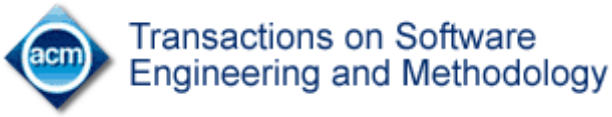


Measuring and Clustering Heterogeneous Chatbot Designs

Journal:	*Transactions on Software Engineering and Methodology*
Manuscript ID	TOSEM-2023-0032
Manuscript Type:	Paper
Date Submitted by the Author:	01-Feb-2023
Complete List of Authors:	Cañizares, Pablo; Universidad Autonoma de Madrid, López-Morales, José María; Universidad Autonoma de Madrid Pérez-Soler, Sara; Universidad Autonoma de Madrid Guerra, Esther; Universidad Autónoma de Madrid, Computer Science de Lara, Juan; Universidad Autónoma de Madrid, Computer Science
Computing Classification Systems:	Software and its engineering → Software design engineering; Extra-functional properties, Computing methodologies → Natural language processing, Additional Key Words and Phrases: chatbot design, metrics, clustering, quality assurance, model-driven engineering

UAM
Universidad Autónoma de Madrid

Escuela Politécnica Superior

Dear editor,

We attach the paper entitled "*Measuring and Clustering Heterogeneous Chatbot Designs*" for consideration in the ACM Transactions on Software Engineering and Methodology journal.

In the paper, we propose a suite of metrics for chatbot designs, as well as two clustering methods that help in grouping chatbots along their conversation topics and design features. The approach is supported by a tool, which we have used to evaluate the metrics and the clustering methods over a set of 259 Dialogflow and Rasa chatbots from open-source repositories. The results open the door to incorporating the metrics within chatbot development processes – for the early detection of quality issues – and to exploit clustering to organise large collections of chatbots into significant groups to ease chatbot comprehension, search, and comparison.

This paper extends our preliminary works [1], [2] as follows. We have refined our suite of metrics and extended it with a new readability metric, expanding the explanations and rationale of the metrics. We have improved our metrics-based clustering method by using principal component analysis. We have also improved our vocabulary-based clustering by applying stemming (which reduces the dimensionality of the chatbot vector representation and improves efficiency), extracting descriptive terms for each chatbot cluster, and calculating the term frequency-inverse document frequency (TF-IDF) of the chatbots' vocabulary (which improves our previous implementation based on bag-of-words by considering the importance of words according to their frequency of appearance). We have performed a thorough, extended evaluation of the metrics on a dataset of 259 chatbots (≈22x more than in [1]). This dataset is available at https://github.com/asym0b/Dataset. Finally, we report on an evaluation of our vocabulary-based clustering method, which is also a new contribution of this paper.

[1] P. C. Cañizares, S. Pérez-Soler, E. Guerra, and J. de Lara, "*Automating the measurement of heterogeneous chatbot designs*". Proc. 37th ACM Symposium On Applied Computing (SAC), 2022, pp. 1–8.

[2] J.-M. López-Morales, P. C. Cañizares, S. Pérez-Soler, E. Guerra, and J. de Lara, "*ASYMOB: a platform for measuring and clustering chatbots*". Proc. 44th 2022 IEEE/ACM ICSE (Companion) 2022, pp. 16-20.

Madrid, February 1st, 2023

Pablo C. Cañizares, José Mª López-Morales, Sara Pérez-Soler, Esther Guerra, Juan de Lara
Modelling and Software Engineering research group (miso)
Computer Science Department
Universidad Autónoma de Madrid (Spain)

UAM
Universidad Autónoma de Madrid

Escuela Politécnica Superior

Differences between conference and journal versions

This paper extends our preliminary works [1], [2] as follows. We have refined our suite of metrics and extended it with a new readability metric, expanding the explanations and rationale of the metrics. We have improved our metrics-based clustering method by using principal component analysis. We have also improved our vocabulary-based clustering by applying stemming (which reduces the dimensionality of the chatbot vector representation and improves efficiency), extracting descriptive terms for each chatbot cluster, and calculating the term frequency-inverse document frequency (TF-IDF) of the chatbots' vocabulary (which improves our previous implementation based on bag-of-words by considering the importance of words according to their frequency of appearance). We have performed a thorough, extended evaluation of the metrics on a dataset of 259 chatbots (≈22x more than in [1]). This dataset is available at https://github.com/asym0b/Dataset. Finally, we report on an evaluation of our vocabulary-based clustering method, which is also a new contribution of this paper.

[1] P. C. Cañizares, S. Pérez-Soler, E. Guerra, and J. de Lara, "*Automating the measurement of heterogeneous chatbot designs*". Proc. 37th ACM Symposium On Applied Computing (SAC), 2022, pp. 1–8.

[2] J.-M. López-Morales, P. C. Cañizares, S. Pérez-Soler, E. Guerra, and J. de Lara, "*ASYMOB: a platform for measuring and clustering chatbots*". Proc. 44th 2022 IEEE/ACM ICSE (Companion) 2022, pp. 16-20.

Measuring and Clustering Heterogeneous Chatbot Designs

PABLO C. CAÑIZARES, Universidad Autónoma de Madrid, Spain
JOSE MARÍA LÓPEZ-MORALES, Universidad Autónoma de Madrid, Spain
SARA PÉREZ-SOLER, Universidad Autónoma de Madrid, Spain
ESTHER GUERRA, Universidad Autónoma de Madrid, Spain
JUAN DE LARA, Universidad Autónoma de Madrid, Spain

Conversational agents, or chatbots, have become popular to access all kind of software services. They provide an intuitive natural language interface for interaction, available from a wide range of channels including social networks, web pages, intelligent speakers or cars. In response to this demand, many chatbot development platforms and tools have emerged. However, they typically lack support to statically measure properties of the chatbots being built, as indicators of their size, complexity, quality or usability. Similarly, there are hardly any mechanisms to compare and cluster chatbots developed with heterogeneous technologies.

To overcome this limitation, we propose a suite of 21 metrics for chatbot designs, as well as two clustering methods that help in grouping chatbots along their conversation topics and design features. Both the metrics and the clustering methods are defined on a neutral chatbot design language, becoming independent of the implementation platform. We provide automatic translations of chatbots defined on some major platforms into this neutral notation to perform the measurement and clustering. The approach is supported by our tool Asymob, which we have used to evaluate the metrics and the clustering methods over a set of 259 Dialogflow and Rasa chatbots from open-source repositories. The results open the door to incorporating the metrics within chatbot development processes – for the early detection of quality issues – and to exploit clustering to organise large collections of chatbots into significant groups to ease chatbot comprehension, search and comparison.

CCS Concepts: • **Software and its engineering** → **Software design engineering**; *Extra-functional properties*; • **Computing methodologies** → *Natural language processing*.

Additional Key Words and Phrases: chatbot design, metrics, clustering, quality assurance, model-driven engineering

ACM Reference Format:
Pablo C. Cañizares, Jose María López-Morales, Sara Pérez-Soler, Esther Guerra, and Juan de Lara. 2023. Measuring and Clustering Heterogeneous Chatbot Designs. *ACM Trans. Softw. Eng. Methodol.* 1, 1, Article 1 (January 2023), 33 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Conversational agents (also known as chatbots) have become a popular way to access all kinds of software services – like services for customer support, banking, tourism, health and e-commerce – using conversation in natural language [36, 56]. Their use is rising because they lower the entry

Authors' addresses: Pablo C. Cañizares, Universidad Autónoma de Madrid, Madrid, Spain, Pablo.Cerro@uam.es; Jose María López-Morales, Universidad Autónoma de Madrid, Madrid, Spain, JoseMaria.LopezM@uam.es; Sara Pérez-Soler, Universidad Autónoma de Madrid, Madrid, Spain, Sara.PerezS@uam.es; Esther Guerra, Universidad Autónoma de Madrid, Madrid, Spain, Esther.Guerra@uam.es; Juan de Lara, Universidad Autónoma de Madrid, Madrid, Spain, Juan.deLara@uam.es.

1049-331X/2023/1-ART1 $15.00
https://doi.org/XXXXXXX.XXXXXXX

barrier to the services and can be ubiquitously used from many channels – from web sites to social networks and intelligent speakers – without the need to install dedicated applications. Chatbots are also used to assist workers in different domains, like software engineering [26, 41].

In response to this demand, many platforms have been proposed to construct chatbots [44], like Google's Dialogflow [14], Amazon Lex [27], IBM's Watson [61], and Rasa [48], among many others. Overall, Gartner estimates that there are more than 2000 natural language technology providers, with a significant number of them offering facilities to create chatbots [19]. Many of these solutions are low-code platforms covering the design, deployment and operation of the defined chatbots. However, their support for quality assurance is generally limited [36, 44].

Since chatbots are a kind of software, their construction should follow sound software engineering principles. In this regard, some recent approaches [5–7, 15] propose methods and tools for testing chatbots. However, while testing is essential to ensure the quality of the resulting chatbot, it requires having a functional deployed chatbot, it demands a high effort for creating testing phrases and oracles, and it is time-consuming.

We claim that the use of metrics can help to guide and control the quality of chatbots throughout their development and maintenance, becoming a complement to testing. However, to our knowledge, there are hardly any metric proposals for chatbot designs. Metrics are an accepted mechanism for assessing and controlling properties of software products and processes [17]. Chatbot static metrics can be useful to detect potential problems related to user experience (e.g., complex conversation flows, hard-to-read chatbot answers); as indicators of chatbot complexity; to compare properties of heterogeneous chatbots; to discover commonalities and cluster similar chatbots; and to understand how different implementation platforms can impact on the chatbot design. Ultimately, the availability of metrics may have notable impact on current chatbot development practices and tools, helping to increase the quality of chatbots and the efficiency of their production processes.

Moreover, the wide variety of tools with which chatbots are built makes it difficult to search and compare chatbots. This calls for mechanisms to measure the similarity or dissimilarity between chatbots, even if developed with different technologies, to understand their commonalities and enable searching for chatbots akin to a given one, e.g., as a first step towards the reuse of existing chatbots. While clustering techniques are suitable for this purpose [53], they have not been applied to chatbot designs yet.

To improve this situation, we propose a suite of 21 static metrics for chatbot designs, and two methods for clustering chatbots. The first method groups similar chatbot designs according to a selected set of metrics, which enables the identification of chatbots with (dis)similar design features (e.g., size, complexity of conversation paths, verbosity). This is useful, e.g., to classify chatbots according to their design properties; to discover potentially problematic chatbots; or to reason about the design features of large chatbot repositories (e.g., to analyse if chatbots built with different technologies have similar design characteristics). The second method performs a semantic clustering of chatbots along conversation topics, based on the frequency of the words appearing in the chatbot issued and expected phrases. This is useful, e.g., to identify chatbots on the same topic for their reuse; to facilitate the construction of recommender systems for chatbots; to organise large chatbot repositories according to their domain; or to facilitate chatbot search.

Both the metrics and the clustering methods are defined over a neutral design notation called Conga [42, 43]. This way, they become technology-independent and do not need to be reimplemented for each chatbot technology. Our proposal is available as a web platform named Asymob at http://miso.ii.uam.es/asymobService. Asymob provides importers from several chatbot platforms into our neutral design language, and so enables measuring and clustering chatbots developed with heterogeneous technologies. To assess our proposal, we report on an evaluation applying

the metrics and the clustering methods over Dialogflow and Rasa chatbots retrieved from public repositories.

This paper is an extension of our preliminary works [9, 29]. In [9], we introduced a suite of 20 chatbot design metrics, proposed their technology-independent definition atop Conga, made their implementation available via an API, and evaluated their suitability on 12 chatbots. In [29], we developed a web interface to facilitate the usage of the API, and enabled clustering chatbots based on their metrics and vocabulary, in the latter case, using a simple bag-of-words model. The present paper extends these previous works as follows:

- We have refined our suite of metrics, including a new readability metric, and expanding the explanations and rationale of the metrics.
- We have improved our metrics-based clustering by using principal component analysis.
- We have improved our vocabulary-based clustering by applying stemming (which improves efficiency as it reduces the dimensionality of the chatbot vector representation), extracting descriptive terms for each chatbot cluster, and calculating the term frequency-inverse document frequency (TF-IDF) [52] of the chatbots' vocabulary (which improves our previous implementation based on bag-of-words by considering the importance of words according to their frequency of appearance).
- We report on a thorough, extended evaluation of the metrics on a dataset of 259 chatbots (≈*22x* more than in [9]), which is available at https://github.com/asym0b/Dataset.
- We detail a novel evaluation of our semantic clustering method.

The rest of this paper is organised as follows. Section 2 introduces the core concepts behind chatbots. Section 3 overviews related work. Section 4 introduces our proposed metrics suite, and Section 5 our two methods to cluster chatbots. Section 6 describes tool support. Section 7 reports on the results of our evaluation. Finally, Section 8 finishes with the conclusions and prospects for future work.

2 BACKGROUND

Chatbots are conversational software systems with a natural language interface. They can work in open domains – like OpenAI's ChatGPT [38] or Microsoft's DialoGPT [64] – or be task-oriented. The latter are typically built to access existing software services like those in banking or shopping; or automating human services, like those for customer support. In this work, we are interested in task-oriented chatbots.

Figure 1 shows a diagram with the typical working schema of a task-oriented chatbot. The user normally starts the interaction by providing an *utterance* – a phrase in natural language – via some channel (step 1). The interaction can be using text (e.g., if the chatbot is deployed on a social network like Telegram[1]) or voice (e.g., if the chatbot is deployed on smart speakers like Amazon echo[2]). Then, the chatbot processes the utterance to give a proper response (step 6). This involves several steps, which we detail next.

First, the chatbot analyses the utterance to discover the user goals. For this purpose, most chatbots are designed around a set of *intents* (step 2 in the figure). These are conversation topics the chatbot aims at recognising, related to the offered functionality. Depending on the implementation platform, intents are defined either using *regular expressions* and *templates* (e.g., as in Pandorabots [39]) or with *training phrases* that become interpreted using natural language processing (NLP). Intents can also include *parameters*, identifying relevant information pieces to be extracted from the user utterances (step 3).

[1] https://telegram.org/ [2] https://en.wikipedia.org/wiki/Amazon_Echo

As an example, a chatbot for a cafeteria would include an intent to recognise the users' orders. This intent would be activated upon a phrase like *"I'd like a medium cappuccino"*, from which the chatbot would extract two parameters: the type of drink (*cappuccino*) and its size (*medium*). If the parameters are mandatory, but the user does not provide them, the chatbot explicitly prompts the user for their value. Parameters are typed by *entities*, which can be either pre-existing in the platform (e.g., dates or numbers), or defined for a specific domain by the chatbot developer as a list of literals with synonyms (e.g., the type and size of drinks).

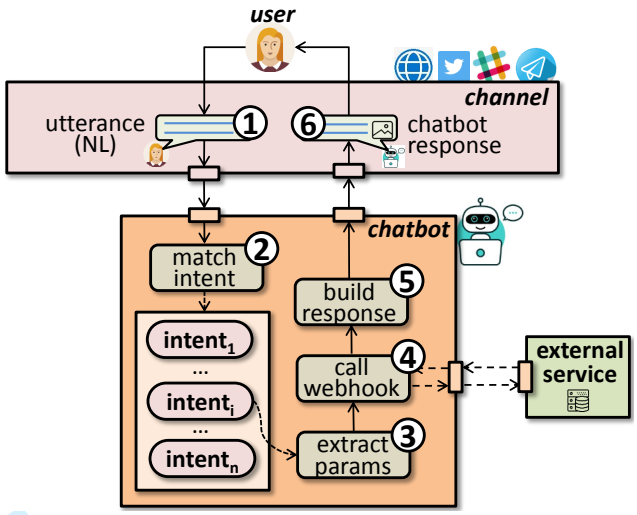


Fig. 1. Chatbot working schema.

When the chatbot receives a user utterance, it matches the more likely intent. If no matching intent is recognised, the chatbot can trigger a *fallback* intent to ask for clarification. Otherwise, if an intent is matched, the chatbot performs the actions associated to the intent. These actions always include a text response (step 5), which may incorporate media elements (e.g., images, links) or widgets supported by the specific channel (e.g., buttons in Telegram). Other possible actions are accessing a backend service (e.g., to store the drink order, step 4) or performing a computation.

Finally, conversations are structured into *flows* that intertwine user utterances and chatbot responses. As an example, when the user says *"I'd like a medium cappuccino"*, the chatbot may answer *"Would you like something to eat?"*, leading to a new user interaction (e.g., *"No thanks"*), and so on, according to the defined conversation flow. Flows may bifurcate into different conversation *paths* depending on the user utterance.

Moore and Arar [33] propose a classification of chatbots based on their conversation style. *System-centric* chatbots answer user queries or interpret commands by means of two-turn conversations (i.e., each user turn starts a new conversation, and the chatbot lacks state). *Content-centric* chatbots provide a conversational interface for FAQs, typically supplying long document-like responses that may be unsuitable for voice-based interfaces or mobile devices with small screens. *Visual-centric* chatbots facilitate user interaction via buttons and other widgets, in a style borrowed from mobile phones. Finally, *conversation-centric* chatbots mimic human dialogs, offering conversation management utterances (e.g., "*What do you mean?*") and short responses. This kind of chatbots are normally preferred because their conversation style suits a wider variety of devices and engages better in natural conversations.

3 RELATED WORK

In order to show the novelty of our contributions, this section reviews the state of the art on chatbot quality assessment (Section 3.1) and chatbot clustering (Section 3.2).

