Motivations, Challenges, Best Practices, and Benefits for Bots and Conversational Agents in Software Engineering: A Multivocal Literature Review

STEFANO LAMBIASE, SeSa Lab – University of Salerno, Italy
GEMMA CATOLINO, SeSa Lab – University of Salerno, Italy
FABIO PALOMBA, SeSa Lab – University of Salerno, Italy
FILOMENA FERRUCCI, SeSa Lab – University of Salerno, Italy

Bots are software systems designed to support users by automating a specific process, task, or activity. When such systems implement a conversational component to interact with the users, they are also known as ***conversational agents***. Bots—particularly in their conversation-oriented version and AI-powered—have seen their adoption increase over time for software development and engineering purposes. Despite their exciting potential, ulteriorly enhanced by the advent of Generative AI and Large Language Models, bots still need to be improved to develop and integrate into the development cycle since practitioners report that bots add additional challenges that may worsen rather than improve. In this work, we aim to provide a taxonomy for characterizing bots, as well as a series of challenges for their adoption for Software Engineering associated with potential mitigation strategies. To reach our objectives, we conducted a *multivocal literature review*, reviewing both research and practitioner's literature. Through such an approach, we hope to contribute to both researchers and practitioners by providing (i) first, a series of future research routes to follow, (ii) second, a list of strategies to adopt for improving the use of bots for software engineering purposes, and (iii) third, enforce a technology and knowledge transfer from the research field to the practitioners one—that is one of the primary goal of multivocal literature reviews.

CCS Concepts: • **General and reference** → **Surveys and overviews**; • **Software and its engineering** → **Software system structures**; • **Human-centered computing** → *Interaction design.*

Additional Key Words and Phrases: bot, conversational agent, software engineering, literature review

ACM Reference Format:
Stefano Lambiase, Gemma Catolino, Fabio Palomba, and Filomena Ferrucci. 2018. Motivations, Challenges, Best Practices, and Benefits for Bots and Conversational Agents in Software Engineering: A Multivocal Literature Review. *J. ACM* 37, 4, Article 777 (August 2018), 35 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

In the last decade, continuous human activities increased—both in private and working life—thus raising the necessity for automation. Consequently, the adoption of ***bots*** [134, 150]—i.e., software systems designed to automate a specific function or set of activities—has grown and available for a plethora of purposes [22, 141]. In particular, ***conversational agents (CAs)*** [141, 149, 150]—also known as *chatbots*—are bots that communicate with users, by natural language or similar, using a communication channel. Recently, CAs—and, more generally, bots—have started to be adopted

Authors' addresses: Stefano Lambiase, slambiase@unisa.it, SeSa Lab – University of Salerno, Salerno, Italy; Gemma Catolino, gcatolino@unisa.it, SeSa Lab – University of Salerno, Salerno, Italy; Fabio Palomba, fpalomba@unisa.it, SeSa Lab – University of Salerno, Salerno, Italy; Filomena Ferrucci, fferrucci@unisa.it, SeSa Lab – University of Salerno, Salerno, Italy.

Manuscript submitted to ACM

in the software development field [134, 146]. For example, practitioners use bots to automate software maintenance and evolution activities. Similarly, they adopt bots for collaboration, speeding up communication and knowledge-sharing. Besides proposing them, the research community worked on frameworks to ease the development process of conversational systems.

The exploration of bots and conversational agents in systematic literature reviews reveals a focus on both technological and strategic applications within software engineering. Lewandowski et al. [135] and Suhaili et al. [151] emphasize the technical construction and advocate for considering interaction design and natural language capabilities. Motger et al. [141] and Del Carpio and Angarita [115] discuss integration into user interactions and software development. Moguel-Sánchez et al. [139] examine practical applications, such as project management and static code analysis. As a common finding, despite their potential, bots, especially with Generative AI and Large Language Models, present development and integration challenges.

To support both the research community—providing new research opportunities—and practitioners—delivering knowledge for adopting bots and CAs—we carried out a *multivocal literature review (MLR)* [120, 142], following the guidelines of Garousi et al. [120]. Specifically, we aimed to understand the current role and impact of bots and CAs in the software engineering (SE) field, focusing on the (1) motivations for their adoption, (2) challenges, (3) best practices, and (4) benefits coming from their adoption.[1]

Our work complements and diverges from existing literature by reviewing both academic and grey literature and identifying best practices for bot design and adoption. By focusing on engineering and interaction design challenges, we provide guidelines to enhance the practical application of bots in the industry. Additionally, this research aims to be useful to both researchers and practitioners. Moreover, the decision to conduct a Multivocal Literature Review (MLR) rather than an SLR for studying bots in software engineering is supported by Garousi et al. [120]. The rapid technological evolution led by practitioners and the lack of extensive academic publications necessitates incorporating grey literature (GL) for current insights [144, 151]. The practical application of bots further underscores the need for GL to understand real-world impacts. Aligning academic research with practical experiences provides a comprehensive synthesis of knowledge [120], and including practitioner-driven insights is crucial to avoid publication bias [144, 151].

We collected 107 literature items—79 formal studies and 28 grey ones. Results show how CAs are complex to design and deploy despite the diffusion of tools to support their development. Moreover, we noticed that these systems are prone to several problems that undermine their adoption. In particular, interaction—like interruption and noise—emerged as the most critical issue. Nevertheless, we discovered that bots and CAs benefit the practitioners' community considerably, justifying putting effort into increasing their adoption. In terms of contributions, we provide the following:

- A taxonomy to categorize and describe bots and conversational agents based on the motivation of their use for software development purposes.
- A list of challenges arising from adopting bots and conversational agents.
- A set of best practices to deal with bots and conversational agents and an association between such practices and challenges (reported in the Discussion section).
- A list of benefits when adopting bots and conversational agents in the software development context.
- An **online appendix** [133] containing all our findings and all the data gathered during the steps of our multivocal literature review.

[1]From now and for the rest of the paper, we will use the term "bot" to refer to both bots without and with a conversational component (CAs) when it is not necessary to put in evidence the conversational component.

These findings can be helpful for researchers, proposing new research opportunities and standardizing the terminology for referring to bots and CAs, and practitioners, reporting best practices and knowledge to ease the adoption and development of conversational systems.

2 BACKGROUND AND RELATED WORK

According to Storey and Zagalsky [22, 85, 150], a *software bot* is *"a conduit or an interface between users and services, typically through a conversational user interface"* [150], that can be exploited for automating processes/activities, thus improving productivity [22, 150]. For this reason, software bots are widely used for different purposes. For instance, in the context of software development, they have been adopted for automating application deployment—e.g., build, test, and report—or for supporting developers during communication activities—e.g., agenda and information retrieval [22, 32, 40].

Conversational agents (CAs)—also known as chatbots— are particular types of software bots that use communication channels like *Slack*, *Teams*, or *Discord* to interact with users, generating human-like conversations [3, 156].

The properties of bots and conversational agents, other than their potential applicability to various fields, have attracted the interest of multiple research communities—e.g., Software Engineering and Human-Computer Interaction—over the last few years. Despite being a young research field, we have identified six systematic literature reviews that synthesize state of the art on the usage of bots and CAs in the software engineering context [135, 139, 141, 144] and in related fields [151].

Lewandowski et al. [135] conducted a systematic literature review aiming to provide a structured overview of how conversational agents might be used from a strategic standpoint, namely how they can be used within work and company processes, other than governance structures. The authors found 21 relevant scientific records from 2015 to 2020. The key findings of the systematic literature review suggested that most of the papers focused on the technical aspects behind conversational agents. At the same time, other perspectives should be considered when designing more practical and usable CAs. These aspects concern (1) natural language capabilities and training and (2) interaction design, namely, the way a conversational agent can interact with its stakeholders. According to Lewandowski et al. [135], the missing analysis of these aspects may prevent the broader adoption of conversational agents in the industry. Concerning this literature review, our work shares the goal of eliciting the best practices to design conversational agents. Moreover, our analysis also considers themes connected to motivations, challenges, and benefits of conversational agents and bots in software engineering. Finally, our scope is larger since we also include resources from the grey literature, thus including complementary perspectives. As a result, we provide a more comprehensive analysis of the topic of interest.

Suhaili et al. [151] investigated the technology employed to develop conversational agents. Indeed, they primarily focus on the components and techniques used to design those tools, other than on the datasets for their training, evaluation metrics, and application domains. Their review covered the scientific literature from 2011 and 2020. The authors reported that deep and reinforcement learning represent popular methods for understanding users' intents and generating appropriate responses. In addition, they found that Twitter[2] and Airline Travel Information Systems[3] represent the most widely used datasets to train and test the capabilities of conversational agents. Finally, metrics such as accuracy, F1-score, and BLUE index are more frequently used to evaluate the built agents. The work by Suhaili et al. [151] is complementary to ours; besides their goal, we aim to elicit the best practices for designing bots and conversational agents.

[2]The Twitter dataset: https://www.kaggle.com/datasets/jp797498e/twitter-entity-sentiment-analysis.
[3]The Airline Travel Information Systems dataset: https://www.kaggle.com/datasets/hassanamin/atis-airlinetravelinformationsystem.

Motger et al. [141] conducted a systematic literature review of secondary literature, reviewing 28 articles. Their goal was to analyze the anatomy of CAs by providing a taxonomy regarding (1) the human-computer interaction features more relevant for users when interacting with CAs, (2) which techniques are used for the design and implementation of CAs, and (3) which approaches are used for the training phase of the NLP and NLU modules of bots. Our work can be considered complementary to the one by Motger et al. [141]. First, we focus on the design of both bots and conversational systems, hence providing a more extensive overview than Motger et al., who only analyzed CAs. More importantly, we approach the literature review from the software engineering perspective rather than artificial intelligence. As such, we are interested in providing guidelines and recommendations on how to engineer bots and CAs rather than understanding which are the best practices to train their internal artificial intelligence components. Moreover, we elaborate on additional concerns that may inspire further research in software engineering, such as the definition of a taxonomy to characterize bots and a list of benefits from their adoption. Last but not least, our work is based on a multivocal review [142] and, therefore, we analyze both white and grey literature by following the guidelines by Kitchenham et al. [131] and Garousi et al. [120].

Moguel-Sánchez et al. [139] conducted a systematic literature review to identify (1) the activities for which bots are used in software development, (2) the benefits, and (3) the problems. They reviewed 83 primary studies and conducted a thematic analysis to organize their quantitative and qualitative findings. They found that bots are mainly used for project management, to automate tasks with a low level of abstraction, such as tagging pull requests and commits, assigning team members to code reviews, performing static code analyses, and tracking changes in project repositories. Divergently from Moguel-Sánchez et al. [139], we included in our research also the grey literature, resulting in a more comprehensive knowledge of the topic other than a larger quantity of literature. Moreover, we also investigated the best practices associated with using bots in SE and created a mapping between challenges and best practices as an ulterior contribution to our work. Finally, regarding the challenges, we rely on already published taxonomy to enhance it and continuously extend the state of the art.

Del Carpio and Angarita [115] conducted an SLR studying the support given by software assistants (SAs)—including bots—to the entire software development process. Specifically, they studied (1) what processes are supported by SAs, (2) how practitioners interact with SAs, (3) what techniques are used to implement SAs, and (4) what challenges face SAs. They reviewed 40 studies and found that most SAs are—actually—conversational agents, and their interaction with users is far from optimal. The work by Del Carpio and Angarita [115] is complementary to ours; besides their goal, we aim to elicit the best practices for designing bots and conversational agents, and we focus on the benefits of using bots. Moreover, we included the grey literature, which helped us identify a set of practices to complement the findings of the white literature.

3 RESEARCH STUDY DESIGN

The *goal* of the study is to understand the current role and impact of bots and conversational agents in software engineering, focusing on the (1) motivations for their adoption, (2) challenges, (3) best practices, and (4) benefits provided to software engineers. The purpose is to provide a comprehensive overview of the matter, which might be useful to learn about the current state of the art and of the practice. Moreover, we want to stimulate further research to be pursued. The *perspective* is of both researchers and practitioners: the former are interested in having a unique resource to use as a starting point to deepen their knowledge of the research field; the latter are interested in understanding best practices, challenges, and benefits of using bots in practice.

