based App Development** |
| **6) Streamlining cloud-native deployment and integration** | **6) Technology Selection and Tracking** |

**Environment**

| Environment |  |
|-------------|---|
| **1) Enhancing tool performance in applications** | **1) Enhancing site performance and error handling** |
| **2) Simplifying error resolution** | **2) Monitoring memory and CPU usage** |
| **3) Streamlining problem-solving with New Relic** | **3) Identifying and resolving performance issues in applications** |
| **4) Optimizing user experience in real-time production environments** | **4) Streamlining software development with New Relic** |
| **5) Microservice architecture with New Relic** | **5) Defining and measuring user experience in real-time production environments** |
| **6) Automating and measuring application performance** | **6) Automating and measuring application performance** |
| **7) Cross-functional team collaboration in software development** | **7) Cross-functional team collaboration in software development** |

**Deployment**

| Deployment |  |
|------------|---|
| **1) Enhancing test performance in deployment** | **1) Adaptation to emerging technologies** |
| **2) Adapting to emerging technologies and custom workloads** | **2) Streamlining cloud-native application management** |
| **3) Streamlining cloud-native deployment and integration** | **3) Streamlining cloud-native deployment and integration** |

**Security**

| Security |  |
|----------|---|
| **1) Enhancing tool performance in applications** | **1) Enhancing site performance and error handling** |
| **2) Simplifying error resolution** | **2) Monitoring memory and CPU usage** |
| **3) Streamlining problem-solving with New Relic** | **3) Identifying and resolving performance issues in applications** |
| **4) Optimizing user experience in real-time production environments** | **4) Streamlining software development with New Relic** |
| **5) Microservice architecture with New Relic** | **5) Defining and measuring user experience in real-time production environments** |
| **6) Automating and measuring application performance** | **6) Automating and measuring application performance** |
| **7) Cross-functional team collaboration in software development** | **7) Cross-functional team collaboration in software development** |
| **8) Streamlining cloud-native deployment and integration** | **8) Streamlining cloud-native deployment and integration** |
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 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 focuses 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 tracking 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. RQ5. 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 Table 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 issues in agents for monitoring and monitoring. 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 toward 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 RQ1, 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 an 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 (RQ3 and RQ4) 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, OpenTelemetry, Dynatrace and AppDynamics are the most discussed tools in the selected online media (RQ3). 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 [27]". 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 [228]. 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 [15].

**Construct validity.** Construct validities are concerned with issues that to what extent the object of study truly represents theory behind the study [15]. The RQs and the classification schema adopted might might suffer from 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 (RQ3-RQ4) may fall short 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 Bitemp topic model [229], 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 [15]. 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. [230] 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. [231] propose using tracing data to extract service metrics, dependency graphs and workflows 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. [232] propose ASTRacer, a tracing tool for the Apache Hadoop\(^2\) distributed file system. The authors point out the shortcomings of tracing tools like Zipkin, Jaeger and Htrace\(^3\), 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. [233] 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 [234], Jaeger, and Canopy [235], 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. [236] 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 XTrace [237], 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. [238] 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. [239] 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 solution, Jaeger and other tracing tools can be used to collect application level information. Avritzer et al. [240] propose PPTAM, a set of tools for performance testing and performance-based application monitoring. They also extend their approach in [241] to detect performance anti-patterns.

---

\(^2\)https://hadoop.apache.org

\(^3\)http://htrace.org
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., [242], where Barker compares Zipkin, Jaeger, and Appdash. 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.