3.1 Chatbot quality assessment

Since the early days of conversational systems [62], researchers have proposed ways to evaluate their quality. For example, PARADISE [60] is an early framework for the evaluation of spoken dialogue agents, based on the correlation of performance and user satisfaction.

More recently, the popularity of chatbots has raised concerns on proper conversational design. For example, IBM's Natural Conversation Framework [34] relies on conversation patterns [35] and conversation design principles [33, 54], such as *recipient design* (i.e., provide multiple conversation paths for different user types), *minimization* (i.e., design concise chatbot answers), and *repair* (i.e., provide support for clarifications). In this line, *Chatbottest* [11] defined guidelines for identifying chatbot design issues in categories like answering, error management, intelligence, navigation, personality and understanding. This evaluation is enacted via the *Alma* chatbot, which interactively asks the chatbot designers about the features of interest. However, the burden is on the developer to manually test whether the chatbot fulfils the guidelines.

Literature reviews [40, 47] have also identified chatbot quality properties and ways to assess them. Radziwill and Benton [47] aligned chatbot quality attributes with the ISO-9241 notion of usability [20] (efficiency, effectiveness and satisfaction), while Peras [40] added further categories (e.g., information retrieval, affect). Generally, the assessment of these quality properties relies on the dynamic execution of the chatbot, on collecting statistical data, or on subjective evaluations [12, 22, 32, 49].

In their survey, Motger et al. [36] identified some chatbot research challenges, like advanced testing features and analysis methodologies to improve chatbot quality attributes. In this line, some chatbot testing approaches have been proposed to assess chatbot quality and find defects. Tools like Botium [5] or OggyBug [15] support test automation, and some works target the generation of challenging test user utterances [6, 7]. ChatEval [55] targets testing readability, which can be done automatically by applying metrics – like BLEU2 and average cosine similarity [28] – to the chatbot responses, and interactively by requiring the user complete evaluation tasks.

Finally, the NLP community has developed useful readability metrics [28, 45] that can be applied to chatbots. Pitler and Nenkova [45] combined lexical, syntactic, and discourse features in a predictive model of human readers' judgements of text readability. This model associates features like the average number of verb phrases per sentence, the number of words in the text, and the vocabulary, with human assessments of how well a text is written. Liu et al. [28] evaluate metric weaknesses based on qualitative and quantitative results, and provide recommendations for future chatbot evaluation systems.

Overall, we observe a lack of metrics specific to chatbot designs that can be evaluated statically, automatically, and independently of the chatbot implementation platform. Our goal is to fill this gap.

3.2 Clustering techniques

A means to facilitate finding artefacts of interest (chatbots in our case) is to organise them into meaningful groups. This way, a developer who wants to create a new artefact can profit from reusing existing artefacts in a cluster of interest.

Clustering is an application of unsupervised machine learning to organise items according to some criteria (e.g., a concrete metric like the number of intents). Some popular clustering algorithms are K-means, hierarchical clustering [21] and density-based spatial clustering of applications with noise (DBSCAN) [16]. All of them must be configured with a distance measure. In addition, both K-means and hierarchical clustering require setting the number of clusters as hyperparameters, while DBSCAN has other parameters like the neighbourhood distance and the minimum number of points required to form a dense region.

Clustering has been used in several areas of software engineering [57]. For example, some works [24, 30] relied on clustering to analyse and provide insights on source code. For this purpose, these works derive topics from the vocabulary used in the code (i.e., identifier names, comments), and then use latent semantic analysis (LSA) to compute the linguistic similarity between source

artefacts (i.e., functions, classes) and cluster them according to their similarity. In a similar vein, MUDABlue [23] used LSA to cluster (open-source) software systems along categories, which are determined automatically. As we will see, our semantic clustering uses similar concepts, but extracting the vocabulary from the chatbot phrases. In the chatbots field, Xu et al. [63] clustered chatbot sessions based on similarity metrics at two levels: user responses and whole chat sessions. Their goal was providing feedback to the administrator to refine the chatbot logic. Instead, our goal is to use clustering to enable comparison of chatbot designs along similar design metrics or conversation topics.

Clustering has also been employed to organise model and meta-model repositories. For instance, Babür et al. [3] represented meta-models as vectors based on their vocabulary, and used hierarchical clustering to organise and visualise collections of models and meta-models. MDEForge [4] uses a similar approach to facilitate model search. AURORA [37] uses supervised machine learning to classify meta-models into a fixed set of categories. In our proposal to chatbot clustering, we use similar encodings and techniques, but focus on chatbot designs instead of meta-models.

Other approaches group software artefacts based on metric values. For example, Arshad and Tjortjis [2] used 7 metrics to cluster C# programs with the aim of identifying potential problems in the code. Zhong et al. [65] combined clustering with expert input (to inspect and label clusters) for analysing software quality. In this case, the clustering is based on 13 code metrics like lines of code, branch count, comments or cyclomatic complexity.

In summary, while both metrics and word embeddings have been used to obtain meaningful clusters of software artefacts, to the best of our knowledge, clustering for chatbot designs has not been studied yet in the literature.

4 MEASURING CHATBOT DESIGNS

In this section, we propose a suite of metrics tailored to chatbot designs. To define the metrics independently from the chatbot implementation technology, we use a neutral design notation to represent chatbots, over which the metrics can be computed. Section 4.1 starts by introducing the chatbot design notation, and then, Section 4.2 details our proposed metrics.

4.1 A neutral notation for chatbot designs

Since our aim is to develop metrics for chatbot designs, we need a concrete notation over which to define the metrics. For this purpose, we rely on the chatbot neutral notation we proposed in [42, 43], called Conga. We opt for this neutral notation since, as explained in [42], its definition is based on a thorough review of 15 widely used chatbot development platforms. This means that the design concepts in Conga can be mapped from and to all these platforms. Hence, by defining the metrics over Conga, they become platform-agnostic as well as significant for many chatbot development platforms. As we will see in Section 6, another practical implication is that one can build importers from different platforms into Conga to measure and cluster existing chatbots built with different technologies.

Figure 2 depicts the meta-model of Conga. It permits representing a chatbot by a Chatbot object that contains a set of Intents, user-defined Entities, chatbot Actions, and conversation Flows. Conga supports multi-language chatbots (attribute Chatbot.lang), and so each intent can declare TrainingPhrases per defined language. The phrases can include Parameters, declared on the intent defining the phrase. Parameters are typed either by predefined entities (enumeration PredefinedEntity) or by user-defined Entity objects. Entities can be Simple, Regex (regular expressions) or Composite, and for each language (EntityLanguage), they declare the literals and synonyms making up the entity. For example, a chatbot can declare a simple entity for drink sizes with literals small, medium and large in English, and additionally define synonyms regular for medium and big for large.

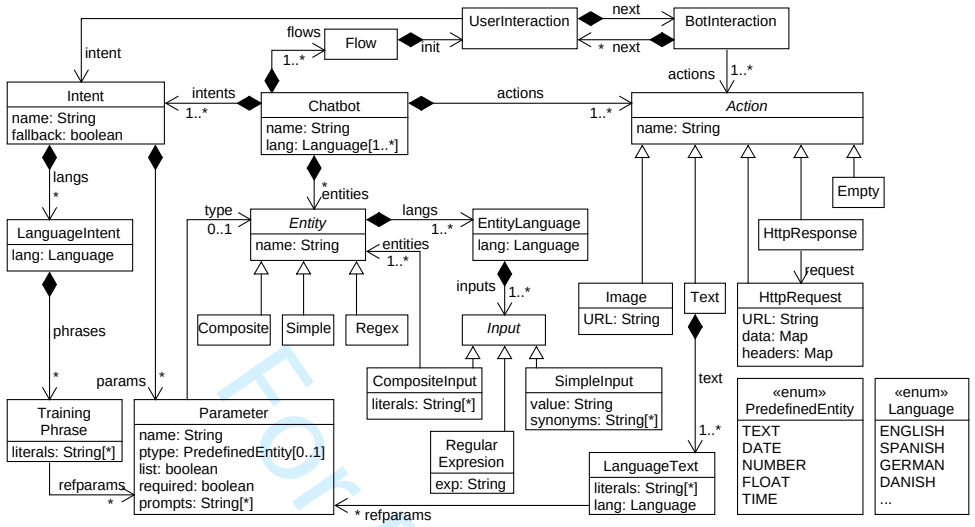


Fig. 2. Meta-model for chatbot design (from [9]).

A chatbot can define one or more Actions of type Image, Text, HttpRequest, HttpResponse and Empty. The first two types are used to compose responses combining images and text. HttpRequest and HttpResponse allow configuring the chatbot communication with external services in the backend. The Empty action is a wildcard for other platform-specific actions.

The conversation flow between the chatbot and the users is modelled by Flow objects consisting of user and chatbot turns (classes UserInteraction and BotInteraction). The user turn has a reference to the intent to be recognised in the interaction (reference UserInteraction.intent). The bot turn specifies the actions that the chatbot must perform (reference BotInteraction.actions).

4.2 A metrics suite for chatbot designs

Table 1 shows our proposed suite of 21 metrics for chatbot designs. All metrics measure internal attributes of chatbots. We considered three sources when designing these metrics:

- Some of them, like INT (the number of intents) or ENT (the number of user-defined entities), are calculated by taking statistics of concepts covered by the meta-model in Figure 2. According to [42], these concepts are common to many chatbot development frameworks.
- Some other metrics, like VPTP, READ and OPRE, have been adapted from the NLP literature [8, 18, 28, 45] to assess the readability of the chatbot responses, their estimated reading time, or the complexity of the expected user utterances.
- Finally, we use the conversation design principles proposed in [33, 54], and Moore and Arar's classification of chatbots [33], to interpret the value of some metrics such as PATH (the number of conversation paths), FLOW (the number of conversation entry points) and WPO (the number of words per chatbot output).

The fourth column of Table 1 classifies the potential impact of the metrics on usability (as defined in the ISO 9241-11 [20]) in terms of Effectiveness (i.e., accuracy and completeness with which users achieve their goals), efficiencY (i.e., time and resources that users expend to achieve their goals) and Satisfaction (i.e., comfort and acceptability of use). Metrics are also classified depending on their target: either global design properties, or specific aspects of intents, entities or conversation

Table 1. Chatbot design metrics. Column Dim (Dimension) uses abbreviations for Effectiveness, efficiencY and Satisfaction.

Metric	Description	Type	Dim
Global metrics			
INT	# intents	design size	E
ENT	# user-defined entities	vocabulary size	S
FLOW	# conversation entry points	conversation diversity	E
PATH	# different conversation flow paths	conversation complexity	S,Y
CNF	# confusing phrases	bot understanding	E,S
SNT	# positive, neutral, negative output phrases	user experience	S
Intent metrics			
TPI	# training phrases per intent	topic complexity	E,S
WPTP	# words per training phrase	topic complexity	Y
VPTP	# verbs per training phrase	topic complexity	S,Y
PPTP	# parameters per training phrase	topic complexity	E
WPO	# words per output	readability	S,Y
CPO	# characters per output	readability	S,Y
VPOP	# verbs per output phrase	readability	S
READ	reading time of the output phrases	readability	Y
OPRE	output phrase readability	readability	S, Y
Entity metrics			
LPE	# literals per entity	vocabulary complexity	S
SPL	# synonyms per literal	vocabulary complexity	S
WL	word length	readability	Y,S
Flow metrics			
FACT	# actions per flow	bot response complexity	E,S
FPATH	# conversation flow paths	conversation complexity	S,Y
CL	conversation length	conversation complexity	Y

flows. Non-global metrics can be computed per element (intent, entity, flow) or averaged for all elements of a kind.

4.2.1 Global metrics. We start explaining *global metrics*. These measure the number of intents (INT), entities (ENT) and conversation flows (FLOW, PATH), and also include understanding and user experience metrics (CNF, SNT).

INT is an indicator of design size and functionality, since each intent contributes functionality offered to the user. The larger INT is, the more functionality the bot offers, potentially impacting effectiveness. ENT measures the size of the chatbot vocabulary and the conversation topic diversity, which may affect satisfaction. FLOW counts the number of conversation entry points for users, being an indicator of conversation diversity. Since each entry point might correspond to a functionality, FLOW may impact effectiveness. PATH measures the number of paths a conversation may take, which is an indicator of conversation complexity. If PATH=FLOW, all conversations are linear, while if PATH>FLOW, some conversation splits into several paths.

As an example, Figure 3 shows two small excerpts of chatbot designs conformant to the Conga meta-model. The chatbot design (a) depicts a linear flow (i.e., FLOW=PATH=1). Linear flows enable simple conversations, typically request/response, which may indicate a system-centric chatbot [33]. The chatbot design (b) shows a conversation flow that splits after the bot interaction (FLOW=1, PATH=2). This kind of flows permits non-linear conversations with multiple turns and dialogue alternatives, typical of conversation-centric chatbots [33].

The combined use of FLOW and PATH can help detecting deviations of some design principles. The *recipient* principle [54] advices to design for the target users, from experts (who may give all information at once) to novices (where the bot needs to prompt for more information). In turn, the

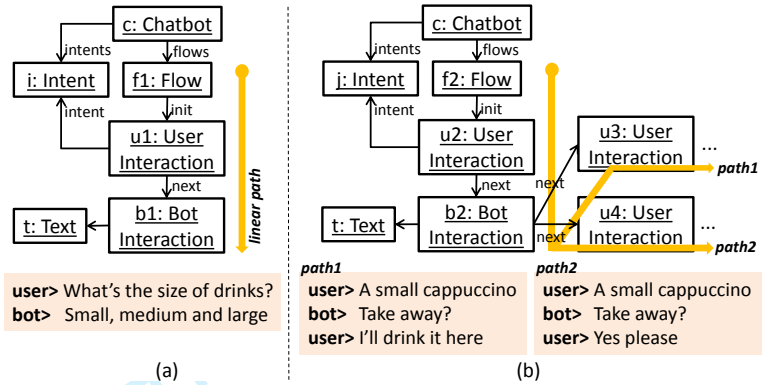


Fig. 3. Chatbot design excerpts illustrating (a) a linear conversation flow, (b) a forked conversation flow.

repair principle [54] recommends supporting clarifications in the conversation, and multiple paths may be an indication of this. Moreover, having several paths per flow potentially results in more natural conversations (impacting satisfaction) but less predictable for the user (likely impacting the user effort or efficiency).

The CNF metric measures the semantic distance between the training phrases of different intents [10]. For this, the training phrases are tokenised and converted into 512-dimensional vectors using sentence embedding. Then, the semantic distance between two phrases is given by the cosine similarity between their embedding vectors. This metric is used to detect similar phrases that may confuse the bot to make it identify a wrong intent. Since this may cause errors, the metric is related to effectiveness and satisfaction.

The last global metric, SNT, measures the sentiment of the chatbot output phrases, classifying the phrases into positive, negative and neutral. This sentiment analysis uses a compositional model over trees based on deep learning [58]. Specifically, sentences are parsed into binary trees, where all leaves correspond to a word – represented as a vector – and each node receives a sentiment score. Then, the sentiment is calculated by recursive neural models that compute parent vectors in a bottom up fashion using several compositionality functions. The SNT metric is related to satisfaction, since a bot that outputs mostly negative phrases may cause a negative user experience [49].

4.2.2 Intent metrics. *Intent metrics* measure quality properties of each intent with respect to the expected user utterances and the bot output phrases.

Related to user utterances, TPI is the number of training phrases that an intent defines. The larger TPI is, the more precise the intent recognition might be, but this may also indicate a complex intent. WPTP measures the length of the training phrases in words. Long phrases are not adequate or even possible in social networks (e.g., Twitter restricts the message length), and are more difficult to understand for the chatbot. Therefore, large WPTP values might be problematic. VPTP measures the number of verbs per training phrase. This is an indication of interaction complexity, since composite phrases with several verbs can be more difficult to elaborate for the user [45]. PPTP counts the number of information items (i.e., parameters) the user needs to provide in a phrase, and high values signal intents involving complex domain concepts.

Regarding chatbot outputs, WPO measures the number of words per chatbot output. According to the *minimization* principle [33], the bot answers should be concise. Large phrases are more difficult to understand and can be problematic in social networks. The latter is more concretely targeted by CPO (characters per output), since high values may require scrolling (e.g., in mobile devices)

and long reading times (with the risk that the user does not read the phrase completely [33]). Long outputs are especially problematic for voice-based chatbots, since speaking takes longer than reading [33]. Hence, large CPO values may decrease user satisfaction and efficiency. Similarly, VPOP is another indicator of the complexity of the bot responses, given by the number of verbs per output phrase. A chatbot can answer several phrases in a single output, and VPOP is computed for each phrase separately to better assess phrase complexity. READ gives an estimation of the reading time of the bot responses, which is related to efficiency. It is calculated as the ratio between the number of words per output, and the number of words that an average person can read per minute [8]. Finally, OPRE computes the readability of the chatbot responses using the Flesch Reading Ease Formula [18]. This metric yields a number ranging from 0 to 120, and the higher the number, the easier the chatbot responses are to read. Hence, OPRE may affect both satisfaction and efficiency.

4.2.3 Entity metrics. *Entity metrics* target user-defined entities representing domain concepts. LPE (literals per entity) and SPL (synonyms per literal) are indicators of the complexity of the concepts managed by a chatbot, impacting satisfaction. High LPE and SPL values signal elaborate concepts, but since SPL counts synonyms, a large number may improve recognition in user utterances (better satisfaction). A narrow vocabulary (low SPL) may constrain the way users communicate with the chatbot, and may lead to frustration if the chatbot does not recognise important parameters within user utterances. WL measures the length of words (i.e., the literals within the entities), and like CPO, it contributes to readability and may impact user satisfaction and efficiency.