To achieve our goal, we performed a *Multivocal Literature Review (MLR)* [142, 145], namely, a research process where past published (formal) literature (e.g., journal and conference papers) and *grey literature* (e.g., blog posts, videos, and white papers) on the topic investigated are systematically identified, selected, and critically assessed. An MLR builds on top of the concept of *Systematic Literature Review (SLR)* [130, 131]—where only the formal literature is taken into account—to provide benefits not only to the research community but also to practitioners.

We relied on two guidelines when defining the protocol and conducting the review. In particular, concerning the formal literature analysis, we adopted the well-established approaches by Kitchenham et al. [130, 131]. In addition, we followed the guidelines defined by Garousi et al. [120] when it comes to the grey literature. Finally, when organizing and reporting the results, we followed the *"General Standard"* and *"Systematic Reviews"* guidelines provided by the *ACM/SIGSOFT Empirical Standards*.[4] In the following subsections, we describe the main steps of each phase.[5]

3.1 Planning the Review

3.1.1 Initiation Phase. The Initiation phase consisted of three sub-phases.

Initial Examination of Previous Studies. To collect insights and basic knowledge about the adoption of bots and conversational agents in software engineering, we conducted a preliminary investigation in which we reviewed a few existing studies on the matter, e.g., [135, 151]. In particular, we used the *Repository for Bot-related Research* [108]—i.e., a publicly available repository of formal literature maintained by the software engineering research community. After reading the studies, we discarded them in a *"throw-away prototyping"*—as suggested in the literature [120, 131]—to start the main review process with basic knowledge, avoiding possible bias or influence from no systematic intervention.

Motivation and Needs Identification. Before choosing to conduct an MLR, Garousi et al. [120] suggested investigating the current state of the art on the topic of interest to identify limitations that grey literature could address. For this reason, Garousi et al. [120] defined a checklist that consists of seven "Yes" /"No" questions related to the complexity and interest of the topic. A high number of "Yes" justifies the inclusion of grey literature in the systematic analysis. This step is crucial to prevent the inclusion of grey literature from being of no benefit to research and resulting in a futile effort. As for our study, Table 1 reports the filled checklist. To fill such checklist, we used the knowledge obtained during the initial phase (described in the previous paragraph) and the SLRs on the topic [135, 143, 144, 151], as suggested by Garousi et al. [120]. Six out of seven responses were "Yes," which allowed us to decide to proceed with including grey literature in our study.

- The first two questions were about the topic itself rather than the work. We answered the first question positively since the use of bots in software engineering is intrinsically a complex topic from both a technical and social perspective [A1–A3]. Moreover, bot technology advances primarily on the practitioner side rather than the research side, thus leading to a fast evolution of technology that is difficult to capture by research studies alone, often requiring slow publication times to facilitate systematic review and process. Furthermore, the broad adoption of bots, coupled with the recent advent of generative AI, leads to a wide range of issues to analyze, many of them from both a research and practitioner perspective. Regarding the second question, since the topic of bots in software engineering is young, there is not enough literature to determine the presence of a lack of consensus. Nevertheless, the youthfulness of the topic further poses the need to conduct studies involving the gray literature as well.

[4] *ACM/SIGSOFT Empirical Standards*: https://github.com/acmsigsoft/EmpiricalStandards.
[5] The entire process is depicted in Appendix A—Fig. 1 [133].

Table 1. Questions to decide whether to conduct an MLR [120].

#	Question	Answer
1	Is the subject "complex" and not solvable by considering only the formal literature?	Yes
2	Is there a lack of volume or quality of evidence, or a lack of consensus of outcome measurement in the formal literature?	No
3	Is the contextual information important to the subject under study?	Yes
4	Is it the goal to validate or corroborate scientific outcomes with practical experiences?	Yes
5	Is it the goal to challenge assumptions or falsify results from practice using academic research or vice versa?	Yes
6	Would a synthesis of insights and evidence from the industrial and academic community be useful to one or even both communities?	Yes
7	Is there a large volume of practitioner sources indicating high practitioner interest in a topic?	Yes

***Note**: The possible answers to each question are "Yes" or "No". One or more "Yes" responses suggest that it could be useful to conduct an MLR.*

- The third question is about the subject under study in the work. Literature has repeatedly shown that the study context is central to software engineering research [118, 147]. This becomes even more true when it comes to bots and their impact on practitioners. The present work aims to elicit the motivations, challenges, strategies, and benefits of practitioners' use of bots for different purposes and in different conditions. Hence, the contribution of grey literature to characterize this context is essential. For such a reason, we answered the third question positively.
- Questions four, five, and six are about the work objectives. As mentioned above, the goal of the work is to provide knowledge suitable for (1) opening new avenues of research and (2) improving the use and adoption of technology by practitioners. Given this goal, corroborating academic research with practitioners' knowledge is essential. In fact, where formal research can provide a systematic and robust set of information, grey literature can augment that information with concrete and applicable strategies useful to both worlds [120]. On the other hand, academic research may shine precisely in disproving false beliefs born in the practitioner world. In summary, the goal of this paper is certainly to provide knowledge that is useful to both worlds, and that is as true and reliable as possible.
- Regarding question seven, we found significant evidence about the extensive adoption of bots by practitioners, consequentially increasing the importance of the context on the matter [144, 151]. Beyond that, the professional world's commitment to process automation has led to the development of many frameworks to support the use, development, and adoption of bots.
- Last, including grey literature can provide current perspectives and address the gaps in formal academic literature. First, it helps to avoid publication bias, although it is noted that the grey literature accessed may not represent all unpublished studies. Moreover, excluding grey literature would risk losing critical insights and perspectives on the topic, a conclusion also observed in the case study presented in the guidelines papers [120, 137].

Goals and Research Questions Definition. Our main objective was to provide an overview of the use of bots and conversational agents for software engineering purposes. We investigated four aspects that led to multiple research questions to reach our goal.

First, we aimed to investigate the reason behind using bots in software engineering; our goal was to create a taxonomy to categorize bots based on their use. Hence, we formulated our first research question.

> **RQ_1—Motivations and Goals**
>
> *Which motivations and goals cause the use of bots and conversational agents in software engineering?*

While the prevalence of bots and conversational agents has been growing significantly in the recent past [144, 151], developing, adopting, and interacting with them is still problematic [A1, A156]. For this reason, the second research question aimed at categorizing and mapping the challenges faced by practitioners, along with possible solutions.

> **RQ_2—Challenges**
>
> *What are the challenges related to using bots and conversational agents in software engineering?*

Based on the considerations above, we proceeded with our analysis by extracting a list of known best practices from the formal and grey literature, with the goal of helping practitioners adopt and develop bots for engineering purposes. Moreover, we aimed to identify possible solutions to the limitations elicited while answering the second research question.

> **RQ_3—Best Practices**
>
> *What are the best practices for using bots and conversational agents in software engineering?*

Finally, the last research question aimed to investigate the impact of adopting bots, collecting and reporting the benefits of their usage. This led us to the definition of our last research question.

> **RQ_4—Benefits**
>
> *What are the benefits of using bots and conversational agents for software engineering?*

3.1.2 Search Phase. The Search Phase consists of two sub-phases.

Data Sources Selection. This phase aims to identify the most reliable databases to extract the literature and start our process. This step differs between formal and grey literature.

- *Formal Literature.* To collect the formal literature, we selected *Scopus*,[6] *IEEE Xplore*,[7] and *ACM Digital Library*[8] as data sources. These databases are considered the top three among the computer science formal literature sources,[9] other than have been used in various other literature reviews [119, 121, 125, 131]. Furthermore, they were recommended in the guidelines [120, 130] since they can provide a complete overview of the published research. As recommended by [120, 130], we took all the results extracted from the first research and filtered them using eligibility criteria.

[6] *Scopus* website: https://www.scopus.com/home.uri
[7] *IEEE Xplore* website: https://ieeexplore.ieee.org/Xplore/home.jsp
[8] *ACM Digital Library* website: https://dl.acm.org/
[9] The top list of computer science research databases: https://paperpile.com/g/research-databases-computer-science/

- *Grey Literature.* To search the grey literature, we used the Google search engine. The reason behind our choice is that Google (1) has been used by other MLRs on Software engineering [119, 121, 125], and (2) is recommended in various MLR guidelines [109, 120]. However, it is worth noting that deciding when to stop the search process is a complex problem—mainly caused by the high volume of available items [120, 142]. For this reason, we adopted two different choices—as recommended by Garousi et al. [119–121]:
  - We stop the research when no new concepts emerge from the search results anymore, i.e., *Theoretical saturation*;
  - We applied the *Effort Bounded* that regards the inclusion of the top N search engine hits only. We analyzed the first ten pages produced by Google, observing that its algorithm retrieves and shows the most relevant results in the first few pages.

Search String Design. In this phase, we defined different search strings to collect the formal and grey literature. In particular, we used the knowledge obtained during the initial examination to draft the search string. Its definition was based on (i) key terms on the topic of interest–gathered from relevant papers, (ii) synonyms and alternative terms, and (iii) logical connectors to combine different terms in different ways.

The initial version of the search string included basic terms such as "bots" and "software engineering." During the preliminary exploration phase (Section 3.1.1), we began to use the string and improve it in accordance with the results obtained. We focused on the individuation of synonyms, which led first to the expansion of the search string for formal literature and then to the detailed definition of the different strings for grey literature.

As for the formal literature, the final search string is shown in the box below. As it is possible to see, we did not include key terms of our research questions, e.g., *"goals"*, since we aimed to collect as much formal literature as possible, even if the phase led to high effort.

Formal Literature Search String

("bot(s)" OR "chatbot(s)" OR "chat bot(s)" OR "conversational agent(s)") AND ("software engineering")

As for the grey literature, following the guidelines provided by [121, 132], we started from a general search string to define a more specific one based on the goal of each research question. The reason behind our choice arose from the noise present in the output that characterizes informal databases, like the Google search engine. Table 2 reports the search strings for the grey literature based on the research questions of our study.

3.1.3 Eligibility Criteria Phase. In this phase, the aim is to define a series of criteria to apply after collecting the literature for filtering and identifying those that provide direct evidence based on our research questions [120, 130]. The phase consisted of two sub-phases: (1) Defining the formal literature eligibility criteria and (2) Defining the grey literature eligibility criteria.

As for the *formal literature*, the eligibility criteria are divided into three sets that had to be evaluated for each paper extracted from the search phase.

(1) *Exclusion Criteria*, namely, criteria that, if met for at least one of them, decree the elimination of the study from the analysis set;
(2) *Inclusion Criteria*, namely, criteria that, if met for at least one of them, decree the inclusion of the study into the analysis set;
(3) *Quality Criteria*, namely, criteria that are used to grade studies and may lead to their elimination if the grade is below a certain threshold.

Table 3 shows exclusion and inclusion criteria. We defined such criteria according to the guidelines by Kitchenham et al. [120, 130] and past literature in the Software Engineering field [125, 132]. Each criterion could be only *True* or *False*. Using these filters, we could exclude all preliminary research results, e.g., workshops or posters, and avoid considering duplicated papers derived from the combination of three data sources. Nevertheless, we decided not to exclude workshop papers reported in the Repository for Bot-related Research [108] since these could be highly related to this work's main reference community. In addition, we included EC6 in order to ensure that the collected papers were concerned with the topic of interest. Moreover, as for other LRs [121, 125, 135, 144, 151], we defined our inclusion criteria to match our objective and research questions. Regarding our exclusion criteria, it is worth discussing the EC5—Studies written before 2014. We decided to apply such criteria based on the findings of previous LRs [144, 151], which report that literature on software bots—particularly in their communication-oriented form—started growing after 2014. The other exclusion criteria are widespread among other LRs, and some have been applied using the filters integrated into the selected databases for data source selection.