References

[1] Brooks, No Silver Bullet Essence and Accidents of Software Engineering, Computer 20 (1987) 10–19.
[2] N. E. Fenton, S. L. Pfleeger, Software Metrics: A Rigorous and Practical Approach, PWS Publishing Co., USA, 2nd edition, 1998.
[3] B. C. Society, R. A. of Engineering (Great Britain), 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, 2004.
[4] R. Kalman, On the general theory of control systems, IFAC Proceedings Volumes 1 (1966) 491–502. 1st International IFAC Congress on Automatic and Remote Control, Moscow, USSR, 1960.
[5] L. Finkelstein, M. Leaning, A review of the fundamental concepts of measurement, Measurement 2 (1984) 25–34.
[6] Merriam-Webster Dictionary, trace, https://www.merriam-webster.com/dictionary/trace, accessed on July 7, 2022.
[7] Merriam-Webster Dictionary, Telemeter, https://www.merriam-webster.com/dictionary/telemeter, accessed on July 7, 2022.
[8] I. Magi, Distributed tracing for dummies, https://plumbir.io/blog/tracing/distributed-tracing-for-dummies, 2020. Accessed on July 7, 2022.
[9] V. Garoussi, M. Felderer, M. V. Müntylä, Guidelines for including grey literature and conducting multivocal literature reviews in software engineering, Information and Software Technology 106 (2019) 101–121.
[10] OpenAI, ChatGPT, https://chat.openai.com/api/docs, accessed on May 4, 2023.
[11] C. Wohlin, Guidelines for Snowballing in Systematic Literature Studies and a Replication in Software Engineering, in: International Conference on Evaluation and Assessment in Software Engineering, Easy ’14, Association for Computing Machinery, New York, NY, USA, 2014, pp. 1–10.
[12] V. R. Basili, A. Trendowicz, M. Kowalczyj, J. Heidrich, C. Seaman, J. Münch, D. Rombach, Aligning Organizations Through Measurement: The GQM+Strategies Approach, The Fraunhofer IESE Series on Software and Systems Engineering, Springer International Publishing, 2014.
[13] V. R. Basili, G. Caldiera, H. D. Rombach, The Goal Question Metric Approach, John Wiley & Sons, 1994.
[14] B. Kirchenham, S. Charters, Guidelines for performing Systematic Literature Reviews in Software Engineering, Technical Report Ebse 2007-001, Keele University and Durham University Joint Report, 2007.
[15] C. Wohlin, P Runeson, M Höst, M. C. Ohlsson, B. Regnell, Experimentation in Software Engineering, 2012.
[16] N. E. Fenton, S. L. Pfleeger, Software Metrics: A Rigorous and Practical Approach, PWS Publishing Co., USA, 2nd edition, 1998.
[17] The OpenTelemetry Authors, OpenTelemetry, https://opentelemetry.io/docs/, accessed on July 7, 2022.
[18] H. Naushan, Topical Clustering with Latent Dirichlet Allocation, https://towardsdatascience.com/topic-modeling-with-latent-dirichlet-allocation-efe9c7290f8, 2020. Accessed on Dec. 29, 2022.
[19] SPDX Workgroup, SPDX License List, https://spdx.org/licenses/, 2022. Accessed on July 7, 2022.
[20] AppDash, Github repository of AppDash, https://github.com/sourcegraph/appdash, accessed on Jan. 11, 2023.
[21] AppDash, Github repository of AppDash, https://github.com/sourcegraph/appdash#language-support, accessed on Jan. 11, 2023.
[22] AppDynamics, Github repository of https://github.com/Appdynamics, accessed on July 7, 2022.
[23] AppDynamics, Agent Installation by Type, https://docs.appdynamics.com/appd/22/s/latest/en/application-monitoring/install-app-server-agents#id-1-agentinstallationbytype, accessed on July 7, 2022.
[24] ContainIQ, Github repository of ContainIQ, https://github.com/containiq/containiq-deployment, accessed on Jan. 11, 2023.
[25] ContainIQ, Using ContainIQ, https://github.com/containiq/contai
[26] DataDog, Github repository of DataDog, https://github.com/DataDog/DataDog, accessed on July 7, 2022.
[27] Dynatrace, Github repository of Dynatrace, https://github.com/Dynatrace, accessed on Jan. 11, 2023.
[28] Dynatrace, Instrument your Service with OpenTelemetry, https://www.dynatrace.com/support/help/extend-dynatrace/ope
[29] Elastic APM-Server contributors, Github repository of Elastic APM-Server, https://github.com/elastic/apm-server, accessed on July 7, 2022.
[30] ElasticAPMInstrumentation, Github repository of ElasticAPMInstrumentation, https://github.com/elastic/apm-server, accessed on July 7, 2022.
[31] Haystack, Github repository of Haystack, https://github.com/ExpediaDotCom/haystack, accessed on Jan. 11, 2023.
[32] Haystack, Clients Deployment, https://expediadotcom.github