4.2.4 Flow metrics. *Flow metrics* consider features of the conversation flows. FACT counts the bot actions (presenting images, text, calling backends) in each conversation flow. The more actions, the more sophisticated tasks can be achieved. Moreover, rich controls can help to reduce the user cognitive load and speed up the completion of the intended task. Hence, FACT may impact effectiveness and satisfaction.

FPATH measures the number of possible paths per conversation flow. High values signal complex conversations (i.e., more natural-sounding but less predictable). The PATH global metric is calculated by adding up FPATH for each flow.

Finally, CL measures the length of each path within a flow, as the number of bot and user turns. This is an indicator of conversation complexity. Longer paths require more time to complete – which affects efficiency – and are typical of conversation-centric chatbots [33].

5 CLUSTERING CHATBOTS

In this section, we propose two methods to cluster chatbots based on two disjoint criteria: metric values (Section 5.1) and chatbot vocabulary (Section 5.2). The former allows grouping chatbots by internal quality features (e.g., complex/simple conversations, large/succinct outputs). The latter groups chatbots by conversation topic.

5.1 Metrics-based clustering

Metrics-based clustering is useful to identify groups of chatbots with (dis)similar design features. For this purpose, the chatbots are characterised based on the value of one or more metrics of interest. For example, clustering by metric INT (i.e., number of intents) would create groups of chatbots with similar size complexity, whereas if the user performs the clustering using metrics FLOW and PATH, then the chatbots would be grouped according to the complexity of their conversations.

Figure 4 shows the steps in our metrics-based clustering process. First, the process transforms the chatbots of interest into the CONGA neutral notation (step 1). Then, the user selects a subset of the metrics proposed in Section 4.2, which are applied to the chatbots (step 2). This way, each chatbot becomes represented by a vector of size n, where n is the number of metrics selected by

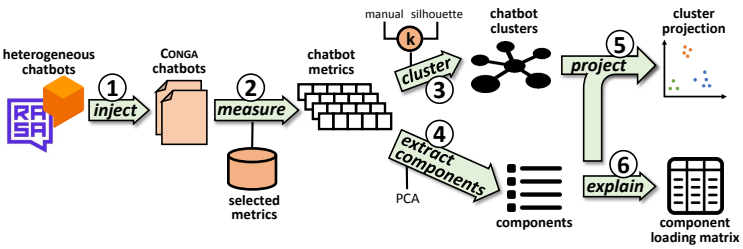
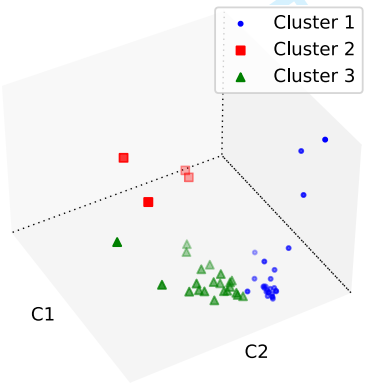


Fig. 4. Metrics-based clustering process.



	Cluster 1	Cluster 2	Cluster 3
Size (#chatbots)	28	4	22
PATH	7.18	88.75	11.05
INT	10.46	77.5	11.81
FLOW	3.54	73.25	9.77
WL	0	5.15	9.19
PPTP	0.21	1.13	1.55
FPATH	3.34	1.25	1.24

Fig. 5. Cluster projection into three dimensions, and average metric values for the chatbots in each cluster.

the user, and each position of the vector contains the value of a selected metric for the chatbot. Next, the process calculates the clusters by applying the *k*-means algorithm on the chatbot metric vectors (step 3). The number of clusters to create (i.e., the *k*-value) can be either specified by the user or automatically computed using the silhouette coefficient [51]. Our process also performs a principal component analysis (PCA) of the metric vectors (step 4). PCA is a method to reduce the dimensionality of data while minimising information loss. In our case, it produces new variables (the components) that compress information of the metrics so that the first few components explain most of the variance of the metric values. We use PCA with two purposes. On the one hand, to project the clusters into a two or three-dimensional space for visualization, by selecting the two or three principal components (step 5). On the other hand, to identify sets of related metrics (step 6). The latter is possible since PCA permits explaining how each metric contributes to each component, by providing the *loading* factor of the metric for the component. This effectively groups the metrics that contribute the most to each component.

As an example, Figure 5 shows the result of clustering a set of 54 chatbots[3] by using the metrics PATH, INT, FLOW, WL, PPTP and FPATH. In the graphic, each dot represents a chatbot, and the shape of the dot identifies the cluster where the chatbot belongs. Overall, the process yields 3 clusters with sizes 28, 4 and 22. The table in Figure 5 shows the average metric value for the chatbots in each cluster. At a first glance, one can see that clusters 1, 2 and 3 contain chatbots with low, high and medium values of PATH, INT and FLOW; low, medium and high values of WL and PPTP; and high, low and low values of FPATH.

[3] Dataset available at https://github.com/ASYM0B/SmallDataSet2.

Table 2. Explained variance per component, and loading matrix, for the metrics and clusters in Figure 5.

	Components		
	C1	C2	C3
Explained variance (%)	48%	27%	16%
Cumulative explained variance (%)	48%	75%	91%
Loading matrix			
PATH	0.58		
INT	0.57		
FLOW	0.57		
WL		0.66	
PPTP		0.63	
FPATH			0.88

Table 2 shows the first three components found by PCA, their eigenvalues (the percentage of variance they explain), and their loading factors (the table omits factors lower than 0.42). The results confirm our previous intuition: the first component (C1) aggregates metrics PATH, INT and FLOW; the second one (C2) aggregates WL and PPTP; and the third (C3) includes FPATH. These three components explain 91% of the variance (cf. cumulative explained variance).

We can interpret this information to summarise the chatbots that are in each cluster. Specifically, the first component (C1) distinguishes cluster 2 from clusters 1 and 3. The metrics with the highest weight in C1 are PATH, INT and FLOW, which measure the conversation flow paths, the intents, and the conversation entry points. The table in Figure 5 shows that cluster 2 has the highest values for these metrics, so we conclude that the chatbots in this cluster are bigger than the rest, in the sense that they take into account more conversational contexts. Looking again at Figure 5, we can see that the second component (C2) separates clusters 1 and 3. The metrics that impact C2 the most are WL and PPTP, which measure the average word length of entities and the parameters per training phrase. We observe that the chatbots in cluster 1 have very low values for these metrics. In particular, they have zero words per entity, likely because most of them are defined using Rasa, which lacks a standard way to declare entities. In summary, cluster 2 contains complex chatbots, cluster 1 contains Rasa chatbots, and cluster 3 includes the rest of chatbots.

5.2 Vocabulary-based clustering

In addition to our metrics-based clustering, which groups chatbots depending on their internal design properties, we provide another clustering method based on the chatbot conversation topics. For example, chatbots for ordering food are likely to be in the same cluster, since their vocabulary is similar. For this purpose, we use a technique similar to Latent Semantic Analysis (LSA) [25] to compute the distance between chatbots.

Figure 6 describes our process for vocabulary-based clustering. The input to the process is a set of chatbots, which may have been developed with different technologies. Then, the process performs the following steps.

(1) **Inject.** The first step is the same as for metrics-based clustering, and transforms the chatbots into the Conga neutral notation.
(2) **Extract vocabulary.** Next, the process extracts the vocabulary that each chatbot uses. This is made of the training phrases of the intents, the chatbot output phrases, and the literals and synonyms in entities.
(3) **Pre-processing.** The extracted vocabulary is tokenised to retrieve the words in phrases, and then normalised as follows. Stop words such as prepositions, articles and conjunctions are

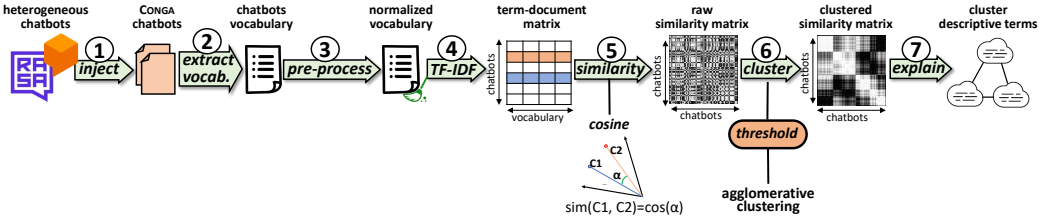


Fig. 6. Vocabulary-based clustering process.

removed, since they do not provide relevant information. Moreover, the *Porter2* stemming algorithm [46] is applied to the words to keep just their morphological root (i.e., the stem). This way, two words that only differ in their inflected forms are considered equal. Overall, stemming reduces the dimensionality of the chatbot vector representations, and it is useful for our purposes since it permits considering two chatbots using the same stemmed words as similar.

(4) **TF-IDF.** Next, to quantify the relevance of a word to a chatbot, the process computes the term frequency-inverse document frequency (TF-IDF) [52]. This is one of the most popular term-weighting methods today, widely used in information retrieval, text mining and analysis, and clustering of source code [24]. Specifically, each chatbot C is represented as a vector, where each position in the vector represents a (stemmed) word w, and contains the weight $W(w, C)$ of the word w for the chatbot C. This weight is calculated considering the frequency $f(w, C)$ of the word w in the chatbot C and in the set of chatbots $\mathcal{R}$ to be clustered, as given by the formula:

$$W(w, C) = \frac{f(w, C)}{max\{f(t, C) \mid t \in C\}} \cdot log\left(\frac{|\mathcal{R}|}{|\{D \in \mathcal{R} \mid w \in D\}|}\right) \quad (1)$$

A word w receives a high weight when w has a high appearance frequency in the given chatbot C, and a low appearance frequency in the whole set of chatbots R. Hence, the weights tend to filter out common terms. Note that the weight vector of small chatbots may have many zeros.

Overall, the output of this step is a term-document matrix, where each row corresponds to a chatbot, each column corresponds to a (stemmed) word, and each cell of the matrix contains the TF-IDF weight of a word for a chatbot.

Other chatbot representations, such as bag-of-words, or embeddings like Word2Vec [31], are also possible. We experimented with the former representation in [29]. Using Word2Vec would require a very large corpus of chatbots for training, and it is left as future work.

(5) **Similarity.** The term-document matrix is used to measure the similarity between each two chatbots. For this purpose, we use the cosine similarity, which computes the angle between the vectors representing the two chatbots. To optimise this computation, each TF-IDF vector is normalised so that the only operation to do is multiplying the weights of each word. The cosine similarity yields a number between 0 and 1. Chatbots having analogous conversation topics will have a similarity close to 1, while chatbots of very different domains will obtain a low similarity tending to 0.

(6) **Cluster.** Next, our process uses agglomerative clustering [50] to group the chatbots based on their vocabulary-based similarity. This algorithm relies on a notion of distance. In our case, the distance d between two chatbots C_1 and C_2 is defined as $d(C_1, C_2) = 1 - sim(C_1, C_2)$, where $sim(C_1, C_2)$ is the cosine similarity of the chatbots. The algorithm creates clusters by

Table 3. Most descriptive (stemmed) terms in the eight biggest clusters.

		Top terms				
Clusters	**Size**	#1	#2	#3	#4	#5
Cluster 1	4	restaur	food	indian	north	serv
Cluster 2	4	transact	payment	transfer	pay	account
Cluster 3	4	student	graduat	bachelor	ec	faculti
Cluster 4	4	room	book	hotel	reserv	breakfast
Cluster 5	4	flight	return	leav	travel	book
Cluster 6	3	virus	corona	covid	spread	coronavirus
Cluster 7	3	news	headlin	latest	bbc	australia
Cluster 8	3	ticket	movi	concert	theater	cinema

merging successively the two nearest chatbots C_1 and C_2 to yield a new element C_{12}. The distance $d(C_{12}, C_3)$ from this new element C_{12} to another chatbot C_3 is defined by $d(C_{12}, C_3) = \frac{d(C_1,C_3)+d(C_2,C_3)}{2}$. This way, at each iteration, the algorithm removes the entries corresponding to C_1 and C_2 from the distance matrix, and adds a new element C_{12}. The algorithm stops when there are no more elements with a distance less than a given threshold t. This step yields two outputs: a clustered similarity matrix having chatbots as rows and columns, where each position (i, j) in the matrix contains the distance (a value between 0 and 1) between chatbots i and j; and a dendrogram with the cluster hierarchy.

(7) **Explain.** To understand the obtained clusters, a final step extracts the most descriptive terms for each cluster. A term is descriptive with respect to a cluster if it has a high appearance frequency in the cluster, in comparison to its frequency in the whole corpus of chatbots. Formally, we define the importance I of a term t in a cluster C with respect to the set of chatbots $\mathcal{R}$ as:

$$I(t)_{C,\mathcal{R}} = \frac{1}{|C|}\sum_{C\in C} W(t, C) - \frac{1}{|\mathcal{R}|}\sum_{C\in\mathcal{R}} W(t, C)$$

where $W(t, C)$ is the weight of the term t for the chatbot C in the term-document matrix (cf. Equation 1). This way, the importance of a term for a cluster is greater the more it appears in the cluster, and the less it appears in the chatbots outside the cluster.

As an example, Figure 7 shows the clustered similarity matrix resulting from step 6 for a set of 54 chatbots[4]. The matrix uses colours to represent chatbot similarity, where higher similarities (i.e., lower distance values) are depicted with darker colours. In addition, the figure represents the grouping of clusters by means of dendrograms. The number of clusters can vary depending on the used threshold. For example, a threshold of 0.87 yields 17 non-singleton clusters.

Table 3 shows the size of the eight biggest clusters, as well as their five most representative terms obtained as explained in step 7[5]. These terms enable recognising common topics of the chatbots that belong to a same cluster. For example, we can guess that the clusters contain chatbots related to restaurants (cluster 1), banking (cluster 2), teaching (cluster 3), hotels (cluster 4), air travel (cluster 5), covid (cluster 6), news (cluster 7) and leisure (cluster 8).

6 TOOL SUPPORT

We have built a tool called Asymob supporting the measurement and clustering of chatbot designs specified with Conga. The tool is deployed as a web platform at http://miso.ii.uam.es/asymobService. Its code is open source and available at https://github.com/PabloCCanizares/asymob.

[4] Dataset available at https://github.com/ASYM0B/SmallDataSet. This dataset is different from the one used in the example of Section 5.1, as it serves to illustrate better vocabulary-based clustering. [5] Additionally, there are nine clusters with size two, and the rest are singletons.

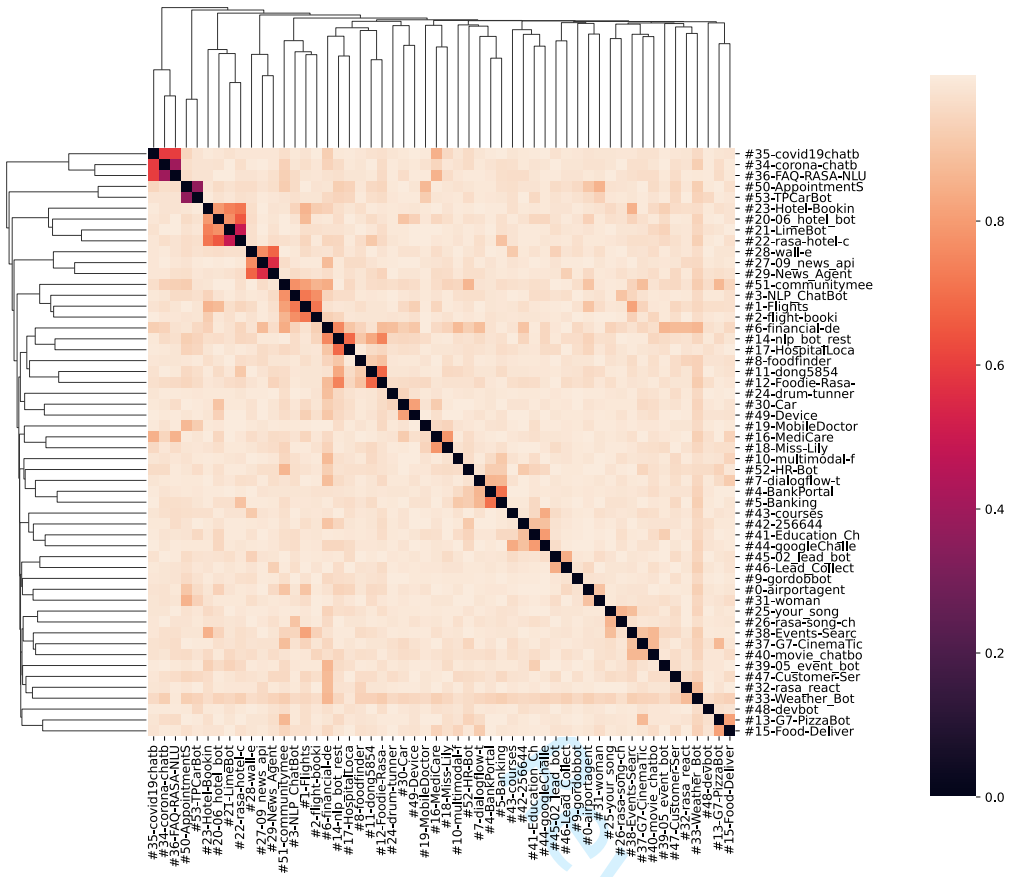


Fig. 7. Clustered similarity matrix of chatbots. Each cell represents the distance between two chatbots by colour intensity, where darker colours mean lower distance (i.e., higher similarity).