Regarding the quality criteria, we defined them as the following:

Q1 Is the valuable information for the review clearly reported and indicated in the paper?
Q2 Are the findings clearly reported in the paper?
Q3 Are the RQs or objectives clearly defined?
Q4 How many pages does the paper consist of? (with respect to the top venue formats)

The first three questions could be answered as *"Yes"*, *"Partially"*, *"No"*. Each answer corresponded to a numerical value, i.e., *'1'*, *'0.5'*, *'0'*, respectively. The sum of these values reflects the quality score of the study. If the score was lower than 2, we excluded it from the review. The last one could be answered as *"1 to 2"*, *"3 to 4"*, and *"4 and up"*.

Concerning the *grey literature* eligibility criteria evaluation, we followed the guideline proposed by Garousi et al. [120] combining inclusion and exclusion criteria with quality assessment criteria (Appendix A—Table 1).

Table 2. Search strings for grey literature.

#	Search String	RQ
0	("bot(s)" *OR* "chatbot(s)" *OR* "chat bot(s)" *OR* "conversational agent(s)") *AND* ("software engineering")	General String
1	("bot(s)" *OR* "chatbot(s)" *OR* "chat bot(s)" *OR* "conversational agent(s)") *AND* ("software engineering") *AND* ("goal(s)" *OR* "motivation(s)" *OR* "objective(s)")	RQ_1
2	("bot(s)" *OR* "chatbot(s)" *OR* "chat bot(s)" *OR* "conversational agent(s)") *AND* ("software engineering") *AND* ("challenge(s)" *OR* "limitation(s)" *OR* "negative impact(s)")	RQ_2
3	("bot(s)" *OR* "chatbot(s)" *OR* "chat bot(s)" *OR* "conversational agent(s)") *AND* ("software engineering") *AND* ("best practice(s)")	RQ_3
4	("bot(s)" *OR* "chatbot(s)" *OR* "chat bot(s)" *OR* "conversational agent(s)") *AND* ("software engineering") *AND* ("benefit(s)" *OR* "positive impact(s)")	RQ_4

***Note**: The search string #0 was the same as the search string used for the formal literature and was used as a base for the search strings for the grey literature.*

Table 3. Exclusion and Inclusion Criteria for Formal Literature.

Exclusion Criteria	Inclusion Criteria
EC1 Studies not written in English.	IC1 Studies that report about the uses of bots in Software Engineering.
EC2 Posters.	IC2 Studies that report about the challenges of bots in Software Engineering.
EC3 Workshop papers.	IC3 Studies that report about the best practices in the use of bots in Software Engineering.
EC4 Study not entirely available for free.	IC4 Studies that report about the benefits of using bots for Software Engineering.
EC5 Studies written before 2014.	
EC6 Studies not related to the topic of interest.	
EC7 Duplicated studies.	

3.2 Conducting Phase

3.2.1 Studies Selection Phase. In the studies selection phase, the aim was to apply the planned strategy to collect the literature [120, 130]. The analysis included all studies published from 2014 to 4/4/2024 (Appendix A—Fig. 2 [133]).

As for the extraction and selection of the formal literature, we started by running the search string on the three selected databases, e.g., Scopus, obtaining a set of 2108 studies. We imported the results of the three datasets in Excel, and the first two authors of the paper started the inclusion and exclusion process. As a first filtering step, we started deleting duplicates by the three datasets merged. Then, the remaining set of papers was divided in two, and the first two authors started applying inclusion and exclusion criteria on the two separate sets; the authors determined if they include or exclude the paper starting by reading the title, abstract, and—if necessary—the entire body of the studies. After analyzing 20 papers each, the two authors reviewed corresponding work and discussed; no conflict arose, and then the authors continued applying the exclusion and inclusion criteria, resulting in 173 studies included and 1935 excluded. Next, an ulterior session of filtering was applied jointly by all the authors on the 173 studies that remained; this aimed at identifying potential errors in the inclusion process. After this step, we excluded other 64 studies, obtaining a final set of 109 studies. Finally, the first two authors of the paper applied the quality criteria, obtaining a final set of 79 studies from the white literature.

Regarding the grey literature, the first author started using the search strings on the Google Search Engine to collect the literature. For each search string, the first 10 pages were checked, and the items judged related to the work objectives were imported into an Excel file until theoretical saturation was reached [120]. By doing so, we obtained a set of 52 items. Then, the first two authors of the paper started applying all the eligibility criteria simultaneously, as Garousi et al. [120] suggested. Such a process led to identifying 28 items for the grey literature.

After completing the extraction and selection process for both the formal and grey literature, we combined the two sets. This final step resulted in a comprehensive collection of 107 studies, which formed the basis for the subsequent data extraction phase.

Regarding the distribution of literature for type—i.e., blogs (for grey literature), conferences, workshops, and journals—most of the works are conference papers (52), while 28 are blogs, 13 are workshop papers, and 14 are journal papers. This is not a surprise, seeing that the research topic specifically concerning conversational agents—is extremely recent and tends to be more relegated to conferences and live discussion settings. Moreover, as expected, a large amount of the literature comes from the *International Workshop on Bots in Software Engineering (BotSE)*; it is the main research community on bots and CAs for Software Engineering purposes. In addition, there are also top venues on Software

Engineering, e.g., *International Conference on Software Engineering (ICSE)* and *Transaction on Software Engineering (TSE)* (Appendix A—Fig.3). More details are in the online appendix [133].

3.2.2 Data Extraction Phase. In the data extraction phase, we started from our objectives and research questions to define a series of data extraction forms, i.e., sets of questions to identify useful elements to answer a research question in the studies [120, 130].

Aligning with other similar works, we defined the extraction forms jointly—through brainstorming and discussion—and then the first author applied them. We identified and extracted the relevant data using a pre-defined data extraction form from each selected source that we needed to answer the research questions. We also extracted some general information—e.g., authors names, published year, and keywords—as recommended by guidelines on literature review [120, 130]. To validate and test our forms, we conducted a preliminary extraction on a set of 15 randomly selected sources—like in other literature reviews [119, 121, 125, 138].

From a practical standpoint, we imported all the papers into a new Excel sheet and added a column for each data form identified. Then, we began extracting information from the papers in random order. If the extracted information was different from what had been extracted previously, we would create a new column in the file under the data form group (following a hierarchical structure). However, if the information extracted from a paper had already been indicated by another paper, meaning the column already existed, we would simply mark the corresponding cell with a sign and add any new information. For example, one of the data forms for RQ1 was "what are the reasons for using bots in SE?" Through the paper by Van Tonder and Le Goues [A4], we extracted the information "repair bots, to identify issues in the code and automatically resolve them.". Subsequently, the paper by Monperrus [A5] also reported the same information, so an "X" was added to the corresponding cell in the repair bots column.

All the extracted data are in our online appendix [133]. We maintained traceability between the forms and the research questions as recommended by MLR guidelines [120].

3.2.3 Data Synthesis and Analysis Phase. In the data synthesis phase, we synthesized data previously collected to present them to the target audience in a readable and usable way.

After extracting and dividing the data between the various form groups, the first two authors analyzed them. In this way, we avoided losing important information or misunderstanding data [120, 130]. If the first two authors were unsure how to synthesize a study's results, the plan was to ask the remaining two authors for help; this case never occurred.

To conduct a correct data synthesis, we relied on two qualitative analysis methods, namely, *narrative synthesis*—i.e., the description and ordering of primary evidence with commentary and interpretation of a set of data—and *thematic analysis*—i.e., the summary of studies under a set of recurrent themes in literature [114]. First, we used narrative synthesis to perform an initial analysis of the gathered data from the previous phase. Then, we used thematic analysis to classify and categorize the data.

Concretely, we started with the data extracted in the previous phase and began annotating them with descriptions and labeled comments using keywords, such as "Repair bots" and "Maintenance bots". Subsequently, we identified descriptions that closely resembled each other or appeared to be related and began aggregating them into individual categories. Each category became a new column in the Excel file, and through iterative repetition of this process, some columns were grouped into thematic similarity groups. For instance, "Repair bots" and "Bots used for identifying flaky tests" were grouped under the category test bots. To maintain traceability, within the file, all papers contributing to the generation of a category were marked with an "X" corresponding to the paper and the column dedicated to that category.

The first two authors jointly conducted the steps mentioned above—at least for a subset of the items—to identify possible discussion points to strengthen the final results. At the end of the process, we obtained a new set of insights and information categorized under four macro categories—one for each research question. We reported all our findings in the online appendix for our work [133].

3.3 Results Reporting Phase

Based on the target audience of this review, i.e., practitioners and researchers, we relied on the guidelines provided by Garousi et al. [120] to enhance the readability—particularly from the practitioners' side. For such a purpose, we (1) decided to include a practical-oriented Discussion section (Section 5.3), (2) asked practitioners for feedback on our results, and (3) adopted various tools to enhance results summaries.

4 ANALYSIS OF THE RESULTS

In the following section, we describe the results achieved for each research question and report some statistics of our papers collection.

4.1 RQ_1: On the use of bots and conversational agents for software engineering purposes

As a first goal, we aimed to identify the motivations and objectives behind the adoption of bots for software engineering purposes. Our purpose was to gather general and fundamental pieces of information to (1) define a taxonomy to categorize bots based on their use and (2) better perform the following research step.

Our investigation into answering the first research question (described in Section 3) led to the identification of 13 fundamental motivations for adopting bots and conversational agents in software engineering. In order to provide a more comprehensive and understandable taxonomy, we categorize each of the fundamental motivations into 4 macro categories of bots (Depicted in Appendix B—Fig. 1 [133]). The identified motivations are not mutually exclusive; the same bot can be used for different motivations according to a plethora of reasons, e.g., working roles, needs, and types of projects. In the following section, we elaborated on each category and associated motivations, corroborating the discussion with the primary studies and grey literature supporting them.

4.1.1 Software Product Bots. In this category, we group all the bots and associated motivations of use used to perform automatic operations on source code or related artifacts and activities. We identified the following sub-categories:

Development. This sub-category refers to bots that *are used to generate source code instead of developers or other practitioners*. In our investigation, we found 11 studies, 9 from the white literature [A6–A13] and 2 from the grey one [B80, B81] (≈ 10.28% of the entire dataset).

Part of the literature focuses on the use of bots (in most cases empowered by Generative AI) to automatically write source code [A9, A13][B80, B81] and code for machine learning purposes [A9]; in this way, practitioners can save time and increase productivity. Nevertheless, Generative AI is still raw in this task, and software engineers do not rely heavily on it [A9, A13][B80, B81]. Another part of the literature combined bots and models-based language to generate source code and entire applications [A6, A7, A10–A12, A14]; in this context, bots act like facilitators in creating models representing complex systems and modules. Interestingly, bots to generate smart contracts [A7, A12] and recommendation systems [A11] (thus, bots used to generate other bots) are being proposed.

Refactoring. This sub-category refers to bots that are used to *(1) improve source code* and (2) *avoid and/or resolve technical debts by means of refactoring operations*, i.e., *Refactoring Bots*. Our dataset has 11 studies, 8 from the white

literature [A9, A15–A21] and 3 from the grey one [B82–B84] (≈ 10.28% of the entire dataset).

Some of them are related to technical debt [A15, A17]; for example, bots in such a context are used to refactor code smells automatically. Moreover, bots can also be used to support quality aspects of source code, e.g., by means of quality criteria [A16] or code readability [B82], and provide fixes when static analysis tools detect violations [A18, A19]. In the recent past, refactoring has also been automated using Generative AI, even without satisfying results [A9].

Testing. This sub-category refers to bots that *support testers and similar roles during the testing activity and for executing test code*. Our dataset has 11 studies, 10 from the white literature [A4, A5, A17, A19, A22–A27] and 1 from the grey one (≈ 10.28% of the entire dataset).

Most of the literature regards the use of bots for automation of test cases execution [A4, A5, A17, A22, A23, A26, A27]. Furthermore, we found that many bots are used to identify bugs and perform automatic fixes [A4, A5, A17, A19, A22, A23][B82], i.e., *Repair Bot*. Last, some works propose bots to (i) support developers in writing test code [A27] and (ii) implement heuristics to detect *flaky tests* [A22]—i.e., tests exhibiting both a passing and failing behavior when run against the same code [136].

Configuration. This sub-category concerns bots used to support practitioners during the setup and configuration processes of new software projects. Our dataset has 6 studies, 4 from the white literature [A19, A22, A28–A30] and 1 from the grey one [B84] (≈ 5.61% of the entire dataset).

From the identified literature, two studies propose bots for automatically creating/modifying software configuration files or similar [A28, A29], and four studies [A19, A22, A30][B84] report about bots used to manage and monitor project dependencies.

CI/CD. This sub-category regards the usage of bots *to support everyday activities in the context of CI/CD*. In our investigation, we found 8 studies, 4 from the white literature [A22, A25, A31, A32] and 4 from the grey one [B82, B84–B86] (≈ 7.48% of the entire dataset).

Some studies analyze the usage of bots for monitoring software systems (for example, on GitHub) and performing automatic operations if certain conditions are met (e.g., automatically build the system on branch merge) [A22, A25]. Finally, five works study the use of bots to automatically trigger actions when specific criteria—related to no-code platforms—are met [A22, A31, A32][B82, B85]. For example, automatically creating an issue on GitHub when a bug is reported, or a task is assigned in the collaboration platform of the team (e.g., *Trello* and *Jira*). Moreover, some bots to support DevOps-related activities started to emerge [B86].

4.1.2 Software Process Bots. Under this category, we group all the bots used to improve and enhance communication, collaboration, and management activities. We identified the following three sub-categories:

Team. This sub-category refers to bots used *to support practitioners in handling team quality attributes like turnover and socio-technical aspects*. We found 6 studies, 3 from the white literature [A33–A35] and 3 from the grey one [B82, B85, B87] (≈ 5.61% of the entire dataset).

Regarding the motivations, some bots can be used to support management activities such as risk management [B87] and *community smells* management [A34], i.e., anti-patterns precursors of social debt [152]. Moreover, a study proposes to use bots specifically to reduce *turnover*—the phenomenon of continuous influx and retreat of human resources in a team [116]—in an open-source software community by improving the reviewer selection process in code review [A33]. More related to the open-source communities, bots are used to help onboard new contributors [B82] and alleviate burnout and stress [A35][B85] by—for example—supporting core contributors' activities.

Communication and Collaboration. This sub-category concerns bots used *to help communication and collaboration between practitioners*. During our investigation, we found 8 studies: 7 from the white literature [A6, A8, A33, A36–A39] and 1 from the grey one [B85] (≈ 7.48% of the entire dataset).

Two studies reported that practitioners use bots to facilitate meetings and communication in community chats, encouraging team participation [A37][B85]. Bots can also assist managers with scheduling and planning meetings [A36]. Additionally, bots help detect when two individuals work on the same artifact, preventing code conflicts [A38]. Bots can record past decisions to avoid information loss and support future decisions [A6, A8]. They also identify experts to reduce turnover and assist newcomers [A33, A39].

Task. Practitioners adopt bots *to support activities related to task management*, e.g., evaluation, assignation, and planning. During our investigation, we found 5 studies: 3 from the white literature [A36, A40, A41] and 2 from the grey one [B87, B88] (≈ 4.67% of the entire dataset).

Some of the bots are adopted to find the right person for the right task by using metrics and developers profiling [A40, A41][B87]. More related to the managers, a work proposed an all-round bot for task management assistance; the focus of such a bot is task monitoring, issues management, planning, and also discussion of task characteristics to assigning it to the right person [A36].

Code Review. The review revealed that practitioners use bots to support the code review process, often associating it with pull requests. Our dataset has 6 studies on this topic, 5 from the white literature [A33, A42–A45] and 1 from the grey one [B85] (≈ 5.61% of the entire dataset).

In general, researchers report that the adoption of such bots could effectively benefit software projects in different ways. [A33, A42, A44] For example, turnover decreases, and communication becomes more effective; moreover, pull requests increase in quality, and contributions to open-source projects are more rapid than without the bots. Nevertheless, the risk of introducing noise exists; some studies proposed the adoption of bots to orchestrate code review bots and summarize their activity and outputs more concretely and cleanly [A43, A45].

4.1.3 Knowledge Bots. Under the category Knowledge Bots, we group all the bots used to store, share, and manage information and knowledge about the software project. We identified the following sub-categories:

Documentation. Practitioners use bots *to support documentation writing*. Our dataset has 10 studies regarding this type of bot, 9 from the white literature [A6, A7, A22, A24, A28, A29, A46–A48] and 1 from the grey one [B85] (≈ 9.35% of the entire dataset).

Several works focus their attention on bots used to automatically create and update documenting artifacts [A6, A28, A29][B85] (e.g., code comments, documents, and reports). The state of the art [A7, A22] has proposed bots to automatize the writing of notes for new software product releases, using NLP to synthesize developers' commit messages and comments. Moreover, practitioners started to use Generative AI to write code commits messages [A47, A48]. Other studies propose bots to improve and speed up agile processes by automatizing the agile retrospective step by collecting metrics from different sources [A7, A46]. Last, bots are also used to record and document design decisions and, for some of them, generate static or behavioral models on-the-fly [A6, A7] (e.g., UML graphs).

Metrics. Bots in the Metrics category are used *to monitor the team environment (both product and process) and collect metrics*. We found 7 studies related to this category, 4 from the white literature [A22, A25, A49, A50] and 3 from the grey one [B86–B88] (≈ 6.54% of the entire dataset).

Some studies are related to collecting and analyzing product and process metrics, allowing practitioners to make informed decisions [A49, A50][B87, B88]; this is particularly important when dealing with services and micro-services

applications [A50]. Moreover, also in this case, bots seem to be used as managers' peers to improve the overall management process [B87, B88]. Furthermore, some bots have also been adopted to support DevOps activities by computing metrics and suggesting ways to improve the pipeline [A49][B86].

4.1.4 Emergent Bots. In this last category we have included three reasons for adopting bots that have been emerging in recent times and that we believe are worthy of mention although still in an embryonic stage.

Digital Twins Bots. Bots in this category are integrated into the development workflow *to support specific developers and act as companions*. We identified 19 studies about this type of bot, 16 from the white literature [A17, A22, A24, A25, A35, A51–A61] and 3 from the grey one [B82, B85, B88] (≈ 17.76% of the entire dataset).

The majority of works report that bots are used to support the retrieval of information regarding general implementation questions and, particularly, API documentation and guidelines [A22, A52, A53, A56–A58][B85]. Then, some studies focus on using bots to enforce the application of formatting guidelines or requirements [A17, A59][B82] and good programming practices [A51][B82]. Furthermore, some bots are also used like actual peers during the programming activity (pair-programming bots) [A60, A61]; among these, some are used to detect possible implementation choices that could lead to defects in the code and propose solutions [A55]. Related to the open-source field, some bots act like peers to recommend useful tools to solve a problem [A54], support elite developers in general tasks, and help them stay focused on the work by eliminating noise [A35].

Orchestration Bots. Despite being useful for practitioners, a well-known reported problem with bots is the *noise* arising from their use. In short, since bots are still unable to behave according to the operational context, if their interaction is not well designed, it could result in them providing too much information to practitioners, often unhelpful, causing distraction and discomfort [A1, A25, A63, A134][B85]. To solve this problem, some practitioners operationalize bots specifically designed *to orchestrate and summarize the results of other bots*. In such a sense, these Orchestration Bots act like bot managers. We identified 2 studies about this type of bot, both in the white literature [A43, A45] (≈ 1.87% of the entire dataset).

Wessel et al. [A43] proposed a series of guidelines for developing these types of bots, while Ribeiro et al. [A45] created FunnelBot, a bot designed using the previously mentioned guidelines to orchestrate pull-request bots.

Technology Transfer Bots. Last but not least, and more attractive for researchers, some bots are starting to be used *to support the technology transfer between research and the practitioner field*. Indeed, the limit of research-developed tools is often the poor effort invested in developing their non-functional requirements, resulting in poor quality. Some bots are starting to be developed specifically to surpass such a limitation. We identified 3 studies about this type of bot, 2 in the white literature [A31, A34] and 1 in the grey one [B85] (≈ 2.80% of the entire dataset).

In this sense, Beschastnikh et al. [A31] and Voria et al. [A34] proposed an ecosystem of bots specifically designed to integrate other research-originated bots and make them more easily available to and usable by practitioners.

4.2 RQ_2: On the challenges of bots and conversational agents for software engineering purposes

The diffusion of bots and conversational agents for supporting software development processes has also led to the emergence of new challenges for computer scientists. For this reason, we investigated the challenges related to the interaction with, adoption, and development of bots for software engineering purposes.

Regarding the existing literature, Wessel et al. [156] identified a series of challenges in the interactions with software bots in the open-source development context, with a particular focus on the concept of *"noise"*—namely, annoying bot behaviors, e.g., verbosity and unsolicited actions, which lead to deterioration of communication in teams. In terms of

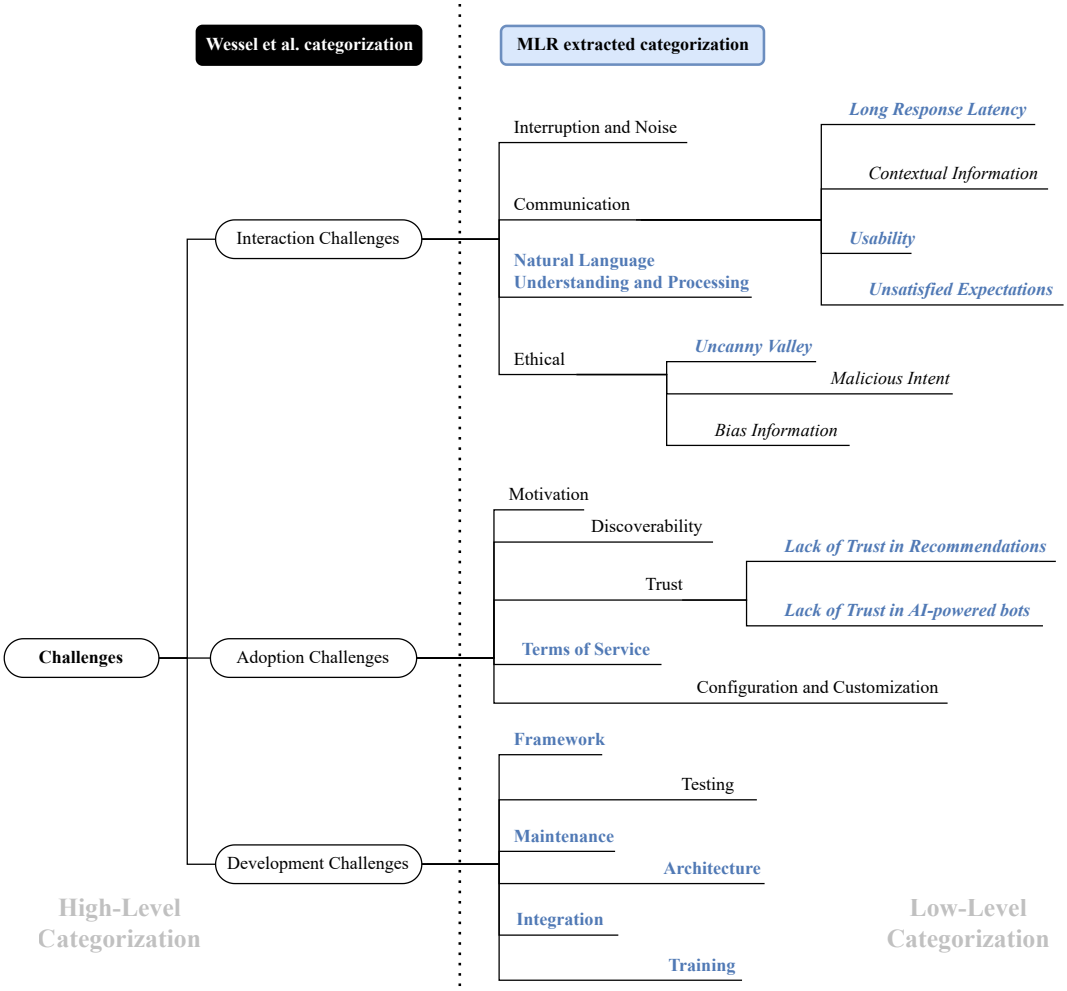

Fig. 1. Categorization of challenges associated with bots in the context of software engineering. This categorization is based on the structure proposed by Wessel et al. [156]. New challenges originated from our work are shown in blue.

contributions, the authors provided a series of challenges experimented by practitioners, categorized under three main categories, i.e., *Interaction*, *Adoption*, and *Development Challenges*.