Next, Section 6.1 presents the architecture of Asymob, including its main features and underlying technologies; Section 6.2 details the conversion of chatbots implemented in two mainstream platforms (Dialogflow and Rasa) into Conga; and Sections 6.3 and 6.4 describe the usage of the tool for chatbot measurement and clustering.

6.1 Overview of Asymob

Asymob supports *uploading* chatbots of heterogeneous technologies into a common repository. The uploaded chatbots are then *measured* using the metrics presented in Section 4.2. Asymob provides *statistics* of the metrics across all chatbots in the repository. In addition, users can *query* the repository to search for chatbots within certain metric bounds and *compare* them against each other according to their metric values. The platform also allows *clustering* chatbots by their metric values (cf. Section 5.1) or by the conversation topics as given by the words appearing in their training phrases, chatbot responses and entities (cf. Section 5.2).

Figure 8 depicts the architecture of Asymob. Its functionality is offered via a web interface, which interacts with a service layer via a REST API. The *front-end* is implemented in HTML and JavaScript,

and supports the interactive presentation of metrics and clusters using the libraries Plotly[6] and Cytoscape[7], while word clouds for describing chatbots and clusters are visualised with JQCloud[8].

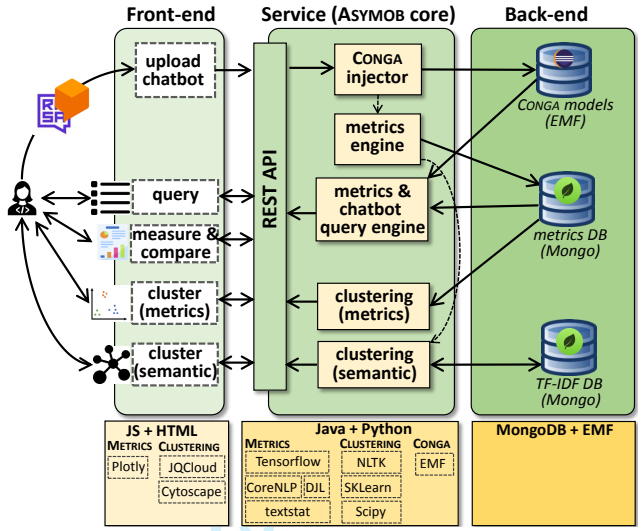


Fig. 8. Architecture of Asymob.

The *service layer* (the Asymob core) implements the functionality related to measuring and clustering chatbots. This core has an extensible design, which makes it easy to add new types of metrics, clustering criteria and chatbot technologies. To support chatbots from different platforms, it relies on the neutral chatbot design notation Conga, introduced in Section 4.1. Conga is built atop the Eclipse Modeling Framework (EMF) [59]. This way, Asymob computes the metrics on Conga models, independently of any chatbot implementation platform. To measure chatbots from a specific platform, an importer from the platform into Conga must be provided. Currently, Asymob has importers from Dialogflow and Rasa. Section 6.2 provides more details about these two importers.

Asymob supports some third-party technologies to simplify the implementation of new metrics. Specifically, it uses the library Stanford CoreNLP[9] to perform sentiment and syntactic analysis of the chatbot training and output phrases. The metrics SNT and VPTP use this library. In addition, Asymob uses textstat[10] – a Python library to obtain statistics from text – to calculate the metric OPRE, as well as Deep Java Learning (DJL)[11] and TensorFlow[12] to detect confusing phrases between intents using the cosine similarity and calculate the metric CNF based on this algorithm. The chatbot clustering functionality uses other Python libraries like NLTK[13], SKLearn[14] and SciPy[15].

The service layer has a REST API, which we have used to create the web platform (i.e., the front-end). This API also permits integrating our services in custom chatbot construction platforms, or within custom in-house development workflows (e.g., using Github actions, continuous integration servers like Jenkins, or DevOps processes).

An additional *backend layer* provides persistence. This stores the uploaded chatbots in the filesystem of the machine where the Asymob core is deployed, and uses MongoDB[16] for storing

[6] https://plotly.com/ [7] https://cytoscape.org/ [8] https://mistic100.github.io/jQCloud/
[9] https://stanfordnlp.github.io/CoreNLP/ [10] https://pypi.org/project/textstat/ [11] https://djl.ai/
[12] https://www.tensorflow.org/ [13] https://www.nltk.org/ [14] https://scikit-learn.org/ [15] https://scipy.org/
[16] https://www.mongodb.com/

the data produced in the service layer (i.e., the metric values and the information required for conducting clustering).

As Figure 8 shows, when a chatbot is uploaded into the platform, it is converted to Conga and immediately measured, storing the results in a MongoDB database. In addition, the TF-IDF frequency of each word in the chatbot is calculated and stored in a database as well. These are background processes that do not slow down the uploading process, but reduce the response time of future requests of chatbot measurement, comparison and clustering.

6.2 Importing chatbots into Conga

Next, we provide details of the importers that Asymob provides to convert chatbots from two representative and widely used chatbot platforms (Dialogflow and Rasa) into Conga.

6.2.1 Importer from Dialogflow to Conga. Dialogflow [14] is a low-code development platform to create chatbots using a graphical interface within the browser. Chatbots so defined can be exported as JSON files, which our importer converts into Conga models.

In the JSON-based representation of a Dialogflow chatbot, the file *Agent.json* describes global chatbot features, like its name, definition languages, or connection data to external services (the *webhook*). The latter include details such as the URL of the service, headers, and authentication credentials. Our importer creates a Conga Chatbot object using the agent's name and languages, and an HttpRequest action with the webhook data.

Entities in Dialogflow can be predefined or user-defined. The latter data are described either by a regular expression, a list of literals with synonyms, or a composite entity. Each user-defined entity becomes exported as a JSON file containing the entity name and configuration information (if it is a regular expression or a composite entity), and one file per definition language with the corresponding literals. Our importer converts these files into Conga Entity objects.

Intents in Dialogflow have a name, training phrases, responses, parameters, and an indication of whether they are fallback or enable a webhook, among other features. Intents are exported into JSON files. For each intent definition file, our importer creates a Conga Intent object with its Parameters and TrainingPhrases, as well as the necessary Actions to compose each response. We currently support text and image responses, and convert other custom responses into Empty actions (but this does not affect the defined metrics).

Finally, Dialogflow controls the conversation flow via contexts. These can be input/output to intents, and can store relevant conversation state. Our importer uses the contexts and the responses of the related intents to generate Conga Flow objects.

6.2.2 Importer from Rasa to Conga. Rasa [48] is a framework to develop chatbots combining Python, markdown and YAML. The definition of Rasa chatbots comprises several files. The *config.yml* file defines configuration properties, like the chatbot language or the used NL prediction model. The *data/nlu.md* file contains training phrases for the intents, together with entities and synonyms or regular expressions. As illustration, the *data/nlu.md* file in Listing 1 defines an intent called order (lines 1–4). The parameters in the training phrases are defined within square brackets and are followed by the entity name either in parenthesis (e.g., [cappuccino](type)) or within curly brackets (e.g., [medium]{"entity": "size", "value": "medium"}). The listing also declares synonyms for literals small (lines 5–7), medium (8–10) and large (11–13).

The file *domain.yml* defines the chatbot intents, entities and actions. Actions can be text, images, buttons, or custom actions defined in the Python file *actions.py*. Finally, the file *data/stories.md* specifies the conversation flows. Listing 2 shows a flow example, specifying that matching the intent order triggers the response utter_confirm_order.

```
## intent:order
- I'd like a [medium]{"entity": "size", "value": "medium"} [cappuccino](type)
- I want a [small]{"entity": "size", "value": "small"} [latte](type)
- Can I order a [large]{"entity": "size", "value": "large"} [black](type) coffee?
## synonym:small
- little
- short
## synonym:medium
- regular
- median
## synonym:large
- big
- extra
```

Listing 1. Example of *data/nlu.md* Rasa file

```
## story1
* order
    - utter_confirm_order
```

Listing 2. Example flow from *data/stories.md* Rasa file

We have built an importer that reads the chatbot language from the *config.yml* file and creates Conga intents and entities from the *data/nlu.md* file, Conga actions from the *domain.yml* file, and Conga flows from the *data/stories.md* file. As in the case of Dialogflow, our importer from Rasa supports text and image responses, and converts Rasa custom actions into Conga empty actions. As a limitation, entities in Rasa can also be defined using Python code, instead of using the declarative, explicit approach shown in Listing 1. In this case, our importer is not able to produce Conga entities, and signals this fact using a warning.

6.3 Measuring chatbots with Asymob

When a chatbot is uploaded, Asymob computes its metrics and displays their value in a table and also in interactive graphs that compare these values with statistics of the chatbots in the repository. Figure 9 shows the graph for metric INT. The left bar displays statistics of the chatbot repository, and the bar to the right displays the metric value for the uploaded chatbot. We observe that the new chatbot can be considered small since it has 4 intents, while the average number of intents of the chatbots is around 15 (with a median of 7). The computed metrics are persisted to speed up the generation of statistics when new chatbots are uploaded, and to facilitate the following functionalities.

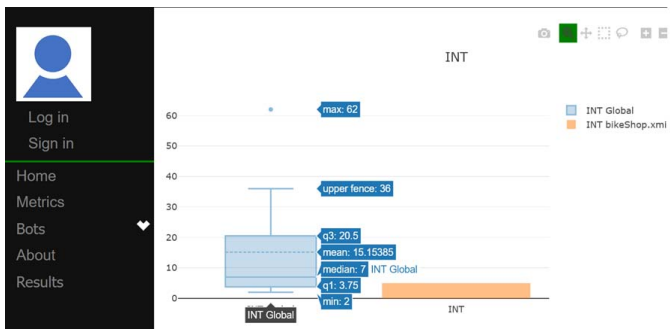


Fig. 9. Chatbot measurement.

First, Asymob offers statistics of the metrics of all chatbots in the repository (average, minimum, maximum, median and 1st and 3rd quartiles). They are displayed as a table, as a graph, and side-by-side with the metric values of a specific chatbot, as Figure 9 shows.

Asymob permits comparing a collection of chatbots based on a set of metrics selected by the user, as shown in Figure 10. In the upper graphic, the x-axis displays the selected metrics (13 in the figure), and the y-axis shows their value for the chatbots selected from the repository (10 bots in this case). The lower graphic permits zooming on the values for one of the selected metrics (bars to the right) and comparing them with the average metric value in the repository (bar to the left). In the figure, the user has zoomed on the values for metric INT. We can see that Car and recruitment-bot-rasa stand out in this metric, meaning that they have more conversation alternatives (intents). This comparison can also be performed for several versions of the same chatbot (if different versions were uploaded into the repository) to reason about the evolution between chatbot versions in terms of metrics.

Fig. 10. Chatbot metrics comparison.

The platform also includes a metric-based chatbot search facility, where users can specify the lower and upper limits for the value of some metrics of interest, and Asymob displays the chatbots in the repository with metric values within these boundaries. This is useful to obtain sets of chatbots with certain characteristics, e.g., simple chatbots with few intents and no defined entities, or complex chatbots with many intents and complex conversation flows.

6.4 Clustering chatbots with Asymob

Asymob supports the metrics-based and vocabulary-based clustering methods described in Section 5. In both cases, the user starts by selecting the chatbots to cluster. Then, the result depends on the clustering method.

In the case of metrics-based clustering, the tool displays the resulting clusters in a table and graphically, as Figure 11 shows. The graph can display two or three dimensions, so if the user selects more than three metrics as clustering criteria, the platform reduces the number of dimensions using PCA. The graphic represents each chatbot as a dot, and uses a different colour for each cluster of chatbots. The graphic is interactive, supporting rotation, zooming, and visualisation of each chatbot name. In Figure 11, there are 4 clusters with 161, 82, 9 and 7 chatbots. A table displays the average of the selected metrics of each cluster. This page also enables the inspection of each cluster (displaying the chatbots within the cluster in a table), and the information of the PCA (explained variance and loading matrix).

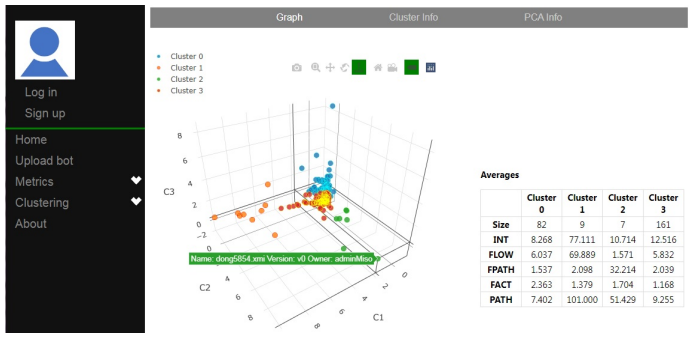


Fig. 11. Metrics-based clustering.

With regards to vocabulary-based clustering, in addition to the chatbots, the user can select a similarity threshold for the agglomerative clustering algorithm. Then, the resulting clusters are shown in a table, in an interactive hierarchical graph, and it is also possible to visualise the clustered similarity matrix on demand (cf. Figure 7). In the hierarchical graph visualization, the first graph layer has a node per cluster, and clicking on a cluster shows the bots it contains. As an example, Figure 12 shows the chatbots within a cluster, and a word cloud with the most frequent words of the chatbots in the cluster. The most relevant words for the cluster can also be displayed on a table. The width of the edges in the graph is proportional to the similarity between the two connected chatbots. Clicking on a chatbot displays its metrics on a table.

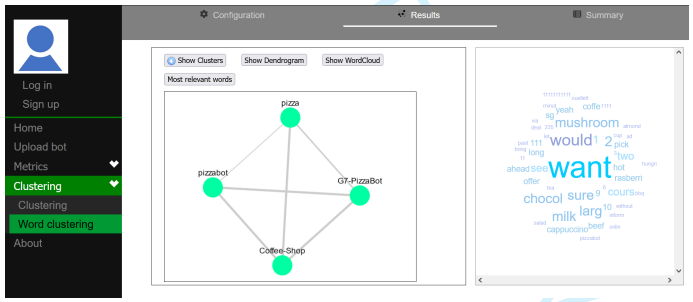


Fig. 12. Vocabulary-based clustering.

7 EVALUATION

This section reports on an empirical study whose goal is assessing the usefulness of our metrics and the efficacy of our vocabulary-based clustering. First, Section 7.1 overviews the followed research methodology and states the research questions (RQs).

7.1 Research methodology

Our evaluation aims to answer the following RQs:

RQ1 How do chatbots in the wild compare with respect to their size, conversation style, outputs, expected inputs and vocabulary?

RQ1.1 Do design metrics for Rasa and Dialogflow chatbots follow the same distributions?

RQ2 Can the defined metrics detect quality issues in real chatbots?

RQ3 Can the vocabulary-based clustering create meaningful groups of semantically related chatbots?

For this purpose, we followed the research methodology depicted in Figure 13. Firstly, we created a dataset of Rasa and Dialogflow third-party chatbots, obtained from several relevant sources. Section 7.2 characterises this dataset. Then, we applied different research methods to answer each RQ.

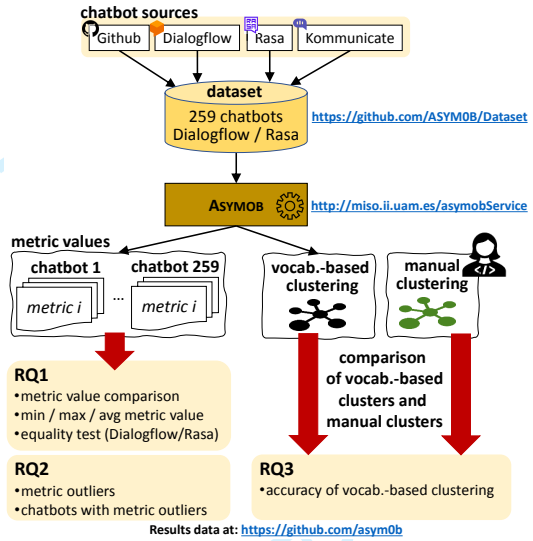


Fig. 13. Research methodology for our evaluation.

The study of RQ1 aims to obtain a panorama of the features of chatbots in the wild, and RQ1.1 pursues to assess if metrics could serve as a means to analyse the impact of the used implementation platform in the features of the underlying chatbot designs. To answer both questions, we used ASYMOB to import the dataset of chatbots into CONGA, and take measurements. Then, we compared different dimensions of the chatbots (size, conversation style, responses, expected user utterances and vocabulary) based on the chatbots' metric values. We also compared the minimum, maximum and average values of each metric per technology. Finally, we conducted some equality tests to analyse whether the metrics have the same probability distribution of values in Dialogflow and Rasa. Section 7.3 reports on this analysis.

The goal of RQ2 is to evaluate whether the proposed metrics can help to statically detect potential problems ("*bad smells*") in chatbot designs. We do not expect every discordant metric value to signal a real problem, but instead, we are interested in assessing the usefulness of metrics as an inexpensive chatbot quality assurance mechanism (e.g., compared to testing) that can be used early in the development process. To answer this question, we analysed the obtained measurements to identify statistical outliers for each metric (i.e., metric values that largely differ from the other metric values). After that, we looked at the chatbots with outlier values for some of metrics, to identify whether these values signalled a design error. Section 7.4 identifies the metric outliers per technology, reporting the percentage of those that present problems.

Finally, the aim of RQ3 is evaluating the extent to which our semantic clustering can produce groups of chatbots that are meaningful. To answer this, we performed the clustering of all chatbots in the dataset using our vocabulary-based approach on the one hand, and manually on the other

hand. Then, we computed the accuracy with which the vocabulary-based clusters mimic the manual clusters built by a human. Section 7.5 reports the results.