On the one side, we found challenges already covered by Wessel et al. [156]. On the other side, we contributed state of the art with new challenges—indicated in Figure 1 in bold and red text color. Specifically, we decided to maintain the high-level separation proposed by Wessel et al. [156]—the left side of Figure 1—and to re-design the low-level categorization—the right side of the figure—based on the findings of our MLR. The following subsections report each identified challenge in detail.

4.2.1 Interaction Challenges. Most of the challenges identified by the literature regard the interaction between practitioners and bots. Such interaction can be intended as unidirectional—the user receives a message from the bot and reacts to it—and bidirectional—the user and the bot actively exchange messages. During our investigation we found 27 studies; 17 from the white literature [A1, A3, A19, A25, A39, A43, A61, A63–A71, A134] and 10 from the grey one [B81, B85, B86, B89–B95] that report on this challenge (≈ 25.23% of the entire dataset).

Interruption and Noise. One of the most frequently noted challenges involves certain bots exhibiting a tendency towards verbosity or producing irrelevant output [A1, A25, A43, A63, A70, A134][B85]. This issue, commonly referred to as "noise" in the literature, can significantly affect bot usability as well as team communication and collaboration. The origin of this behavior often lies in the bot's design—sometimes, bots are intentionally programmed to "harass" developers. On the other side, errors in the bot's operation can also result in unexpected and disruptive noise.

Communication. Another challenge refers to the communication between the user and the bot [A1, A3, A19, A25, A39, A64–A68, A70, A71][B81, B85, B86, B89, B90, B92, B93]. For example, **ensuring a short response time** comes to be mandatory for bot use: various bots—especially those based on API calls or off-the-shelf modules—are characterized by long responses to users' inputs and consequent frustration in them [A65][B90]. Nevertheless, architectures of this type are often recommended for these software systems, and developers are faced with a trade-off between high modularity and maintainability—given by using such architectures—and fast response time. To continue, several works report the difficulty of bots in **providing practical and contextualized information**: indeed, bots should have information about the environment in which they are operating (e.g., repositories address, commit template, and team composition), but this knowledge is difficult to collect automatically [A1, A19, A67, A70][B85, B90] and is often limited by technology limitations of Generative AI tools [A71][B93]. Another primary concern regards how bots communicate with the user, specifically in the conversation design, generation of reply messages, and design of the chatbot's graphical user interface [A1, A3, A25, A39, A64, A66–A68][B85, B89, B92]. Moreover, most practitioners struggle to **provide straightforward and satisfying user interfaces**, and researchers underline the **lack of guidelines and standards for bots usability** [A1, A3, A25, A39, A64, A66–A68][B81, B85, B89, B92]. The last challenge is related to the use of Generative AI-powered chatbots; this technology is relatively recent and, like many other disruptive technologies, is at the point of maximum expectation and illusion about its actual use by practitioners based on the Gartner hype cycle. However, the technology still needs to be updated to meet user demands, specifically regarding software engineering professionals. This aspect causes **disillusionment and frustration because of unsatisfied expectations** when interacting with such types of chatbots [B86].

Natural Language Understanding and Processing. As already described in Section 2, conversational agents' interactions are based and modeled on the concept of *intent*, i.e., a conceptual representation of a possible user request characterized by a set of mandatory information to fulfill the request (called *entities*). Their correct interpretation is crucial for the bot operation and is often based on text interpretation using natural language processing (NLP) and understanding (NLU) algorithms. Therefore, **training NLP and NLU machine learning models** represented the most mentioned challenge [A3, A39, A61, A68, A69][B81, B86, B91, B94]; the main problem lies on the fact that a conversational agent works in singular contexts, and it is difficult to (1) extract sentences to train models and (2) reuse sentences from other contexts. operation [A1, A39][B81, B85, B94, B95]. **skeptical of cause of past experiences with bots exposing malicious intents** For example, practitioners tend to be **skeptical because of past experiences with bots exposing malicious intents**—e.g., theft of sensitive information or insertion of malicious code. So, interacting with these bots can be challenging because their identification is not always an

easy task [A1]. In addition, another challenge is related to a well-known topic such *fairness* [155], intended as the equitable and just treatment—without discrimination—assumed by software systems—particularly in the context of artificial intelligence. Specifically, conversational agents are based on trained machine learning models using datasets containing sentences and assertions by potential users, thus possibly leading the bot to be biased given the different cultures and backgrounds of the individuals who built the dataset, which could **lead to unexpected and often discriminatory behaviors** [A1][B81, B94, B95]. Last but not least, the literature highlighted the phenomena known as *Uncanny Valley*—i.e., the phenomenon whereby a computer behaving like a human being evokes a sense of unease and revulsion in the person interacting with it [129, 140]. The phenomenon is extremely diffused, especially in the interaction with conversational agents. It is related to the humanization of robots: people tend to exhibit unpleasant feelings—such as revulsion and uneasiness comparable to perturbation—when interacting with machines with overly "human" attitudes. Managing this phenomenon so that **the bots turn out to be human enough to improve communication without being disturbing** is a difficult challenge [A1, A39][B85].

4.2.2 Adoption Challenges. Several studies reported challenges in adopting bots and chatbots for development reasons. We found 19 studies: 12 from the white literature [A1, A4, A5, A20, A22, A25, A30, A54, A60, A72–A74] and 7 from the grey one [B81, B84–B86, B92, B94, B95] that report on such a category (≈ 17.76% of the entire dataset).

Motivation and Trust. According to various studies, the first challenge involves **motivating practitioners to use bots** [A1, A5, A25, A74][B84–B86, B92]. In fact, various practitioners are skeptical of bots because of bias—they think they can implement malicious behaviors or are useless and noisy, underlining a **lack of trust** [A5, A20, A25, A30, A54, A60, A72–A74][B84–B86, B92] of developers in bots' recommendations.Furthermore, **the problem of trust becomes harder when dealing with Generative AI-powered bots**. In fact, such AI is notoriously subject to hacking actions performed by malicious persons and aimed at influencing their behaviors [B94, B95] by introducing bugs or defects in the source code [B86]. Furthermore, the risk of unexpected privacy violations [B86] and the lack of transparency in personal data management [B81, B94] constitute a severe limit to the adoption of Generative AI-powered bots.

Discoverability and Customization. Furthermore, **identifying the correct bot to solve a problem is challenging** [A1, A22] for two reasons: first, there are limited search mechanisms to identify appropriate bots given a problem; second, the lack of a unified standard to characterize bots leads to difficulties in their description and categorization. This becomes even more problematic when practitioners identify the correct bot but have **difficulty configuring it** [A1, A4, A30][B86]. In fact, such systems often have few configuration options, even if they have to be used in significantly different contexts.

Terms of Services. The adoption of bots can be **hindered by the specific terms of service of the platforms they are intended to operate on**, which may restrict their functionalities [B84]. This legal and regulatory environment can severely limit the actions that bots are allowed to perform, thereby reducing their effectiveness and potential benefits. As a result, practitioners often hesitate to integrate bots into their workflows due to concerns over compliance and the potential legal ramifications of bot activities that violate these terms. This fear of inadvertently breaching platform policies can lead to a reluctance to adopt bot technology despite its potential to enhance productivity and efficiency.

4.2.3 Development Challenges. The last category of identified challenges concerns with the development of bots. Specifically, we found 19 studies, 13 from the white literature [A1–A3, A26, A39, A48, A61, A64, A68, A72, A75–A77] and 6 from the grey one [B90, B96–B100] that report on such a category (≈ 17.76% of the entire dataset).

Framework, Architecture, and Integration. First of all, various actors have proposed various frameworks to support developers in the development of bots and chatbots (e.g., Microsoft Azure Bot Framework, Google DialogFlow, Amazon Lex, and Rasa). Although such variety can provide practitioners with a large plethora of possibilities to reach their objectives, the literature identifies a **difficulty in identifying the correct framework to start development** [A1, A48, A64, A68, A75, A77][B96–B99]. Such a challenge arises for three main reasons. First, the enormous plethora of frameworks that implement the same functionalities makes it hard to select the better one. Second, the lack of readable documentation for each framework ulteriorly hardens the decision process. Third, **bots are complex systems to design from an architectural point of view** [A3, A48, A72, A77][B100] because, to improve modularity, engineers push to adopt client-server architectures in which the core functionalities are implemented on the server and the interaction with the user on the client side. This, in addition to adding complexity due to the management of communication protocols between the various components (e.g., HTTPS, API Call, and Service Calls), often makes it necessary to use different technologies for the same system, and **not all bot frameworks allow easy integration with them** [A48, A64, A68]. Furthermore, all the previously mentioned problems become harder when practitioners deals with AI-powered bots; in this type of bot, in addition to the integration of the already numerous technologies involved, it is necessary to integrate artificial intelligence systems in a way that meets the required nonfunctional requirements [A48].

Testing. Furthermore, despite such a large amount of frameworks, **a significant minority support developers in testing activities of bots and CAs** [A3, A26, A64, A76][B90]. This is a significant problem because the lack of testing for such a system contributes to the mistrust of practitioners when they have to use them. Underlying this lack is the difficulty in automating the interactions between a human user and the bot. Furthermore, the large number of frameworks for bot development makes it difficult to provide a single tool capable of testing for all.

Maintenance. Such a challenge in terms of testing leads to the emergence of another one: **the maintenance—and consequent evolution—of bots is challenging** [A1, A2, A26]. In fact, the lack of tests and the difficulty in monitoring such systems can impact maintenance activities, making it hard to identify and fix a bug.

Training. Training AI-powered bots for natural language understanding and processing is a complex endeavor, especially within the specific context of software engineering. Obtaining suitable datasets for training these tools is a significant challenge [A3, A39, A61, A68, A69]. Not only are these datasets scarce, but they are also particularly difficult to acquire when tailored to the nuanced requirements of software engineering [A48, A61]. This scarcity and specificity of data lead to **considerable difficulties in effectively training AI modules to perform their intended tasks**.

4.3 RQ_3: On the best practices to use bots and conversational agents for software engineering purposes

Besides collecting limitations and challenges for software bots, we were also interested in the possible best practices to overcome them. For such a purpose, we formulated our third research question to extract a list of best practices from the formal and grey literature to help practitioners adopt and develop bots for engineering purposes.

We decided to divide the identified best practices into two categories based on the approach's scope. In the following, we elaborated on each best practice individually.

4.3.1 Development and Design Best Practices. As already reported in Section 4.2.3, the large number of available technologies and the need for standardized methodologies make developing and designing bots challenging. In the following, we report the best practices identified by researchers and practitioners that can facilitate the implementation of bots and conversational systems (Appendix B—Table 1 [133]).

Follow a modular architecture. Despite their significant heterogeneity, bots—particularly conversational agents—are characterized by the need to work similarly on different platforms. For example, CAs should be used on different communication platforms—e.g., Slack and Discord—but still implement the same functionalities. In these situations, a client-service architecture—or derivated ones—represents the best choice [A2, A65]. Srivastava and Prabhakar [A3] defined a reference architecture for conversational agents which specifies that these systems should be divided into two macro-components: (1) *Conversational Subsystem*, which implements the input analysis, the extraction of user intent, and the response; (2) *Business Component*, which contains the logic to fulfill user intents and is implemented like an API. Another possible architecture [B100] can organize chatbots into five modules:

(1) *Environment* represents the core Natural Learning Process (NLP) engine and context interpretation.
(2) *Question and Answer System* is where the system interprets the question and responds with relevant answers from the knowledge base.
(3) *Plugins* is the module that uses API calls to interrogate other systems for executing bot operations.
(4) *Node Server* is the module that handles the traffic requests from users and routes them to appropriate components.
(5) *Front-end Systems* are the possible client-facing platforms users can use to interact with the system.

Make bots adaptive and able to learn over time using AI. Software bots, specifically in the figure of CAs, need to communicate with a large plethora of users characterized by different cultural and language characteristics. Therefore, in most cases, such interaction and the bot's user understanding rely on an NLP module–that uses datasets composed of possible phrases and assertions in different languages–to interpret the user's intent. However, the dataset employed should increase over time based on the user's usage. For this reason, implementing a machine learning system able to learn over time and modify the dataset accordingly to new conditions is needed—namely, *active learning systems* [148]—becomes necessary [B85, B89, B90, B94].

Adopt a specific lifecycle process for bots. Bot are complex systems to design, both from a technical and logical side, especially if integrated with a conversational module, i.e., like for CAs. Consequently, practitioners focused their attention on defining rigorous steps to orchestrate these systems from a development and maintenance perspective [B101–B103]. In the following, we reported the elicited steps for conversational agents development:

(1) clearly defines the process to be automated or supported by the CA;
(2) catch the target audience's needs by engaging with software developers through interviews and usability testing sessions to identify common challenges they face during the development cycle;
(3) use existing sources to analyze competitors by conducting surveys of similar tools already existing or interviewing the potential audience;
(4) define audience intents using use cases and journey maps after conducting individual interviews with the audience;
(5) design the interaction workflow for each intent by using modeling languages, e.g., UML;
(6) choose technologies for the CA development and operation platforms according to company needs;

(7) design the CA architecture to be modular as much as possible and adherent to the guidelines proposed by researchers [A2, A3, A65][B100];
(8) develop a measurement framework to evaluate the CA by establishing metrics such as reduction in time to fix bugs, frequency of use, and developer satisfaction to assess the impact of the CA on the development process [A35, A49, A78][B85, B100];
(9) implement an artificial intelligence (NLP, NLU, and active learning) module capable of understanding technical language specific to software development, using, for example, libraries that can parse and make sense of code snippets and technical queries [A61];
(10) implement the CA application and interaction logic in a way that can handle diverse inputs from developers, such as textual descriptions, code blocks, or even spoken commands, and provide appropriate responses or actions [A35, A54, A60, A67, A79, A134][B85, B101–B103];
(11) deploy the CA and adopt a continuous improvement process.

Carefully select the correct framework based on bot requirements. As mentioned in Section 4.2.3, the increasing adoption of bots and conversational agents in software development leads diverse actors to develop frameworks for supporting their development. Some examples are Azure Bot from Microsoft—able to provide a single environment to develop bots for different communication channels—or Rasa—an open-source solution to develop bots and train NLP models all in the same place. Nevertheless, determining which framework best fits a project's requirements is far from easy [A1, A64, A68, A69, A75][B96–B99]. To address this challenge, practitioners identified criteria and key motivations that should guide while selecting the development framework, i.e., core technology, the target platform of operation, and the distinctive characteristics [B96–B99]. We reported the main framework with associated motivations in Appendix B—Table 2 [133].

Adopt Domain Specific Language Models to develop bots. Developing bots requires expertise in several areas, such as software engineering and natural language processing. Although frameworks and platforms simplify the implementation of CA, there is still a significant obstacle caused by the inability of such frameworks to support all the developers' desires. Some studies suggest approaches based on domain-specific languages (DSLs) for model-driven development, reducing the workload of developers and designers [A77]. Beyond that, such studies indicate that there are already bots developed to support the developer in building models that can then be used to develop systems—e.g., recommendation systems—in an automated manner [A11].

Make the bot able to save and use contextual information using RAG and Vector DBs. Maintaining contextual information is critical for adopting bots in software engineering [A1, A19, A67, A70][B85, B90]. Integrating Retrieval-Augmented Generation (RAG) models and Vector Databases can address this challenge [A71, A79][B93]. RAG models combine retrieval and generation capabilities to access and use context-specific information effectively. Vector Databases store and query contextual data embeddings, enabling precise and context-aware responses. This synergy ensures chatbots provide more meaningful and context-rich conversations, enhancing the overall user experience.

Conduct Wizard of the Oz studies to create datasets for NL modules. Wizard of Oz studies create datasets for training NLP and NLU models for developer-support bots [A61]. Human operators simulate AI responses, allowing researchers to collect nuanced language data from developers. This method captures realistic insights into the support developers need and the queries they pose. Despite the complexity of these studies, collecting extensive data is essential for developing effective models [B94].

4.3.2 Interaction and Adoption Best Practices. Interaction for bots is an important—if not the most important—aspect to take care of during their development and design. For such a reason, it is not surprising that the state of the art focused more on best practices to improve how bots communicate with users (Appendix B—Table 3 [133]).

Integrate bots actions in a way that does not interrupt developer workflow. Bots are naturally designed to automate some processes—or part of them—to allow users to waste less time and focus on other tasks [150]. Such automation comes with no problems: first, bots' actions should be secure to allow users to trust them; second, bots should not interrupt users in their operation, or the advantage of the automation loses its value and usefulness [A1, A5, A25, A43, A45][B84, B85]. For example, many users found tool-recommender-bot interrupted existing processes—most notably by breaking software builds—and were discouraged from merging pull requests from our system [A54]. For the reasons mentioned above, software bots should be designed so that the results of their actions should not slow down or hinder the user's actions, at least not unexpectedly and covertly [A25, A30, A35, A54, A60][B85, B101–B103]. In recent years, orchestrator bots, i.e., bots designed to manage and guide the actions of other bots, have emerged as an efficient way to integrate a large number of bots easily without interrupting the practitioner's workflow [A43, A45].

Allow developers to edit bot configuration easily. As mentioned in Section 4.2.2, configuring bots to allow their operation in different contexts is a crucial challenge [A1, A4]—both for their adoption and development. Therefore, developing the system providing users with easy ways to configure it—e.g., editable configuration files, graphical interfaces for configuration, and exhaustive documentation—could be extremely useful in increasing bot adoption [A4, A16, A25, A30, A35]. Moreover, Erlenhov et al. [A25] demonstrated that such a best practice could mitigate the *Interruption and Noise problem* [A1]—elaborated in Section 4.2.1—through a design or configuration that is mindful of whom to notify of which events.

Provide bots with personality. User perception is essential for bots. Various studies [A5, A60, A134] and grey literature [B101–B103] reported that this perception can be highly influenced by the bot's language register—as well as by its name and graphical elements. Specifically, ensuring that the bot is recognizable and distinguishable, i.e., providing it with a distinctive *personality*, can help in its adoption and use [A60, A134]. Therefore, the bot's language should be casual, accessible, and fun; in a way that is not dull but friendly. Nevertheless, it should be paid attention to not trigger the *unchanny valley* effect, i.e., the phenomenon whereby a computer bears a near-identical resemblance to a human being and evokes a sense of unease or aversion in the person interacting with it [129, 140].

Design a Smooth and frictionless interaction. Various works identified and analyzed the phenomena of "noise"—i.e., annoying bot behaviors that lead to deterioration of communication in teams [156]. Consequently, designing a smooth and frictionless interaction becomes mandatory, and this could be achieved by carefully planning conversational flows, especially for conversational bots. This can be achieved by (1) allowing bots to detect dead ends in conversations and prompt users by giving hints on how to continue the interaction and (2) implementing mechanisms to provide continuous feedback to the user that the bot is listening [A35, A54, A60, A67, A134][B85, B101–B103]. Moreover, designing bots to be proactive—maybe integrating Generative AI in them—is a successful way to please practitioners and make their interactions smooth [A79]. However, this can be achievable if the natural language comprehension of the bot is well enough designed and developed [A67]. Moreover, designers should evaluate the use of graphical user interface elements—e.g., cards, buttons, and boxes—to facilitate the interaction and make the conversation more efficient [A134].

Enforce transparency in bot actions and outputs. Trust is a vital aspect of adopting bots for software development since such systems are often designed to automate product development and release operations, which are considered vital to the success of a software project. The first way to improve it is related to the *uncanny valley* effect: when users interact with a machine that seems too human to them, they begin feeling a sense of unease and revulsion [129, 140]. To avoid such a situation, developers should ensure that the bot's software nature is always apparent and that situations perceived as ambiguous by the user are never created [A5, A30, A35, A134][B85]. Moreover, it should always be evident when the user is asked to take action and why the request is made. For example, implementing graphical elements that are activated only at the moment when the user has to give input can be an excellent way to request an action overt [A5, A134][B85].

Integrating concepts from behavioral science in interaction flow. As already said, *recommendation bots*—namely, bots designed to provide user tips and recommendations—are largely adopted for software development purposes. Nevertheless, such bots' recommendations seem to be ineffective if the interaction with the users violates social contexts within software engineering and interrupts the development workflow of programmers. For This reason, Brown and Parning [A72] proposed adopting behavioral science concepts to design and guide the interaction flow of conversational systems—e.g., *nudging* [153] and *choice architecture* [154]. Notably, elaborated in their work, they use 11 practical tools—provided by Johnson et al. [126]—to integrate *choice architecture* [154]—the way options are organized and presented to humans—in *recommendation bots* and consequentially improve the recommendation process. From a practical point of view, bots should be designed according to three interaction principles:

- *Actionability*. It refers to the ease with which users can act on recommendations. To improve developers' perception of recommendations, better choices should be easier to take and implement.
- *Feedback*. It refers to the way pieces of information are provided to the user. To improve developers' perception of recommendations, useful and essential information should be provided to the users.
- *Locality*. It refers to the setting—both physical and temporal—surrounding the context of developer recommendations. To improve developers' perception of recommendations, they should be presented during their working hours and when they are in the workplace.

Moreover, a study performed by Robe et al. [A60] demonstrated that practitioners are more inclined to adopt and use bots that demonstrate adaptability, motivation, and social presence skills; such characteristics support developers' trust.

Provide users with a high level of control over bot. Another way to improve interaction is through *nudges*—i.e., any aspect of the choice architecture that alters people's behavior in a predictable way without forbidding any options or significantly changing their economic incentives [153]. As a way to use nudge for conversational systems, Brown et al. [A54] reported that making easy and cheap recommendations—instead of forcing updates and changes to developers—could improve the users' perception of recommendation bots [A16, A19, A25, A35, A54, A67][B85]. For example, in the context of a bot that analyzes the source code to find quality problems, making comments above the incriminated line of code instead of automatically refactoring it could be a better choice. Another way to provide a sense of control to the user—as proposed and reported by Erlenhov et al. [A25]—is to make the bot ask for confirmation before starting to work on a task or at the end of interactions—in a *transaction-fashion*—to avoid misunderstanding.

Create a reliable test infrastructure. An alternative way to make developers trust a development bot is to provide it with a reliable and public test infrastructure [A25][B90]. In such a way, the potential audience of the tool can evaluate it in advance and feel more comfortable with it. Nevertheless, testing bots can be challenging due to the need to simulate

all possible user interactions [A3, A26, A64, A76][B90]. In order to facilitate such activity, practitioners propose to (1) use one of the provided frameworks for bots—and chatbots—testing—e.g., testYourBot[10] and Dimon[11]—and (2) develop and perform manual testing of bots based on their interaction flow [B90].

Integrate a feedback system allowing users to evaluate bot choices. Most problems that characterize software bots regard how systems interact with users. In most cases, these issues could be fixed by asking users what they want and what the specific problem is; therefore, implementing a system for practitioners to provide feedback and evaluate bots' actions is highly recommended. Although an automatic approach based on heuristics would be the most appropriate solution—but not the easiest one to implement—in which the user is directly asked to provide opinions can also reach the goal [A35, A49, A78][B85, B100].