7.2 Experiment setting

The dataset for our evaluation contains as many Rasa and Dialogflow third-party chatbots as possible, coming from three sources: code repositories like Github, predefined chatbots from the Dialogflow and Rasa platforms, and predefined chatbots from other platforms like Kommunicate[17].

Table 4 describes our dataset, available at https://github.com/asym0b/Dataset. It contains 259 chatbots, classified either as open source if they were located in an open-source repository, or predefined if they were available as prebuilt agents in some platform. Predefined chatbots typically illustrate the platform capabilities in an application domain, and can be reused and modified to fit specific needs. The dataset is balanced in terms of technologies (117 Dialogflow chatbots, 142 Rasa chatbots), but predefined chatbots are a minority in the dataset (24 predefined chatbots, 235 open source chatbots).

Table 4. Description of the chatbots dataset.

Technology	Source	Kind	#Chatbots
Dialogflow	Github	Open source	96
	Dialogflow	Predefined	14
	Kommunicate	Predefined	7
Rasa	Github	Open source	139
	Rasa	Predefined	3
			259

7.3 RQ1 & RQ1.1: Chatbot comparison and metrics distribution across technologies

To answer RQ1 and RQ1.1, we applied the metrics suite to all chatbots in our dataset. The detailed metric values for each chatbot are available at https://github.com/asym0b/Dataset. Table 5 displays a summary of the results. The table shows the minimum, maximum, average and median values for each metric, distinguishing between Dialogflow chatbots, Rasa chatbots, or globally (i.e., considering Dialogflow and Rasa chatbots together). Intent, entity and flow metrics are calculated averaging the values over the number of intents, entities and flows in each chatbot. An exception is CL, which takes the maximum conversation length in the chatbot to be more informative. The sentiment metric SNT is disaggregated into positive (SNT^+), neutral ($SNT^=$) and negative (SNT^-). The last two columns of the table display the result of two nonparametric tests used to identify differences between the distribution of metric values in Dialogflow and Rasa (for RQ1.1, cf. Section 7.3.6).

Next, we exploit the metrics to compare the chatbots based on their size, conversation style, outputs, expected inputs, vocabulary (pertinent for RQ1) and implementation platform (RQ1.1).

7.3.1 Size. Metrics can be used to compare and classify chatbots based on their size. In particular, metric INT (number of intents) is a good indicator for chatbot size. Figure 14 shows the distribution of this metric across the chatbots' set, where the bars are divided between Rasa and Dialogflow chatbots. The global median is 8 intents, 90% of chatbots have less than 24 intents, just 13 chatbots (4%) have more than 40 intents, and 29 chatbots (11%) have 3 intents or less. Some of the smallest chatbots may be toy examples built to experiment with the technology, most frequently in Dialogflow.

[17] https://www.kommunicate.io/

Table 5. Summary of metric values for the chatbots in the dataset.

Metric		Dialogflow				Rasa				Global				Equality tests	
Type	Name	Min	Max	Avg	Median	Min	Max	Avg	Median	Min	Max	Avg	Median	MW	KS
Global metrics	INT	1.0	89.0	10.9	6.0	2.0	143.0	15.4	9.0	1.0	143.0	13.37	8.0	$\leq 10^{-3}$	$\leq 10^{-3}$
	ENT	0.0	37.0	2.44	0.0	0.0	9.0	0.39	0.0	0.0	37.0	1.31	0.0	$\leq 10^{-3}$	$\leq 10^{-3}$
	FLOW	1.0	89.0	9.27	5.0	1.0	102.0	6.96	3.0	1.0	102.0	8.01	4.0	$\leq 10^{-3}$	$\leq 10^{-3}$
	PATH	1.0	117.0	10.51	5.0	1.0	207.0	15.04	7.0	1.0	207.0	13.0	6.0	0.029	0.070
	CNF	0.0	1606.0	87.42	3.0	0.0	10532.0	190.27	10.0	0.0	10532.0	143.81	7.0	$\leq 10^{-3}$	$\leq 10^{-3}$
	SNT^{+}	0.0	38.0	6.68	4.0	0.0	58.0	17.65	17.0	0.0	58.0	12.69	12.0	$\leq 10^{-3}$	$\leq 10^{-3}$
	$SNT^{=}$	0.0	100.0	47.38	64.0	34.0	100.0	73.55	73.0	0.0	100.0	61.73	70.0	$\leq 10^{-3}$	$\leq 10^{-3}$
	SNT^{-}	0.0	100.0	16.03	17.0	0.0	51.0	8.8	6.0	0.0	100.0	12.07	10.0	$\leq 10^{-3}$	$\leq 10^{-3}$
Intent metrics	TPI	0.0	94.67	9.04	5.56	0.0	235.17	18.46	8.09	0.0	235.17	14.21	6.75	$\leq 10^{-3}$	$\leq 10^{-3}$
	WPTP	0.0	9.62	2.72	2.41	0.0	5.37	3.03	2.83	0.0	9.62	2.89	2.75	0.007	$\leq 10^{-3}$
	VPTP	0.0	1.76	0.55	0.44	0.0	1.33	0.6	0.55	0.0	1.76	0.58	0.52	0.053	0.004
	PPTP	0.0	8.33	0.65	0.33	0.0	1.6	0.32	0.25	0.0	8.33	0.47	0.27	0.008	0.036
	WPO	0.0	23.11	6.02	5.79	2.25	87.63	9.37	6.63	0.0	87.63	7.86	6.27	0.001	$\leq 10^{-3}$
	CPO	0.0	125.56	27.23	23.56	10.82	440.1	43.87	28.4	0.0	440.1	36.35	26.06	$\leq 10^{-3}$	$\leq 10^{-3}$
	VPOP	0.0	3.56	0.91	1.04	0.17	3.52	1.26	1.2	0.0	3.56	1.1	1.12	0.004	$\leq 10^{-3}$
	READ	0.0	19.0	4.84	4.0	1.0	75.0	7.56	5.0	0.0	75.0	6.33	5.0	0.002	$\leq 10^{-3}$
	OPRE	0.0	92.0	54.69	75.0	44.0	113.0	89.3	89.5	0.0	113.0	73.67	84.0	$\leq 10^{-3}$	$\leq 10^{-3}$
Entity metrics	LPE	0.0	1177.13	22.06	0.0	0.0	0.0	0.0	0.0	0.0	1177.13	9.97	0.0	$\leq 10^{-3}$	$\leq 10^{-3}$
	SPL	0.0	7.13	1.25	0.0	0.0	0.0	0.0	0.0	0.0	7.13	0.57	0.0	$\leq 10^{-3}$	$\leq 10^{-3}$
	WL	0.0	36.0	3.87	0.0	0.0	0.0	0.0	0.0	0.0	36.0	1.75	0.0	$\leq 10^{-3}$	$\leq 10^{-3}$
Flow metrics	FACT	1.0	8.0	1.88	2.0	1.0	3.92	1.31	1.1	1.0	8.0	1.57	1.31	$\leq 10^{-3}$	$\leq 10^{-3}$
	FPATH	1.0	3.5	1.11	1.0	1.0	57.0	4.01	1.67	1.0	57.0	2.7	1.13	$\leq 10^{-3}$	$\leq 10^{-3}$
	CL	1.0	8.0	1.51	1.0	1.0	211.0	7.46	4.0	1.0	211.0	4.77	3.0	$\leq 10^{-3}$	$\leq 10^{-3}$

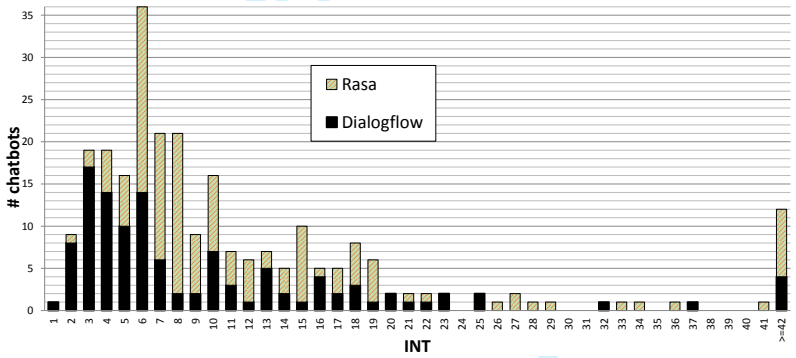


Fig. 14. Distribution of the number of INTents per chatbot.

7.3.2 Conversation. Metrics also serve to compare and classify chatbots based on their conversation style [33]. Some chatbots are prepared to hold a wider variety of conversations according to their number of intents (INT), conversation flows (FLOW) and conversation paths (PATH). Figure 15 projects the dataset along these three metrics. We observe that there is high correlation between the metrics (0.76 correlation between INT and FLOW, 0.62 between INT and PATH, and 0.69 between FLOW and PATH). Naturally, chatbots with more intents tend to have more flows, and therefore, more paths.

Conversations are fully linear in 46% of the chatbots (FLOW=PATH), and the remaining 54% chatbots support more complex conversations (FLOW<PATH). Most chatbots with linear conversations (69.2%) have medium-to-low size (8 intents or less). Figure 15 highlights the chatbot iLearn, as it is the largest chatbot with linear conversation, having 89 intents, flows and paths, and built using Dialogflow. Regarding chatbots with non-linear conversations, they have an average of 4.2 paths per flow (average of $\frac{PATH}{FLOW}$, or FPATH[18]). As Figure 15 shows, some extreme cases include rasa-workshop-pydata-berlin (37 paths per flow) and dong5854 (57 paths in its unique flow). These chatbots feature very complex conversations, offering many choices to the user to continue the

[18] In Table 5, FPATH is calculated for all chatbots, not only for those with non-linear conversations.

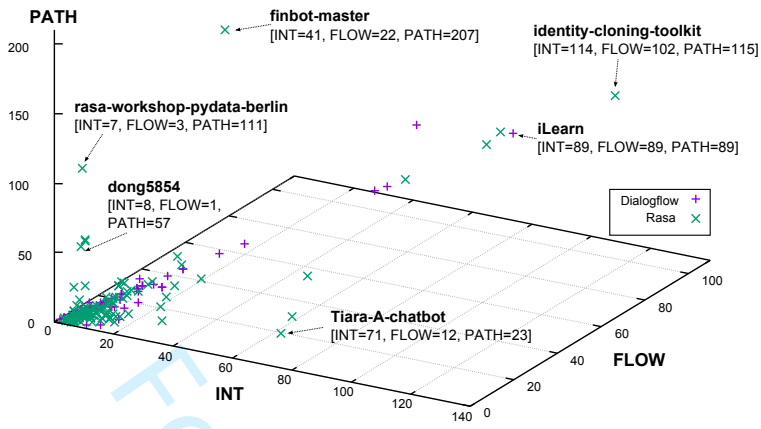


Fig. 15. Comparing chatbot conversation styles based on the distribution of INT, FLOW and PATH per chatbot.

conversation, which might be confusing. Other big chatbots supporting non-linear conversations are more balanced, like finbot-master (41 intents, 22 flows, 207 paths, 9.4 paths per flow), Tiara-A-Chatbot (71 intents, 12 flows, 23 paths, 1.92 paths per flow) or identity-cloning-toolkit (114 intents, 102 flows, 115 paths). The latter chatbot is almost linear, with 1.13 paths per flow.

Additionally, Figure 16 shows the maximum conversation length (CL) of the analysed chatbots. In 92 chatbots, conversations are limited to one user-bot interaction (CL=1), and hence, they can be classified as system-centric [33]. Within this set, chatbots providing long responses are likely content-centric. Just two chatbots – nep-chatbot and FAQ_RASA_NLU – can be classified within this category. Both were built with Rasa and have over 54 words per output in average (WPO>54). Other chatbots allow longer conversations with more turns (CL>1). Chatbots with non-linear conversations (FLOW<PATH) have necessarily more than one turn (CL>1) and can be classified as conversation-centric [33]. In our dataset, the global median of CL is 3. Moreover, CL is 12 or less in 96% of the chatbots, and 6 or less in 84% of them.

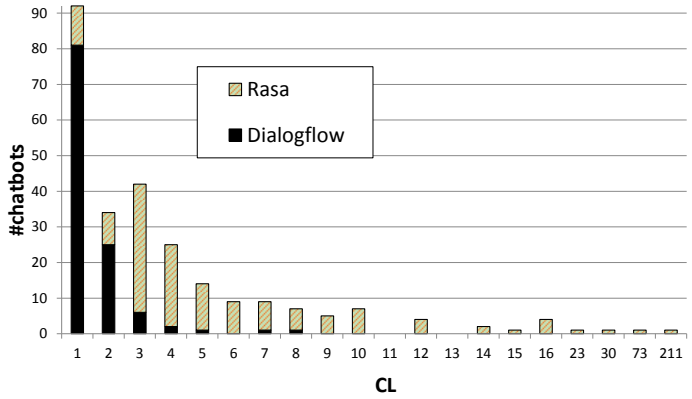


Fig. 16. Distribution of the chatbots' (maximum) conversation length (CL).

7.3.3 Outputs. Chatbots can be compared based on their responses to users. Interestingly, 35 chatbots (a 13.5% of the total) have no output phrases. In some cases – like the Car or Food-delivery predefined agents of Dialogflow – this is because the chatbot is a template agent that the developer needs to complete with custom outputs. In other cases – like the MysteryAnimal chatbot game built with Rasa – it is because a backend API generates the output dynamically.

Metrics WPO (words per output) and CPO (characters per output) help to compare the verbosity of the chatbots. The global median of WPO is around 6 words per output, and CPO is around 26 characters per output. These metrics take high values on verbose chatbots. For example, Tiara-A-Chatbot has over 87 words per output. As we will see in Section 7.4, this may indicate a quality problem. On the other end, two Dialogflow chatbots – fulfillment-temperature-converter and fulfillment-multi-locale – reply 2 words or less on average.

The OPRE metric captures the readability of phrases [18]. Its formula was designed to yield scores between 0 and 100, but it is possible to reach a maximum value of 120. The higher the value, the more readable the phrase. The dataset has a global average for OPRE above 73, which suggests easy-to-read output phrases in general. However, two chatbots – fulfillment-temperature-converter and fulfillment-multi-locale – have values of 17 and 35, hinting poor readability.

Regarding the sentiment of the output, we observe that, globally, most outputs have neutral tone ($SNT^{=}$ is 61.73%), while outputs with positive (SNT^{+}) and negative (SNT^{-}) tone are roughly equally distributed.

7.3.4 Expected inputs. Similarly as with the chatbot outputs, one can compare the complexity of the expected user inputs. Metrics WPTP and VPTP measure the words and verbs per training phrase. With a global average of 2.89 words and 0.58 verbs per phrase, chatbots generally expect uncomplicated user utterances. In addition, PPTP counts the number of parameters (i.e., pieces of information) that users should provide in each phrase. We find that 84 chatbots (32.4%) do not require users to provide any parameters (PPTP=0), 146 chatbots (56.3%) require an average number of parameters equal or less than 1 in user utterances ($0<PPTP\leq 1$), and only 29 chatbots (11.1%) expect more than 1 parameter in average, peaking at 8.33 parameters in the predefined chatbot template Dining-Out. Hence, chatbots are typically designed so that users have to provide very little information, if any, in each interaction.

With regards to the size of the training phrase set, TPI measures the average number of training phrases per intent. The global average is 14 phrases per intent. A total of 14 chatbots (5.4%) have a value of TPI equal or below 1, which may result in intent recognition problems when the chatbots are deployed. On the other extreme, 15 chatbots (5.8%) have an average of 50 training phrases or more per intent. This may signal complex intents that require many examples for accurate intent recognition, or overly specified intents.

7.3.5 Vocabulary. Entity metrics allow comparing the richness of the vocabulary that chatbots can recognise. Globally, 31% of the analysed chatbots define at least one entity ($ENT>0$). The average number of entities per chatbot is 1.31, with extreme cases of chatbots raising to 37 entities in Dialogflow (chatbot MysteryAnimal) and 9 in Rasa (chatbot Foodie-Rasa-Chatbot). High ENT values indicate a wide variety of domain-specific terms (e.g., animals in case of MysteryAnimal, or food types in case of Foodie-Rasa-Chatbot).

We can look deeper into each entity using metrics LPE (literals per entity), SPL (synonyms per literal) and WL (word length). These metrics are 0 in Rasa chatbots because Rasa can defer the recognition of entity literals to Python methods, in addition to being explicitly declared in a text file. We can see that the latter option is hardly ever used. As for Dialogflow chatbots, their average LPE value is just over 22, though a remarkable case is the Dining-Out chatbot with more than 1000 literals per entity. This chatbot recommends bars and restaurants, so its entities define long lists

of cocktails, food types, and specific bars and restaurants. The next subsection discusses this and other chatbot platform differences in more detail.

7.3.6 Metric distribution across technologies. Our approach enables the measurement of chatbots built with different technologies. To assess whether a given metric has the same probability distribution of values in Dialogflow and Rasa, we applied the Mann-Whitney U (MW) and Kolmogorov-Smirnov (KS) equality tests on the metric values obtained for each technology. The last two columns of Table 5 report the p-values returned by these two statistical tests. A p-value >0.05 in either of the two columns indicates that the metric follows the same probability distribution of values in both technologies.