Adapt the bots to the privacy policies of organizations. To enhance trust in bots using generative AI, developers should implement adapters at the bot interfaces. These adapters control the bot's access to information and responses, aligning operations with privacy policies [B94] and ensuring confidentiality and user consent. Configured to limit sensitive data collection, adapters reduce data breach risks and support data minimization [B94]. This integration enhances compliance with privacy regulations and bolsters trust in the bot's application.

4.4 RQ_4: On the benefits of bots and conversational agents for software engineering purposes

In this RQ, we aimed to identify the benefits for development teams of adopting bots for software engineering purposes. Specifically, our goal was to determine which areas are most impacted by bots to identify possible fields yet to be explored and propose future steps for research to deepen (Appendix B—Table 4 [133]).

4.4.1 Productivity Benefits. Most of the benefits provided by bots regard the productivity aspects of software development. This is because bots have been designed initially **to automate processes considered unimportant and tedious for practitioners** [A8, A27][B104–B106]. For this reason, it does not look surprising that the most positive impact of their adoption—recorded by researchers and practitioners—is related to the productivity.

Furthermore, bots can **help developers complete tasks related to meaningful goals for the project** [A6, A8, A22, A35, A40, A42, A45, A134][B87, B88]. Such a benefit is defined by Storey et al. [A22] under the title of *effectiveness* and can be achieved in various ways. For example, by automatically capturing data and disseminating them to the entire team, bots can increase the developers' awareness of the project situation [A22, A134]. An ulterior way to improve developers' effectiveness is through conversation moderation—i.e., providing information during meetings or bringing the discussion back into focus when participants tend to digress [A6].

Bots can also impact productivity by **helping developers complete tasks more quickly** [A6, A8, A13, A15, A19, A22, A27, A35, A42, A44, A45, A47, A51, A52, A57][B86, B104, B105]—this is identified by *efficiency* [A22]. Such a benefit can be reached in many ways: (1) bots can spare developers from reading large amounts of documentation by understanding what they search for and providing it immediately [A22, A52]; (2) bots can support during source code writing by providing code snippets or tips [A15, A51][B105]; (3) bots can also help in documentation phases, by collecting pieces of information from different sources and summary them [A6][B104].

In practice, bots are also used to perform simple tasks, e.g., building a new version of a software project, executing test suites, and preparing execution reports, thus implying an increase in workers' free time and optimizing the execution

[10] testYourBot site: https://shankar96.github.io/testYourBot/
[11] Dimon site: http://dimon.co/

of these tasks. This is because bots **can work without interruption** [A25][B82, B87, B88, B105] and **handle large amounts of tasks simultaneously** [A25][B86–B88, B104, B105].

4.4.2 Collaboration Benefits. Bots also provide benefits when comes to knowledge sharing and collaboration in development teams [A6, A8, A37, A44, A46, A134][B86, B87, B104]. For example, if correctly used, bots can distribute to the team the status of other tasks, consequently **building team trust and cooperation** [A134]. Furthermore, bots can enhance the recording of decisions, performing: (1) automatic fulfillment of artifacts templates (e.g., user stories), (2) summary build from discussions, (3) automatic generation of diagrams, and (4) monitoring of conversations to automatically refer previous decisions. Moreover, bots can enhance developer collaboration by performing (1) recording and retrieving technical discussions, (2) fetching and updating models based on conversation, and (3) parsing committed messages. By doing all of these, bots can **support the decision-making process** of different figures involved in software engineering [A43][B86].

Furthermore, by abstracting practitioners from executing repetitive and tedious tasks, there is **less chance of mistakes** [B104–B107]. This is since (1) humans tend to commit errors if constantly repeating the same task, and (2) software can apply quality standards pre-defined.

Moreover, such an abstraction can be used by managers to improve the risk management activity [B104, B105, B107] and **facilitate the adoption of new methodologies and technologies** in their team [B104, B106]. For example, by using a chatbot to improve code review, managers could provide developers with easy tips and good practices without assigning human effort to such a purpose.

4.4.3 Technical Benefits. Lastly, bots and conversational agents can also provide many benefits from a technical and product view. For example, bots can **improve the maintainability and quality of source code** by detecting technical debt—e.g., code smells and test smells—or by enforcing good programming practices [A8, A15–A19, A25, A27, A51][B82, B104–B107]. Moreover, bots can collect information from different artifacts—e.g., source code, documentation, and reports—to spread to the entire team and **reduce socio-technical problems**, e.g., turnover. Furthermore—it is always related to the knowledge-sharing side—bots can also be used to collect information about API and new technologies and provide them "on demand" to the users. Such communication can occur in any communication channel used by the development team [A8, A25, A31, A50, A56, A57, A134][B82, B86, B106].

5 DISCUSSION, IMPLICATIONS, AND FUTURE STEPS

The results of our study pointed out several observations worth further discussion. We started from them to identify potential future research lines that the research community can explore. Moreover, since the multivocal literature review also comprises gray literature, practitioners can benefit from our observations. In the following paragraphs, we report the observations and implications from our review, organized by topic.

5.1 Conversational Agents and Human Aspects of Software Development

Conversational Agents (CAs) are *de facto* software systems in which interaction with users represents the main characteristic, thus implying that making the potential audience more involved during design phases is crucial to project success [A1, A3, A19, A25, A39, A63, A64, A66–A68, A134][B85, B89–B92]. Moreover, since most of the reported software bots operate with different stakeholders simultaneously, the interaction patterns between different actors also acquire importance. For this reason, representing collaboration and communication patterns in software communities in a quantitative way could help facilitate the CAs' interaction workflow design. In order to operationalize such

collaboration patterns, state of the art defined *community smells* [152]—i.e., a set of anti-patterns in collaboration activities in software development communities that are precursors of socio-technical problems. Based on the above consideration, we can provide two future fields of exploration:

- Community Smells to support bots design. Community Smells could be used as input when designing bot to improve the communication and increase the trust that users have with them. Researchers could study how this aspect impacts the development process.
- Bots-Community Smells. Community Smells represent collaboration and communication anti-patterns in software communities. Adopting bots for software development purposes could lead to the creation of a community over the community, characterized by the interaction between bots and developers and bots with other bots. This could lead to the emergence of a new way to represent patterns—and anti-patterns—through a particular type of community smells. The research community could (1) quantitatively represent them and (2) provide practitioners with instruments to improve their relationship with conversational systems.

5.2 Conversational Agents for Training and Support

Besides practical uses, the adoption of bots also revolves around knowledge sharing [A22, A24, A25, A31, A41, A49, A50, A52, A53, A56–A58][B82, B85, B87, B88]. Specifically, practitioners use bots to (1) share information between team members and (2) record decision rationale. Moreover, conversational agents are used to provide information on software modules to new developers in order to facilitate training phases. Based on our findings, we can provide the following considerations:

- Bots and Turnover. Bots can be used to collect expert knowledge and provide it to newcomers [B82, B85]. In this way, the consequences of turnover [116]—i.e., the continuous influx and retreat of human resources in a team—can mitigate the progressive loss of knowledge on a software module. From here, researchers should focus on how (i) bots should collect such knowledge and (ii) how this should be provided to newcomers.

5.3 Addressing Limitations with Best Practices

Our literature survey identified challenges and best practices related to bots and conversational agents [156]. We tried to map such challenges to best practices based on the various papers to provide solutions and mitigation strategies supported by evidence in research and practitioners' contexts. We reported our findings in Figure 2.

- Challenges and Best Practices. Our mapping—based on literature evidence—should be validated through qualitative studies and could be a valuable contribution for practitioners facing bot challenges.

5.4 Conversational Agents for Project Management: How far are we?

Project Management is the practice of applying knowledge, skills, methods, and tools to meet the requirements of a specific project, achieving quality and performance goals [124]. In the context of software development, project management is a highly complex and hard-to-standardize activity due to the volatile nature of software and the extreme heterogeneity of stakeholders involved—both in terms of background and competencies [111, 112, 122]. Moreover, according to the Standish Group's CHAOS Report [127], approximately 70% of software projects fail—some of them with catastrophic consequences [117, 123, 128] and state of the art demonstrated that management aspects significantly impact project outcomes, leading management activities as a priority to consider [110, 113]. Software bots—and particularly Conversational Agents (CAs)—could help improve and support management activities:

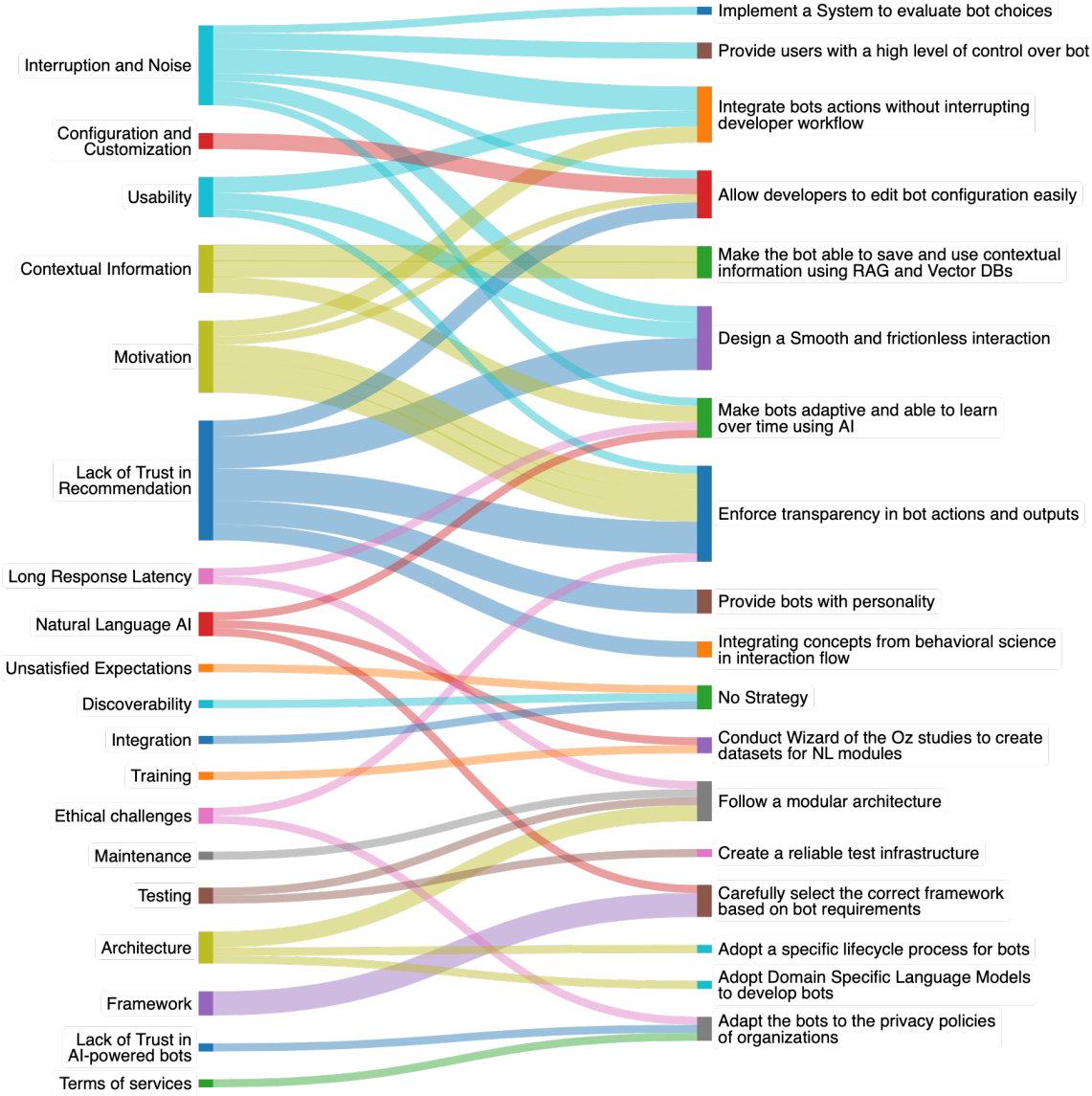

Fig. 2. Challenges (on the right) and Associated Best Practices (on the left).

👉 CAs as Managers' Twins. Project managers (PM) are responsible for ensuring that the project meets the requirements. In order to achieve this objective, managers should have various skills and—most critically—manage various factors simultaneously [124]. To support this goal, having a community of bots collecting different metrics and monitoring different activities could support PMs in their daily tasks [A34]. Moreover, making a hierarchical bots structure in which pieces of information are filtered and summarized differently at each level of the organization could help managers focus on essential tasks, resulting in increased productivity and mental health quality [A43].

- CAs as Meeting Facilitator. There are tools designed to help practitioners manage information sharing during meetings [A6, A8, A37][B85]. Since software engineering is *de facto* a social activity involving different stakeholders with different backgrounds, having such support could greatly benefit the state of practice. However, the reported effectiveness of these tools could be better. Therefore, investing in improving their capabilities and using Generative AI could be a worthy future research topic.

5.5 Open Issues for bots and conversational agents

Despite the significant attention reserved to the topic by researchers and practitioners, using bots for software development is still a young and unexplored field. For such a reason, there are a series of open issues and exploration opportunities that the research community should care of:

- Usability Research for CAs. Although CAs put the interaction with users at their center, to the best of our knowledge, there needs to be more research on the usability of such systems. Moreover, practitioners should put more effort into defining standards and guidelines to design interaction workflows in conversational systems.
- Testing of Bots. Testing bots and conversational agents arose as a complex topic [A3, A26, A64, A76][B90]. Most of the testing strategies for these systems fall under the category of system testing, and only a few have been proposed at lower levels of testing (e.g., unit and integration).
- Behavioral Science and Bots. Motivating users to adopt bots is a widely documented problem [A1, A5, A25][B84, B85, B92]. Various works proposed adopting behavioral science concepts—e.g., nudge and interaction principles—to improve the design phases of these systems [A72]. For this aim, researchers from both fields—computer science and social science—could join forces to open new research opportunities.
- Interruption and Noise: an Open Challenge. Interruption and noise are significant and documented problems for conversational agents and bots [A1, A25, A63, A134][B85]. Despite the work already done to propose mitigation strategies for such phenomena, more should be done to increase the adoption and effectiveness—in terms of accomplished contribution—of bots for development purposes.
- Generative AI for Software Development. The recent emergence of large language models in the consumer market has inevitably led software engineers to experiment with integrating these tools into their workflow [A9, A13]. When such LLMs are used in their chatbot versions (e.g., ChatGPT), they constitute software bots. Despite their impressive capabilities and the potential they could have in all aspects of software development, some clear limitations continue to exist, likely due to the technology's still early stage.

6 LIMITATIONS AND THREATS

Threats to the validity of an MLR review are typically related to the inaccuracy of data extraction and incomplete set of studies due to search terms limitation, selection of academic search engines, and researcher bias regarding exclusion/inclusion criteria. In this section, we identified and carefully addressed potential threats to the validity [131, 157] of our study by taking steps to minimize or mitigate them.

Internal Validity. Threats in this category regard the source and data selection approach, as well as the possibility that the findings are affected by casualty factors without the researcher's knowledge [157]. We relied on a previous systematic approach proposed in the literature [131] and described in Section 3. We defined and reported each step of the process carefully—i.e., search engines, search terms, and inclusion/exclusion criteria—to ensure replicability. These are usually the problematic steps. As for the search terms used for studies extraction, we relied not only on the most

common words—e.g., bot—but also on similar terms used by different authors for pointing to a similar concept—e.g., conversational agents. We followed a formal search using defined keywords, followed by a manual search using the references of the initial pool in our field of study. As for the search engine, we included official comprehensive academic databases, e.g., Scopus, but also fewer ones like Google Scholar. While for the grey literature, we relied on Google Search Engine—like other Grey Literature and Multivocal Literature Reviews [121, 125]. Therefore, we believe that we collected all the relevant studies in the field. Finally, as for the inclusion/exclusion criteria, they can usually suffer from researchers' judgment and experience. For this reason, all the paper's authors were involved in checking criteria by applying joint voting for study selection. As a result, only those with high scores were selected for this study.

Construct Validity. Threats in this category regard the extent to which the object of study truly represents the theory behind the study and are mainly due to imprecision in performed measurements [157]. One threat in this category concerns the lack of empirical evidence in the studies coming from grey literature. Indeed, this literature reports facts based on opinion or experience. Since the grey literature represents the voice of the practitioners, we could assume that they would be speaking from personal experience. This can be a crucial limitation; however, the goal of an MLR is also to show different ways of thinking, i.e., empirical versus practical, since it can have two types of audience: academic and industrial. Moreover, the inclusion of grey literature can help to overcome the scarcity of studies retrieved by automatic searches in the digital databases of scientific literature.

Conclusion Validity. Threats in this category concern the degree to which the conclusion we reach is credible or believable [157]. The first two authors of the paper reviewed each step of the process to ensure the reliability of the study and mitigate bias in data selection and extraction. Each severe disagreement between authors was resolved by consensus among all the authors. Furthermore, in order to strengthen our results, we followed both the guidelines from Kitchenham et al. [130]—related to the formal literature selection—and from Garousi et al. [120]—associated with the grey literature inclusion. Moreover, we followed a rigid quality criteria assessment—both for white and grey literature—and we extracted our sources from the most used and known databases.

External Validity. Threats in this category concern the extent to which our study's results can be generalized [157]. First of all, we conducted our MLR on the most known and suggested databases of literature [120, 130] to increase the quality of extracted sources and generalizability of our findings. Furthermore, we extracted only primary sources written in English, so the rest in other languages were excluded, thus possibly threatening our generalizability. Nonetheless, (i) the top venues in software engineering accept only papers written in English, and (ii) most of the literature reviews already present in the literature include exclusion criteria based on the English language.

7 CONCLUSION

In this work, we reported on a multivocal literature review to shed light on adopting bots and conversational agents for software engineering purposes. The reason behind the choice of a multivocal review is related to the fact that bots and CAs are extremely linked to the practitioners' field, so we wanted to contribute both to researchers and practitioners. Specifically, we aimed to provide a representation and categorization of the (1) motivation for adopting bots, (2) challenges arising from their integration into software development processes, (3) best practices applicable contrasting the before-mentioned challenges, and (4) benefits derived from the adoption of bots. Moreover, we also provide a list of discussion points and opportunities for researchers to encourage new investigations on the matter.

Indeed, our contributions could be considered a fundamental base built to advance the state of the art and provide an advantage both for researchers and practitioners.

ACKNOWLEDGMENTS

The authors would like to thank the two anonymous Reviewers for their insightful comments on our paper.

MLR—FORMAL LITERATURE

[A1] Mairieli Wessel, Igor Wiese, Igor Steinmacher, and Marco A. Gerosa. Don't disturb me: Challenges of interacting with software bots on open source software projects, 2021.

[A2] Shayan Zamanirad, Boualem Benatallah, Moshe Chai Barukh, Fabio Casati, and Carlos Rodriguez. Programming bots by synthesizing natural language expressions into api invocations. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 832–837, 2017.

[A3] Saurabh Srivastava and TV Prabhakar. A reference architecture for applications with conversational components. In *2019 IEEE 10th International Conference on Software Engineering and Service Science (ICSESS)*, pages 1–5. IEEE, 2019.

[A4] Rijnard van Tonder and Claire Le Goues. Towards s/engineer/bot: Principles for program repair bots. In *2019 IEEE/ACM 1st International Workshop on Bots in Software Engineering (BotSE)*, pages 43–47. IEEE, 2019.

[A5] Martin Monperrus. Explainable software bot contributions: Case study of automated bug fixes. In *2019 IEEE/ACM 1st International Workshop on Bots in Software Engineering (BotSE)*, pages 12–15, May 2019.

[A6] Fabian Gilson and Danny Weyns. When natural language processing jumps into collaborative software engineering. In *2019 IEEE International Conference on Software Architecture Companion (ICSA-C)*, pages 238–241, March 2019.

[A7] Ranci Ren, John W. Castro, Adrián Santos, Sara Pérez-Soler, Silvia T. Acuña, and Juan de Lara. Collaborative modelling: Chatbots or on-line tools? an experimental study. In *Proceedings of the Evaluation and Assessment in Software Engineering*, EASE '20, page 260–269, New York, NY, USA, 2020. Association for Computing Machinery.

[A8] Fabian Gilson, Sam Annand, and Jack Steel. Recording software design decisions on the fly. In *SEED/NLPaSE@ APSEC*, pages 53–66, 2020.

[A9] Justine Winata Purwoko, Tegar Abdullah, Budiman Wijaya, Alexander Agung Santoso Gunawan, and Karen Etania Saputra. Analysis chatgpt potential: Transforming software development with ai chat bots. In *2023 International Conference on Networking, Electrical Engineering, Computer Science, and Technology (IConNECT)*, pages 36–41. IEEE, 2023.

[A10] Rijul Saini, Gunter Mussbacher, Jin LC Guo, and Jörg Kienzle. Automated, interactive, and traceable domain modelling empowered by artificial intelligence. *Software and Systems Modeling*, 21(3):1015–1045, 2022.

[A11] Lissette Almonte, Sara Pérez-Soler, Esther Guerra, Iván Cantador, and Juan de Lara. Automating the synthesis of recommender systems for modelling languages. In *Proceedings of the 14th ACM SIGPLAN International Conference on Software Language Engineering*, pages 22–35, 2021.

[A12] Ilham Qasse, Shailesh Mishra, Björn Þór Jónsson, Foutse Khomh, and Mohammad Hamdaqa. Chat2code: A chatbot for model specification and code generation, the case of smart contracts. In *2023 IEEE International Conference on Software Services Engineering (SSE)*, pages 50–60. IEEE, 2023.

[A13] Gian Luca Scoccia. Exploring early adopters' perceptions of chatgpt as a code generation tool. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW)*, pages 88–93. IEEE, 2023.

[A14] Sara Pérez-Soler, Esther Guerra, and Juan de Lara. Flexible modelling using conversational agents. In *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, pages 478–482, Sep. 2019.

[A15] Marvin Wyrich and Justus Bogner. Towards an autonomous bot for automatic source code refactoring. In *2019 IEEE/ACM 1st International Workshop on Bots in Software Engineering (BotSE)*, pages 24–28. IEEE, 2019.

[A16] Vahid Alizadeh, Mohamed Amine Ouali, Marouane Kessentini, and Meriem Chater. Refbot: intelligent software refactoring bot. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 823–834. IEEE, 2019.

[A17] Zhewei Hu and Edward F. Gehringer. Improving feedback on github pull requests: A bots approach. In *2019 IEEE Frontiers in Education Conference (FIE)*, pages 1–9, Oct 2019.

[A18] Antônio Carvalho, Welder Luz, Diego Marcílio, Rodrigo Bonifácio, Gustavo Pinto, and Edna Dias Canedo. C-3pr: A bot for fixing static analysis violations via pull requests. In *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 161–171, Feb 2020.

[A19] Marvin Wyrich., Regina Hebig., Stefan Wagner., and Riccardo Scandariato. Perception and acceptance of an autonomous refactoring bot. In *Proceedings of the 12th International Conference on Agents and Artificial Intelligence - Volume 1: ICAART,*, pages 303–310. INSTICC, SciTePress, 2020.

[A20] Marvin Wyrich, Raoul Ghit, Tobias Haller, and Christian M¨uller. Bots don't mind waiting, do they? comparing the interaction with automatically and manually created pull requests. *arXiv preprint arXiv:2103.03591*, 2021.

[A21] Dragos Serban, Bart Golsteijn, Ralph Holdorp, and Alexander Serebrenik. Saw-bot: Proposing fixes for static analysis warnings with github suggestions. In *2021 IEEE/ACM Third International Workshop on Bots in Software Engineering (BotSE)*, pages 26–30, June 2021.