The results show that the Dialogflow and Rasa chatbots in our dataset follow different distributions for all metrics but PATH and VPTP (i.e., these two metrics are the only ones with p-values of either MW or KS above 0.05). For the rest of metrics, we have that:

- Rasa chatbots have more intents (INT) than Dialogflow chatbots, as well as more confusing phrases (CNF), higher percentages of positive and neutral phrases (SNT^+, $SNT^=$), and higher values of TPI, WPTP, WPO, CPO, VPOP, READ, OPRE, FPATH and CL.
- Dialogflow chatbots have more entities (ENT), conversation flows (FLOW), phrases with negative tone (SNT^-), parameters per training phrase (PPTP), and actions per flow (FACT), than Rasa chatbots.

Overall, Rasa chatbots in our dataset are globally bigger (higher INT, cf. Figure 14); were trained with more examples (TPI) but including more confusing phrases (CNF); have less negative phrases (SNT^-); are more verbose (WPO, CPO) leading to higher reading times (READ) but where phrases are easier to read (OPRE); and maintain longer conversations (higher CL, cf. Figure 16) where users can take more paths (FPATH). As discussed in Section 7.3.5, entity metrics are 0 for Rasa chatbots due to the encoding of entity literals in Python methods.

Regarding Dialogflow chatbots, they define more domain-specific terms (ENT); have more conversation entry points (FLOW); issue a higher percentage of negative phrases (SNT^-); require users to provide more data in utterances (PPTP); and perform more actions per flow (FACT).

Note that, as a sanity check, we used the MW and KS tests to assess that chatbots of a same technology have the same distribution. For this purpose, we randomly split the dataset of each technology in two subsets, and applied the tests on the subsets. In all cases, we obtained p-values greater than 0.05, which implies an underlying distribution of metric values of Dialogflow chatbots, and also of Rasa chatbots.

7.3.7 Answering RQ1 and RQ1.1. With respect to RQ1, as detailed in Sections 7.3.1–7.3.5, our metrics permit comparing chatbot size using INT; conversation style along Moore and Arar's taxonomy [33] using FLOW, PATH, FPATH and CL; chatbot outputs using WPO, CPO, VPOP, OPRE and SNT; expected user inputs using WPTP, VPTP and PPTP; and chatbot vocabulary size using ENT, LPE, SPL and WL. Moreover, since metrics are defined over Conga, they are applicable to different technologies and enable the comparison of heterogeneous chatbots. Regarding RQ1.1, as described in Section 7.3.6, we found that, for our dataset of chatbots, all metrics but PATH and VPTP follow different distributions for Dialogflow and Rasa.

7.4 RQ2: Detection of quality issues

The previous section hints that metrics can reveal chatbot design issues. Now, to answer RQ2 more precisely, we show in Table 6 the outliers of the metrics separated by technology. For each metric, high outliers are values above Q_3+1.5· *IQR*, and low outliers are values below Q_1-1.5· *IQR* (with Q_1

Table 6. Metric outliers per technology.

			High outliers			Low values		
Type	Metric	Median	Cutoff	Bot%	Sample chatbots (max. 2)	Cutoff	Bot%	Sample chatbots (max. 2)
Dialogflow								
Global	INT	6	26.5	5.1%	iLearn (89), Car (77)	2	7.7%	ChronoGG (1), fulfillent-importer (2)
	ENT	0	7.5	8.5%	MysteryAnimal (37), googleChallenge (34)	0	52.14%	iLearn (0), insurance_Bot (0)
	FLOW	5	20.5	7.7%	iLearn (89), MysteryAnimal (62)	1.66	1.7%	ChronoGG (1), Buddy-G7 (1)
	PATH	5	24	7.68%	Car (117), iLearn (89)	1.66	1.70%	ChronoGG (1), Buddy-G7 (1)
	CNF	3	75	17.09%	Car (1606), iLearn (1599)	1	40.17%	airportagent (0), basic-slotfilling (0)
	SNT^+	4	27.5	0.85%	BikeShop (38)	1.33	45.29%	dialogflow-silly-name-maker (0), fulfillment-temperature-converter (0)
	SNT^-	17	62.5	2.56%	dialogflow-google-sign-in (100), dialogflow-ssml (100)	5.66	35.04%	defaults-chatfuel (0), defaults-manychat (0)
Intent	TPI	5.56	20.17	6.83%	Dining-Out (94.67), Hotel-Booking (50.67)	1.85	17.94%	hackathon-group-10 (0.33), in-my-seats-jovo (0.33)
	WPTP	2.41	6.59	1.7%	googleChallenge (9.60), Car (6.8)	0.8	5.98%	fulfillment-multi-locale (0.5), hackathon-group-10 (0.67)
	VPTP	0.44	1.56	0.85%	googleChallenge (1.76)	0.14	12.82%	libsample-advanced (0.03), dialogflow-webhook-boilerplate (0.06)
	PPTP	0.33	2.08	5.12%	Dining-Out (8.33), Hotel-Booking (5)	0.11	29.91%	airportagent (0), keijiban (0)
	WPO	5.79	20.15	3.41%	HHandoffDAgent (23.11), Education_Chatbot (22.89)	1.93	30.76%	Car (0), fulfillment-temperature-converter (1.5)
	CPO	23.56	86.42	6.83%	Education-Chatbot (125.56), googleChallenge (105.16)	7.85	29.91%	Car (0), ChronoGG (0)
	VPOP	1.04	3.42	0.85%	fulfillment-telephony (3.56)	0.35	31.62%	libsample-advanced (0.1), HOTEL-BOOKING-AGENT2 (0.29)
	READ	4	15	5.12%	Education-Chatbot (19), HHandoffDAgent (19)	1.33	31.62%	fulfillment-temperature-converter (1), fulfillment-multi-locale (1)
Entity	LPE	0	12.5	11.11%	Dining-Out (1177.13), ekgsBot (359.86)	0	52.13%	iLearn (0), airportagent (0)
	SPL	0	5.92	1.70%	Formats (7.13), iotairblower (6.83)	0	54.70%	iLearn (0), airportagent (0)
	WL	0	17.95	1.70%	gordobbot (36), ekgsBot (20.61)	0	54.70%	iLearn (0), airportagent (0)
Flow	FACT	2	3.85	2.56%	HOTEL-BOOKING-AGENT2 (8), keijiban (4.71)	0.66	0%	-
	FPATH	1	1	16.23%	Dining-Out (3.5), HR-Bot (2.33)	0.33	0%	-
	CL	1	3.5	4.27%	enoreese (8), Food-Ordering-Chatbot (7)	0.33	0%	-
Rasa								
Global	INT	9	28.12	9.15%	covid-19-chatbot (143), identity-cloning-toolkit (114)	3	2.1%	07_survey_bot (2), 04_feedback_bot (3)
	ENT	0	0	17%	Foodie-Rasa-Chatbot (9), insurance-en (4)	0	83%	covid-19-chatbot (0), identity-cloning-toolkit (0)
	FLOW	3	11.4	10.6%	identity-cloning-toolkit (102), small-talk-rasa-stack (86)	1	16.9%	covid-19-chatbot (1), dong5854 (1)
	PATH	7	26.5	14.78%	finbot-master (207), identity-cloning-toolkit (115)	2.33	7.74%	concertbot (1), formoriginal (1)
	CNF	10	93.12	13.38%	covid-19-chatbot (10532), identity-cloning-toolkit (3461)	3.33	18.30%	04-feedback-bot (0), 07-survey-bot (0)
	SNT^+	17	46.87	0.7%	yassinelamarti (58)	5.66	12.67%	09-news-api (0), Ali (0)
	SNT^-	6	36.8	1.4%	FAQ-RASA-NLU (51), trackncov19 (44)	2	38.73%	05-event-bot (0), yassinelamarti (0)
Intent	TPI	8.08	29.21	12.67%	aniketbangar (235.17), sokkalingam (214.33)	2.69	12.67%	concertbot (0), Tiara-A-Chatbot (1)
	WPTP	2.83	6.23	0%	-	0.94	1.40%	concertbot (0), Tiara-A-Chatbot (0.88)
	VPTP	0.55	1.44	0%	-	0.18	5.63%	07-survey-bot (0), 09-news-api (0)
	PPTP	0.25	1.30	1.40%	flight-booking (1.6), Foodie-Rasa-Chatbot (1.5)	0.08	40.14%	rasa-faq-bot (0), 02-lead-bot (0)
	WPO	6.63	13.94	7.04%	Tiara-A-Chatbot (87.63), Data-Mining-Chatbot (73.37)	2.21	0%	-
	CPO	28.39	68.56	8.45%	Tiara-A-Chatbot (440.1), Data-Mining-Chatbot (382.9)	9.46	0%	-
	VPOP	1.2	2.16	5.63%	Data-Mining-Chatbot (3.52), FAQ-RASA-NLU (3.33)	0.4	2.11%	Chatbot-Banking (0.17), WeatherBot (0.2)
	READ	5	11.5	7.04%	Tiara-A-Chatbot (75), Data-Mining-Chatbot (62)	1.66	0.70%	concertbot (1)
Flow	FACT	1.1	1.88	10.56%	Data-Mining-Chatbot (3.92), concertbot (3.5)	0.36	0%	-
	FPATH	1.67	5.46	14.78%	dong5854 (57), rasa-workshop-pydata-berlin (37)	0.55	0%	-
	CL	4	1.33	7.74%	aniketbangar (211), covid-19-chatbot (73)	1.33	7.74%	01-smalltalk-bot (1), 07-survey-bot (1)

and Q_3 the first and third quartiles, and IQR the interquartile range). Since all our metrics have low outliers below 0, we identify instead chatbots with low metric values below $Q_2/3$.

The table columns show the metric type, metric name, median, high outliers (cut-off value Q_3+1.5·IQR, percentage of chatbots with a metric value above the cut-off, and two outlier chatbots), and low metric values (cut-off value $Q_2/3$, percentage of chatbots with a metric value below the cut-off, and two sample chatbots). For each sample chatbot, the table shows its name and the metric value in parentheses.

Next, we analyse the outliers to detect possible problems related to chatbot size, conversation style, outputs, expected inputs, and vocabulary.

7.4.1 Size. Intents (INT) provide an estimation of the chatbot size. On the low side, 7.7% of Dialogflow chatbots (9 chatbots) have 2 intents or less, while 3 Rasa chatbots have 3 intents or less. This may be an indication of incomplete chatbots, probably built to experiment with the technology (e.g., ChronoGG to experiment with Dialogflow and Firebase, and 07_survey_bot to experiment with Rasa).

On the high side, 5.1% of Dialogflow chatbots have more than 26 intents, and 9.15% of Rasa chatbots have more than 28 intents. The extreme cases reveal complex chatbots and may signal redundant intents. As Section 7.4.4 will show, covid-19-chatbot (143 intents) and identity-cloning-toolkit (114 intents) have many confusing phrases (CNF) suggesting the existence of similar intents that could have been reused.

7.4.2 Conversation. High outliers of metrics FLOW and PATH may reveal chatbots with unusually wide conversation options. Some of these chatbots, like identity-cloning-toolkit (102 flows, 115 paths),

small-talk-rasa-stack (86 flows, 92 paths), finbot-master (22 flows, 207 paths) and iLearn (89 flows, 89 paths) are ELIZA-style chatbots [62] or emulate chit-chat conversation with many possible user entry points. In turn, large values of FPATH signal complex conversations with many possible bifurcations. The chatbots dong5854 (FPATH=57) and rasa-workshop-pydata-berlin (FPATH=37), both performing chit-chat conversations, stand out in this metric. Finally, high outliers of CL (conversation length) may uncover accessibility problems and errors in the conversation design. For instance, aniketbangar, a chatbot to find restaurants, has the highest CL value (211) but only a conversation entry point (FLOW=1). On inspection, we found that the chatbot is erroneous since it concatenates all flows into one, so that when the chatbot answers with the restaurants, the flow continues with a user greeting, starting the search for restaurants again.

On the other side of the spectrum, low values of FLOW, PATH and FPATH may suggest restricted conversation options or incomplete chatbot designs. For example, Buddy-G7 defines 10 intents but only one conversation flow between fallback intents, hence missing conversation options that use the other intents. On inspection, we found that the chatbot has an erroneous setup of the intent contexts in Dialogflow.

7.4.3 Outputs. High outliers of metrics VPOP, CPO, WPO and READ may indicate problematic chatbot outputs. For instance, the chatbot fulfillment-telephony has 3.42 verbs per output phrase (VPOP), which is an indicator of complex chatbot responses. It defines outputs like *"I'm sorry I didn't catch that do you want to continue making a reservation or would you prefer to be transferred to the main line?"*, which lacks punctuation marks separating the phrases within the output.

The CPO of Tiara-A-Chatbot is 440 characters. This may cause accessibility and readability problems, as large answers require scrolling in mobile devices, long reading times (1 minute and 15 seconds according to its READ value), and cannot be fully displayed on social networks like Twitter due to their message length constraints. As an example, Figure 17 shows a response of this chatbot deployed on Telegram using a mobile phone, which requires scrolling as the response has more than 30 lines. This is an example of a content-centric chatbot to access a covid-19 FAQ. However, according to [33], conversation-centric chatbots with short answers and a natural conversation style are usable in more platforms. Another chatbot with the same problem though to a lesser extent is Education-Chatbot (CPO=125.56, READ=19).

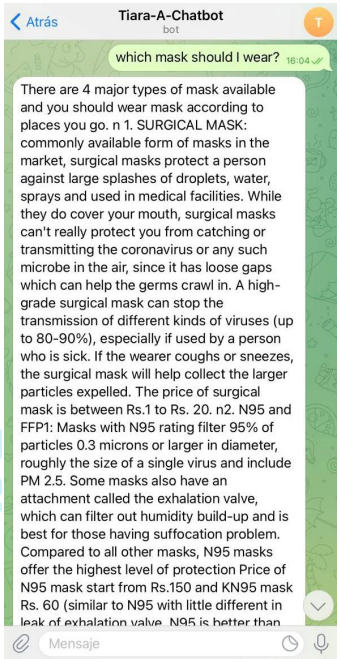


Fig. 17. Long response of Tiara-A-Chatbot in a mobile in Telegram.

The sentiment of the chatbot responses can also affect the user experience. The chatbots dialogflow-google-sign-in, dialogflow-ssml and FAQ-RASA-NLU have 100%, 100% and 51% of negative responses (SNT^-). The responses of FAQ-RASA-NLU are related to covid-19 spread, so many have a negative tone. Chatbots that issue a broad variety of error messages may also have a high percentage of negative responses. For example, dialogflow-ssml only defines 3 responses, but all of them report that the chatbot was not able to access an external service (e.g., *"Sorry, I couldn't get a response for the Welcome intent from your webhook."*). A similar situation occurs with the dialogflow-google-sign-in chatbot. Finally, low values for SNT^+ may be problematic depending on the chatbot domain, like those for chit-chat. Fifteen chatbots (7 Dialogflow chatbots, and 8 Rasa ones) have no responses with positive tone, none of which are chit-chat bots.

7.4.4 Expected inputs. Outliers of metrics concerning expected user utterances (TPI, WPTP, VPTP, PPTP, CNF) may also signal problems.

The chatbots with the fewest training phrases per intent (TPI) are hackathon-group-10 and in-my-seats-jovo. Their average TPI is 0.33, which may affect the chatbot's ability to understand users. Instead, in their analysis of NL engines [1], Abdellatif et al. recommend at least 10 training phrases per intent. Other chatbots, like Tiara-A-Chatbot, define training phrases for a few intents but leave others empty, signalling an unfinished implementation. In general, incomplete chatbots can be more accurately detected by looking if their minimum value of TPI is zero, than observing their average TPI. With respect to high outliers of TPI, they may be due to the need to recognise complex sentences, but other times they are indicative of an unnecessarily large training set. For example, sokkalingam (TPI=214.33) defines training phrases with many combinations of dates, number of tickets and person names; however, dates and numbers are predefined entities, which makes such an exhaustive training unnecessary. Similarly, aniketbangar (TPI=235.17) has training phrases with many combinations of Indian locations, while defining an entity for those locations would yield a more succinct design.

The length of the training phrases (WPTP) may affect the chatbot understandability. Some chatbots define extraordinarily short training phrases, like fulfillment-multi-locale (0.5), which is a toy example to experiment with multi-language chatbots (English and French). The high outliers of WPTP in the dataset are only for Dialogflow chatbots and do not seem problematic.

High values of PPTP (parameters per training phrase) may point to inputs that are too demanding for the user. An extreme case is Dining-Out, a predefined Dialogflow chatbot for obtaining recommendations to dine out, with 8.33 PPTP. The chatbot demands data like location, dish, beverage, venue type, etc., in many training phrases. This may lead to users having to input long phrases or having to answer many follow-up prompts from the chatbot.

Related to user experience, high outliers of metric CNF may reveal chatbot understanding problems due to the existence of similar training phrases in different intents, which may confuse the chatbots. Overall, 17.09% of Dialogflow chatbots and 13.38% of Rasa chatbots have confusing phrases in different intents, with extreme cases such as covid-19-chatbot (10532), identity-cloning-toolkit (3461), Car (1606) and iLearn (1599). For example, several intents of Car have similar training phrases, such as *"turn down the heater for each seat in the car"* and *"turn off the heating in my car"*. Other chatbots with confusing training phrases are small-talk-rasa-stack (*"I am very bored"* / *"I'm bored of you"*), googleChallenge (*"What is the time duration for completing Masters in Artificial Intelligence?"* / *"Completion period for masters in AI?"*), Dining-Out (*"now cafe"* / *"find cafe"*), and BikeShop (*"Can you fix my road bike?"* / *"Can you service my bike?"*). High values of CNF are more frequent in chatbots with many intents, as this increases the probability of having similar intents. Sometimes, high CNF values stem from an incorrect interpretation of what training entails. For example, covid-19-chatbot sometimes uses tags (e.g., coronavirus, covid, corona) instead of full-fledged phrases, which may hamper correct intent recognition since these tags overlap in many intents. This way, CNF helps detecting intents that a chatbot may mismatch, without resorting to intensive dynamic testing. High CNF values can also be an indicator that some intents could be merged. For example, chatbot covid-19-chatbot has several intents to express that the user has tested positive in covid, and these intents could be merged.

7.4.5 Vocabulary. Entity metrics provide insights on the chatbot vocabulary and may uncover problems in it.

In Table 6, the median and low cut-off value of metrics LPE, SPL and WL is zero in Dialogflow. This is because more than a half of the Dialogflow chatbots lack entities, which can be considered normal. However, chatbots that define entities (ENT>0) but have few literals per entity (LPE) indicate

poorly defined entities. This is the case of the Dialogflow chatbot your_song, which defines entities with just one literal, e.g., for music genres, which may compromise a correct entity recognition. In other cases, chatbots may have entities but LPE is zero because the entities are defined via regular expressions or functions. This is the case for the 17 Rasa chatbots defining entities in the dataset.

In addition, WL helps detecting unusually long entity literals. As an example, the Dialogflow chatbot gordobbot defines entities for food origins or cooking utensils, and its WL value is 36. Inspecting the definition of these entities reveals a misconception on the use of entity literals, as instead of providing synonyms for literals, the definition provides explanatory phrases.

7.4.6 Answering RQ2. To answer RQ2, we have manually checked all chatbots with low or high metric outliers to assess if they have problems. Table 7 summarises the results. For each metric, columns 4 to 7 show the number of problematic Dialogflow and Rasa chatbots, and the percentage they represent in the set of outliers for the metric. The last two columns describe the problem found and its type. Detailed information on the discovered problems are available at https://github.com/ASYM0B/MetricsOutliers. Overall, we detected 16 types of problems implying design errors, usability problems, or poor designs that could be improved by re-designs. We refrain from reporting usability problems that require a subjective evaluation, like those related to negative tone in chatbot answers.

Table 7. Summary of problems detected in chatbots with high metric outliers or low metric values. Legend: n/a indicates an empty set of outliers; * indicates the percentage is taken on the whole dataset of chatbots.

Aspect	Metric	Value	# Problems, Outlier%				Problem description	Problem type
			Dialogflow		Rasa			
Size	INT	low	9	100%	3	100%	Incomplete/Toy chatbot	Incomplete design, Usability
		high	2	33.4%	5	55.6%	Redundant intents (when CNF high)	Re-design
Conversation	FLOW	low	2	100%	5	20.9%	Incomplete/Toy chatbot	Incomplete design, Usability
		high	0	0%	7	46.7%	Repeated or redundant flows	Design error, Re-design
	PATH	low	2	100%	6	54.6%	Incomplete/Toy chatbot	Incomplete design, Usability
		high	0	0%	16	76.2%	Repeated or redundant paths	Design error, Re-design
	FPATH	low	n/a	n/a	n/a	n/a	Incomplete/Toy chatbot	Incomplete design, Usability
		high	1	5.3%	15	71.5%	Repeated or redundant paths	Design error, Re-design
	CL	high	0	0%	1	9.1%	Long conversation (hard to complete)	Usability
		high	0	0%	9	81.9%	Error in conversation design	Design error
Outputs	VPOP	high	1	100%	2	25%	Missing punctuation signs	Usability
	CPO	high	0	0%	4	33.4%	Long responses (>280 chars)	Usability, Deployability
	WPO	high	0	0%	5	50%	Long responses (>50 words)	Usability, Comprehensibility
	READ	high	0	0%	5	50%	Long reading times (>30 secs)	Usability, Efficiency
Inputs	TPI	low	11	52.4%	4	100%	Intents poorly trained (≤4 phrases)	Usability
			13	11.2%*	45	31.7%*	Intents without training phrases	Incomplete design, Re-design
		high	2	25%	14	77.8%	Repeated ot redundant phrases	Re-design
	WPTP	low	7	100%	2	100%	Bad quality of training set	Usability, Incomplete design
	CNF	high	7	35%	5	27.8%	Confusing intents	Usability
Vocabulary	LPE	low	3	100%	n/a	n/a	Ill-defined entities (when ENT>0)	Usability
	WL	high	1	50%	n/a	n/a	Bad use of entity literals	Design error

As the table shows, low values for INT, FLOW and PATH typically revealed incomplete or toy chatbots. Conversely, high INT values, in combination with high CNF values, uncovered redundant intents. Some chatbots with extremely high values for FLOW, PATH and FPATH had repeated or redundant flows or paths that could be deleted or simplified to improve the design quality. Inspecting chatbots with high CL values uncovered design errors (concatenated conversation flows) or conversations unlikely to occur in practice (e.g., a chit-chat chatbot that expects the user to jump over unrelated topics in long conversation flows).

Regarding chatbot outputs, some high VPOP values were due to missing punctuation signs, while large values of CPO, WPO and READ can cause usability problems (inability to deploy the chatbot

in certain channels, or sentences that are long or difficult to understand). The table shows the problematic chatbots taking thresholds of 280 characters (Twitter's maximum message size), 50 words and 30 seconds. Still, other thresholds can be used depending on the target channel, chatbot domain and audience [33].

Regarding chatbot inputs, low values of TPI revealed some poorly trained intents, which either could be removed or were indicative of incomplete designs. We set the bar on 4 training phrases, but other thresholds are possible (e.g., 10 as recommended in [1]). The table also shows the chatbots with empty intents (i.e., the minimum value of TPI is zero for any of the chatbot intents). Other problems discovered when checking metric outliers were the presence of redundant phrases which only varied in their parameter values (high TPI), low-quality training phrases made of one or two words (low WPTP), and overlapping intents with many similar training phrases (high CNF).

Finally, the vocabulary metrics only apply to Dialogflow. They helped uncovering under-defined entities (low LPE) and erroneous literals (high WL).

Overall, from the set of chatbots in the dataset with metric outliers, the percentage of those with problems range from 5.3% to 100% depending on the metric and technology (more than 50% in average). Therefore, we can answer RQ2 positively, since a substantial amount of chatbots with outliers in their metric values presented problems.

7.5 RQ3: Chatbot clustering

To answer this research question, we performed a similar experiment to the one in [4], which was directed to evaluate a clustering method for meta-models.

First, one voluntary manually labelled the 259 chatbots in the dataset with their domain, such as "hotels", "basic conversations" or "food". Each chatbot could be assigned several labels (e.g., a chatbot for searching restaurants could be labelled with both "search" and "restaurant"). This manual labelling yielded 43 clusters, 6 of them with a single chatbot (so-called singleton clusters). Table 8 characterises the manual clusters, which have sizes ranging from 1 to 32 chatbots, with an average size of 7.7 and median 5. Further details of the clusters are available at https://github.com/ASYM0B/SemanticClusteringEvaluation.

Table 8. Comparison of manual clustering and best vocabulary-based clustering.

	Manual clustering	Voc.-based clustering
#Clusters	43	29
Max labels per chatbot	3	1
#Singleton clusters	6	3
Size of smallest cluster	1	1
Size of largest cluster	32	43
Average cluster size	7.7	9.2
Median cluster size	5	15

Next, we applied our automated vocabulary-based clustering (described in Section 5.2) to the same chatbots. We used different linkage methods and distance thresholds (from 0.01 to 0.99 with 0.01 increments) to form the clusters. Then, for each combination of linkage method and distance threshold, we computed the balanced accuracy of the returned clusters as we explain below. Table 8 summarises the vocabulary-based clustering with the best balanced accuracy (0.664). It yields 29 clusters containing between 1 and 43 chatbots (9.2 chatbots in average, median of 15, and 3 singleton clusters). An inspection of the clusters shows that they group chatbots about hotels, insurance and banking, news, restaurants, food ordering, time zones, shopping, education, health, songs, and weather, among others.

In the following, we explain our method to identify the vocabulary-based clustering configuration with the most accurate result. Specifically, we used the metric *balanced accuracy* to quantify the percentage of chatbot pairs correctly sharing or not a cluster, taking the manual clusters as the ground truth. For this purpose, given a set of (manually or automatically) clustered chatbots $C = \{c_1, ..., c_n\}$, we built its clustering matrix M_C as follows:

- $M_{Cij} = 1$ if c_i and c_j belong to a same cluster
- $M_{Cij} = 0$ otherwise

Then, we compared the manual clustering matrix M with each vocabulary-based clustering matrix V[19], counting how many chatbot pairs (c_i, c_j) belonged to one of four possible sets:

- Correctly grouped: $CG = |\{(c_i, c_j) \mid V_{ij} = M_{ij} = 1\}|$ is the number of *true positives*.
- Correctly separated: $CS = |\{(c_i, c_j) \mid V_{ij} = M_{ij} = 0\}|$ is the number of *true negatives*.
- Incorrectly grouped: $IG = |\{(c_i, c_j) \mid V_{ij} = 1 \wedge M_{ij} = 0\}|$ is the number of *false positives*.
- Incorrectly separated: $IS = |\{(c_i, c_j) \mid V_{ij} = 0 \wedge M_{ij} = 1\}|$ is the number of *false negatives*.

Next, we calculated balanced accuracy as:

$$BA = \frac{1}{2} \cdot \left(\frac{CS}{CS + IG} + \frac{CG}{CG + IS} \right)$$

The second term of the formula ($\frac{CG}{CG+IS}$) is the percentage of correct 1's returned by the clustering. The first term ($\frac{CS}{CS+IG}$) is the percentage of correct 0's returned by the clustering. For non-overlapping clusters, BA is just normal recall, because a 100% recall of 1's implies a 100% recall of 0's, and vice versa. However, for overlapping clusters, a 100% recall of 0's does not imply a 100% recall of 1's. This is why we balance both terms in the formula for BA.

We use balanced accuracy to quantify the accuracy of our clustering approach because the matrices M and V are sparse (i.e., unbalanced): M has 94% of 0's, and the most accurate V has 93% of 0's. Instead, using precision would give more importance to guessing 0's than to guessing 1's. For example, a V matrix that only contains 0's would obtain a precision close to 1, which does not fit our purposes. The formula BA eliminates this bias.

Figure 18 depicts the value of BA for each linkage method and distance threshold. As a summary, Table 9 shows the distance threshold that permits obtaining the highest balanced accuracy for each linkage method. The best accuracy (0.664) was achieved using weighted linkage and a threshold of 0.97. Even though all linkage methods achieved similar accuracy, the weighted and average methods performed slightly better than the rest. For this reason, our tool uses the weighted linkage method.

Table 9. Best balanced accuracy for each linkage method.

Linkage method	Distance threshold	Best bal. accuracy
Single	0.71	0.615
Complete	0.97	0.619
Average	0.96	0.641
Centroid	0.67	0.607
Weighted	0.97	**0.664**
Median	0.73	0.622

[19] The vocabulary-based clustering matrices V represent a reflexive, symmetric, transitive relation between chatbots, since the clusters are disjoint. Instead, the manual clustering matrix M may not be transitive, since we allowed chatbots with several manual labels. This means that our vocabulary-based clustering will not be able to produce the clusters of the manual matrix, if the manual clusters are not disjoint.

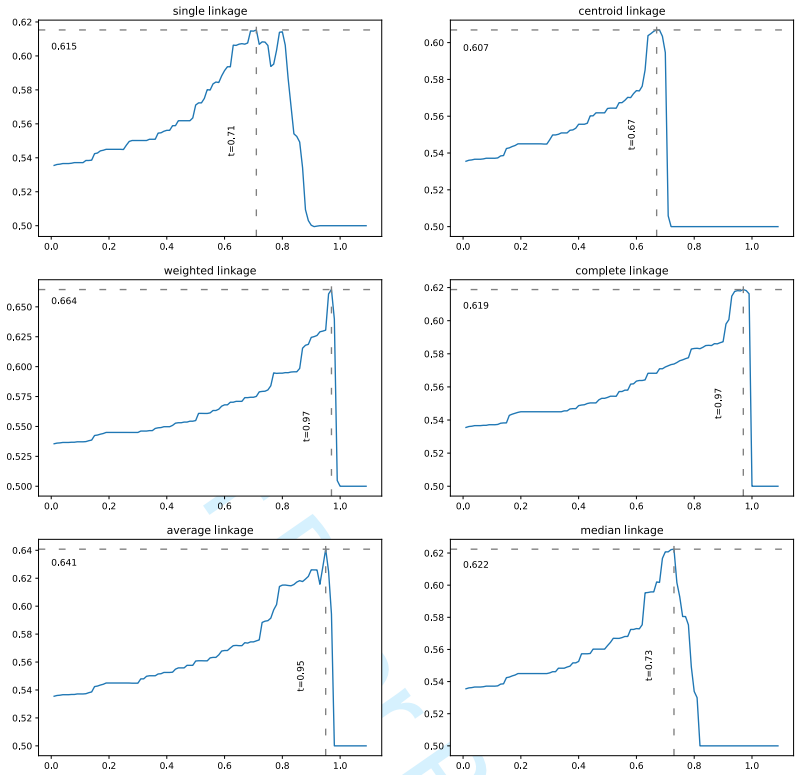


Fig. 18. Balanced accuracy of vocabulary-based clustering depending on the selected linkage method and distance threshold.

Finally, we have compared our clustering approach with three baselines: a clustering matrix containing only 0's, another containing only 1's, and a third one with binary values randomly generated. The balanced accuracy of the first two cases is 0.5, and the one of the third case is 0.502. In comparison, the balanced accuracy of our clustering approach was 0.664, which amounts to an increase of 34% with respect to the baselines.

7.5.1 Answering RQ3. Overall, we can answer *RQ3* positively: vocabulary-based clustering can create meaningful groups of semantically related chatbots (e.g., groups of chatbots about hotels, restaurants, or weather, among others). To have a quantitative assessment, we measured the fraction of correctly clustered chatbot pairs using balanced accuracy, and compared against three baselines obtaining a substantially higher balanced accuracy (0.664 vs 0.5).

7.6 Threats to validity

Next, we discuss threats to the internal and external validity of our evaluation. Internal validity is the extent to which the findings of an experiment truly represent a cause-and-effect relationship. External validity is concerned with the extent to which the results of a study can be generalised.

7.6.1 Internal validity. Regarding RQ1, a limitation of our evaluation is the use of custom-made importers from existing platforms into Conga. In particular, Rasa permits programming some aspects of chatbots in several ways. For example, one may train a chatbot on the fly instead of using

training phrases, or change the conversation flow using Python. Our importer does not handle such code-based functionality variants, which may affect the metric values.

Regarding RQ2, we detected evident problems in some chatbots using metrics. However, we did not confirm those problems with the chatbot authors. Even if this would be desirable, according to their profile, the authors are frequently hobbyist programmers or students, which may hamper confirming the problems. We did not perform any cleaning (e.g., of small-sized chatbots) of chatbots in the dataset, utilizing all chatbots we found. As a consequence, the dataset may contain unfinished chatbots. This is intentional since we argue that metrics can serve for quality assurance during chatbot development, and so, we want our metrics to detect problems related to incomplete designs.

With regards to RQ3, one author of the paper labelled the chatbots manually. To avoid any bias, this labelling was performed before the execution of the vocabulary-based clustering.

7.6.2 External validity. We mitigate external threats by defining our metrics and clustering methods on a neutral notation (Conga) which has been designed out of 15 well-known chatbot development platforms, and using a dataset of third-party chatbots. While the use of Conga increases our confidence on the applicability of our proposal to the most common chatbot development platforms, we cannot claim generality as there may be other platforms supporting specific chatbot features or idioms not captured by Conga.

Regarding RQ1, the limited size of the dataset does not allow claiming differences or similarities between the technology or source of the chatbots, for which we would need a larger scale experiment. Instead, our goal was to hint at the usefulness of the proposed static chatbot metrics.

Related to RQ2, while metrics can detect problems in chatbots, problems need to be confirmed by inspection. Not all possible problems in a chatbot can be detected by metrics, for which a combination of static analysis and testing would be desirable. Also for RQ2, the results in Table 7 show a higher number of problems in Rasa chatbots than in Dialogflow chatbots. However, these problems cannot be directly attributed to the usage of a particular technology, for which a wider study would be needed.

As for RQ3, we show that our vocabulary-based clustering has better accuracy than three baseline methods. This experiment considers 259 third-party chatbots to avoid any bias and promote generality. However, using a different set of chatbots may result in a different accuracy value.

8 CONCLUSION AND FUTURE WORK

Chatbots are increasingly relevant nowadays, so techniques for assessing, comparing and clustering chatbots before their deployment are required. To this aim, we have proposed a suite of metrics, along with two clustering mechanisms, applicable over heterogeneous chatbot designs independently of their implementation technology. Our proposal is supported by the tool Asymob, which can be used both as a web platform and via its REST API for integration within specific chatbot platforms or development processes. We have evaluated the approach on a dataset of 259 chatbots, demonstrating the usefulness of the metrics – for chatbot quality assurance and detecting problems – and the clustering.

In the future, we would like to study how developers perceive and value our metrics by conducting qualitative studies, such as surveys. We also plan to study correlations between our metrics, with other development metrics like effort, and with usability metrics collected dynamically. Ultimately, we would like to derive metric thresholds to guide developers during chatbot construction. We would like to conduct larger-scale studies to better understand the type of errors in chatbots, and their links to the technologies used. For this, we plan to use a combination of metrics and static design analysis. We are also interested in supporting other chatbot representations for semantic clustering. These may include the use of embeddings like Word2Vec or pre-trained language models

like BERT [13]. At the tool level, we are working on presenting warnings about outliers, and on a graphical editor to inspect chatbot designs. Finally, finding chatbots is cumbersome with current search engines. For this reason, we would like to propose a dedicated crawler and search engine for chatbots, able to accurately find chatbot projects for a given technology.

DATA AND CODE AVAILABILITY

The data and code that support the findings of this paper are openly available at:

- Deployed Asymob service: http://miso.ii.uam.es/asymobService
- Asymob core code: https://github.com/PabloCCanizares/asymob
- Datasets and experiment results: https://github.com/asym0b

ACKNOWLEDGMENTS

This work has been funded by the Spanish Ministry of Science (projects TED2021-129381B-C21 and PID2021-122270OB-I00) and the Madrid region (P2018/TCS-4314).

REFERENCES

[1] Ahmad Abdellatif, Khaled Badran, Diego Costa, and Emad Shihab. 2022. A Comparison of Natural Language Understanding Platforms for Chatbots in Software Engineering. *IEEE Trans. Software Eng.* 48, 8 (2022), 3087–3102.

[2] Shumail Arshad and Christos Tjortjis. 2016. Clustering Software Metric Values Extracted from C# Code for Maintainability Assessment. In *Proc. 9th Hellenic Conf. on Artificial Intelligence*. ACM, 24:1–24:4.

[3] Önder Babur, Loek Cleophas, and Mark van den Brand. 2016. Hierarchical Clustering of Metamodels for Comparative Analysis and Visualization. In *Proc. 12th Eur. Conf. on Modelling Foundations and Applications (LNCS, Vol. 9764)*. Springer, 3–18.

[4] Francesco Basciani, Juri Di Rocco, Davide Di Ruscio, Ludovico Iovino, and Alfonso Pierantonio. 2016. Automated Clustering of Metamodel Repositories. In *Proc. 28th Int. Conf. on Advanced Information Syst. Eng. (LNCS, Vol. 9694)*. Springer, 342–358.

[5] Botium. 2023. https://www.botium.ai/. last access in 2023.

[6] Josip Bozic and Franz Wotawa. 2019. Testing Chatbots Using Metamorphic Relations. In *Proc. 31st IFIP WG 6.1 Int. Conf. on Testing Softw. and Syst. (LNCS, Vol. 11812)*. Springer, 41–55.

[7] Sergio Bravo-Santos, Esther Guerra, and Juan de Lara. 2020. Testing Chatbots with Charm. In *Proc. 13th Int. Conf. on Quality of Information and Communications Technology (CCIS, Vol. 1266)*. Springer, 426–438.

[8] Marc Brysbaert. 2019. How Many Words Do We Read per Minute? A Review and Meta-Analysis of Reading Rate. *Journal of Memory and Language* 109 (2019), 104047.

[9] Pablo C. Cañizares, Sara Pérez-Soler, Esther Guerra, and Juan de Lara. 2022. Automating the Measurement of Heterogeneous Chatbot Designs. In *Proc. 37th ACM/SIGAPP Symposium On Applied Computing*. ACM, 1–8.

[10] D. Cer, Y. Yang, S.-yi Kong, N. Hua, N. Limtiaco, R.S. John, N. Constant, M. Guajardo-Céspedes, S. Yuan, C. Tar, et al. 2018. Universal Sentence Encoder. *arXiv preprint arXiv:1803.11175* (2018). arXiv:1803.11175

[11] Chatbottest. 2023. https://chatbottest.com/. last access in 2023.

[12] David Coniam. 2014. The Linguistic Accuracy of Chatbots: Usability from an ESL Perspective. *Text & Talk* 34, 5 (2014), 545–567.

[13] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *CoRR* abs/1810.04805 (2018), 16 pages.

[14] Dialogflow. 2023. https://dialogflow.com/. last access in 2023.

[15] Márcio Braga dos Santos, Ana Paula Carvalho Cavalcanti Furtado, Sidney C. Nogueira, and Diogo Dantas Moreira. 2020. OggyBug: A Test Automation Tool in Chatbots. In *Proc. 5th Brazilian Symposium on Systematic and Automated Softw. Testing*. ACM, 79–87.

[16] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. 1996. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *Proc. 2nd Int. Conf. on Knowledge Discovery and Data Mining*. AAAI Press, 226–231.

[17] Norman E. Fenton and Shari Lawrence Pfleeger. 1996. *Software metrics - a practical and rigorous approach (2. ed.)*. International Thomson.

[18] R. Flesch. 1948. A New Readability Yardstick. *J. of Applied Psychology* 32, 3 (1948), 221.

[19] Gartner. 2022. Competitive Landscape: Conversational AI Platform Providers. https://info.kore.ai/competitive-landscape-conversational-ai-platform-providers. last access in 2023.

[20] ISO 9241-11. 1998. Ergonomic requirements for office work with visual display terminals (VDTs). Part II guidance on usability.

[21] Anil K. Jain and Richard C. Dubes. 1988. *Algorithms for clustering data.* Prentice-Hall, Englewood Cliffs, NJ, USA.

[22] Jiepu Jiang and Naman Ahuja. 2020. Response Quality in Human-Chatbot Collaborative Systems. In *Proc. 43rd Int. ACM SIGIR Conf. on Research and Development in Information Retrieval.* ACM, 1545–1548.

[23] Shinji Kawaguchi, Pankaj K. Garg, Makoto Matsushita, and Katsuro Inoue. 2006. MUDABlue: An Automatic Categorization System for Open Source Repositories. *J. Syst. Softw.* 79, 7 (2006), 939–953.

[24] Adrian Kuhn, Stéphane Ducasse, and Tudor Gîrba. 2007. Semantic Clustering: Identifying Topics in Source Code. *Inf. Softw. Technol.* 49, 3 (2007), 230–243.

[25] Thomas K Landauer, Peter W. Foltz, and Darrell Laham. 1998. An Introduction to Latent Semantic Analysis. *Discourse Processes* 25, 2–3 (1998), 259–284.

[26] Carlene Lebeuf, Margaret-Anne D. Storey, and Alexey Zagalsky. 2018. Software Bots. *IEEE Softw.* 35, 1 (2018), 18–23.

[27] Lex. 2023. https://aws.amazon.com/en/lex/. last access in 2023.

[28] Chia-Wei Liu, Ryan Lowe, Iulian Serban, Mike Noseworthy, Laurent Charlin, and Joelle Pineau. 2016. How NOT to Evaluate your Dialogue System: An Empirical Study of Unsupervised Evaluation Metrics for Dialogue Response Generation. In *Proc. 2016 Conf. on Empirical Methods in Natural Language Processing.* ACL, 2122–2132.

[29] José-María López-Morales, Pablo C. Cañizares, Sara Pérez-Soler, Esther Guerra, and Juan de Lara. 2022. Asymob: A Platform for Measuring and Clustering Chatbots. In *Proc. 44th Int. Conf. on Soft. Eng.* ACM, 1–5.

[30] Jonathan I. Maletic and Andrian Marcus. 2000. Using Latent Semantic Analysis to Identify Similarities in Source Code to Support Program Understanding. In *Proc. 12th IEEE Int. Conf. on Tools with Artificial Intelligence.* IEEE CS, 46–53.

[31] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. In *Advances in Neural Information Processing Systems*, Vol. 26. Curran Associates, Inc.

[32] Sebastian Möller, Roman Englert, Klaus-Peter Engelbrecht, Verena Vanessa Hafner, Anthony Jameson, Antti Oulasvirta, Alexander Raake, and Norbert Reithinger. 2006. Memo: Towards Automatic Usability Evaluation of Spoken Dialogue Services by User Error Simulations. In *Proc. 9th Int. Conf. on Spoken Language Processing.* ISCA, 1786–1789.

[33] Robert J. Moore and Raphael Arar. 2018. Conversational UX Design: An Introduction. In *Studies in Conversational UX Design.* Springer, 1–16.

[34] Robert J. Moore and Raphael Arar. 2019. *Conversational UX Design: A Practitioner's Guide to the Natural Conversation Framework.* ACM, New York, NY, USA.

[35] Robert J. Moore, Eric Young Liu, Saurabh Mishra, and Guang-Jie Ren. 2020. Design Systems for Conversational UX. In *Proc. 2nd Conf. on Conversational User Interfaces.* ACM, 45:1–45:4.

[36] Quim Motger, Xavier Franch, and Jordi Marco. 2023. Software-Based Dialogue Systems: Survey, Taxonomy and Challenges. *ACM Comput. Surv.* 55, 5 (2023), 91:1–91:42.

[37] Phuong Thanh Nguyen, Juri Di Rocco, Ludovico Iovino, Davide Di Ruscio, and Alfonso Pierantonio. 2021. Evaluation of a Machine Learning Classifier for Metamodels. *Softw. Syst. Model.* 20, 6 (2021), 1797–1821.

[38] OpenAI. 2023. https://openai.com/. last access in 2023.

[39] Pandorabots. 2023. https://home.pandorabots.com/. last access in 2023.

[40] Dijana Peras. 2018. Chatbot Evaluation Metrics: Review Paper. In *Proc. 33rd Int. Scientific Conf. on Economic and Social Development.* Varazdin Development and Entrepreneurship Agency, 89–97.

[41] Sara Pérez-Soler, Esther Guerra, and Juan de Lara. 2018. Collaborative Modeling and Group Decision Making Using Chatbots in Social Networks. *IEEE Softw.* 35, 6 (2018), 48–54.

[42] Sara Pérez-Soler, Esther Guerra, and Juan de Lara. 2020. Model-Driven Chatbot Development. In *Proc. 39th Int. Conf. on Conceptual Modeling (LNCS, Vol. 12400).* Springer, 207–222.

[43] Sara Pérez-Soler, Esther Guerra, and Juan de Lara. 2021. Creating and Migrating Chatbots with Conga. In *Proc. 43rd IEEE/ACM Int. Conf. on Soft. Eng.: Companion Proceedings.* IEEE, 37–40.

[44] Sara Pérez-Soler, Sandra Juárez-Puerta, Esther Guerra, and Juan de Lara. 2021. Choosing a Chatbot Development Tool. *IEEE Softw.* 38, 4 (2021), 94–103.

[45] Emily Pitler and Ani Nenkova. 2008. Revisiting Readability: A Unified Framework for Predicting Text Quality. In *Proc. Conf. on Empirical Methods in Natural Language Processing.* ACL, USA, 186–195.

[46] Martin Porter and Richard Boulton. 2001. *The English (Porter2) stemming algorithm.* http://snowball.tartarus.org/algorithms/english/stemmer.html

[47] Nicole M. Radziwill and Morgan C. Benton. 2017. Evaluating Quality of Chatbots and Intelligent Conversational Agents. *CoRR* abs/1704.04579 (2017), 21. arXiv:1704.04579 http://arxiv.org/abs/1704.04579

[48] Rasa. 2023. https://rasa.com/. last access in 2023.

[49] Ranci Ren, John W. Castro, Silvia Teresita Acuña, and Juan de Lara. 2019. Evaluation Techniques for Chatbot Usability: A Systematic Mapping Study. *Int. J. Softw. Eng. Knowl. Eng.* 29, 11&12 (2019), 1673–1702.

[50] Lior Rokach. 2010. *A survey of Clustering Algorithms.* Springer US, Boston, MA, 269–298.

[51] Peter J. Rousseeuw. 1987. Silhouettes: A Graphical Aid to the Interpretation and Validation of Cluster Analysis. *J. of Computational and Applied Mathematics* 20 (1987), 53–65.

[52] Claude Sammut and Geoffrey I. Webb. 2010. TF–IDF. In *Encyclopedia of Machine Learning*. Springer US, Boston, MA, 986–987.

[53] Qusay Idrees Sarhan, Bestoun S. Ahmed, Miroslav Bures, and Kamal Z. Zamli. 2022. Software Module Clustering: An In-Depth Literature Analysis. *IEEE Trans. Software Eng.* 48, 6 (2022), 1905–1928.

[54] Emanuel A. Schegloff. 2007. *Sequence Organization in Interaction.* Cambridge University Press.

[55] João Sedoc, Daphne Ippolito, Arun Kirubarajan, Jai Thirani, Lyle Ungar, and Chris Callison-Burch. 2019. Chateval: A Tool for Chatbot Evaluation. In *Proc. 2019 Conf. of the North American Chapter of the Association for Computational Linguistics (Demonstrations).* ACL, 60–65.

[56] Amir Shevat. 2017. *Designing bots: Creating conversational experiences.* O'Reilly.

[57] Mark Shtern and Vassilios Tzerpos. 2012. Clustering Methodologies for Software Engineering. *Adv. Softw. Eng.* 2012 (2012), 792024:1–792024:18.

[58] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. 2013. Recursive Deep Models for Semantic Compositionality over a Sentiment Treebank. In *Proc. 2013 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 1631–1642.

[59] Dave Steinberg, Frank Budinsky, Ed Merks, and Marcelo Paternostro. 2008. *EMF: Eclipse Modeling Framework, 2nd edition.* Pearson Education.

[60] Marilyn A. Walker, Diane J. Litman, Candace A. Kamm, and Alicia Abella. 1997. PARADISE: A Framework for Evaluating Spoken Dialogue Agents. In *Proc. 35th Annual Meeting of the Association for Computational Linguistics and 8th Conf. of the Eur. Chapter of the Association for Computational Linguistics.* Morgan Kaufmann Publishers / ACL, 271–280.

[61] Watson. 2023. https://www.ibm.com/cloud/watson-assistant/. last access in 2023.

[62] Joseph Weizenbaum. 1966. ELIZA - A Computer Program for the Study of Natural Language Communication between Man and Machine. *Commun. ACM* 9, 1 (1966), 36–45.

[63] Luxun Xu, Vagelis Hristidis, and Nhat X.T. Le. 2019. Clustering-Based Summarization of Transactional Chatbot Logs. In *Proc. 2019 IEEE Int. Conf. on Humanized Computing and Communication.* IEEE, 60–67.

[64] Yizhe Zhang, Siqi Sun, Michel Galley, Yen-Chun Chen, Chris Brockett, Xiang Gao, Jianfeng Gao, Jingjing Liu, and Bill Dolan. 2020. DIALOGPT: Large-Scale Generative Pre-training for Conversational Response Generation. In *Proc. 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations.* Association for Computational Linguistics, 270–278.

[65] S. Zhong, T.M. Khoshgoftaar, and N. Seliya. 2004. Analyzing Software Measurement Data with Clustering Techniques. *IEEE Intelligent Systems* 19, 2 (2004), 20–27. https://doi.org/10.1109/MIS.2004.1274907

Automating the Measurement of Heterogeneous Chatbot Designs

Pablo C. Cañizares
Universidad Autónoma de Madrid
Madrid, Spain
pablo.cerro@uam.es

Sara Pérez-Soler
Universidad Autónoma de Madrid
Madrid, Spain
sara.perezs@uam.es

Esther Guerra
Universidad Autónoma de Madrid
Madrid, Spain
esther.guerra@uam.es

Juan de Lara
Universidad Autónoma de Madrid
Madrid, Spain
juan.delara@uam.es

ABSTRACT

Chatbots are being increasingly used to provide a natural language interface to all kinds of software services. However, while there are many platforms and tools for chatbot development, they typically lack support to statically measure properties of the designed chatbots, as indicators of their size, complexity, quality or usability, and facilitating comparison.

To attack this problem, in this paper we propose a suite of 20 metrics for chatbot designs. The metrics are defined on a neutral chatbot design language, becoming independent of the implementation platform. We have developed a tool, called Asymob, which supports the translation of chatbots defined in several platforms into this neutral format to perform the measurements. As a proof-of-concept, we evaluate the metrics over a collection of Dialogflow and Rasa chatbots from several sources and open-source repositories. Our metrics helped detecting quality issues statically, and served as a basis for comparing chatbots from different origins and built using different technologies.

CCS CONCEPTS

• **Human-centered computing** → **Natural language interfaces**; • **General and reference** → **Metrics**; • **Social and professional topics** → **Quality assurance**;

KEYWORDS

Chatbot design, metrics, quality assurance

ACM Reference Format:
Pablo C. Cañizares, Sara Pérez-Soler, Esther Guerra, and Juan de Lara. 2022. Automating the Measurement of Heterogeneous Chatbot Designs. In *The 37th ACM/SIGAPP Symposium on Applied Computing (SAC '22), April 25–29, 2022, Virtual Event*. ACM, New York, NY, USA, Article 4, 8 pages. https://doi.org/10.1145/3477314.3507255

SAC '22, April 25–29, 2022, Virtual Event

ACM ISBN 978-1-4503-8713-2/22/04…$15.00
https://doi.org/10.1145/3477314.3507255

1 INTRODUCTION

Chatbots are becoming popular to access all sorts of services (e.g., banking, shopping, tourism, health) using conversation in natural language [36], and they are being increasingly used to assist in software engineering activities [17]. For this reason, many platforms are available for their construction [29], like Google's Dialogflow [12], Amazon Lex [18], IBM's Watson [38], the Microsoft bot framework [21], and many others by smaller companies and communities like Rasa [32], Pandorabots [26], or FlowXO [15].

While there are many chatbot implementation platforms, their support for chatbot quality assurance is limited [29]. Since chatbots are a kind of software, their construction should follow sound engineering principles. Some recent approaches [4, 5, 13] and tools [3] propose methods for testing chatbots. Dynamic testing is essential to ensure the quality of the resulting chatbot, but it requires having a functional, deployed chatbot; it demands high effort for creating testing phrases and oracles; and it is time-consuming.

In this paper, we propose the use of metrics as a tool to guide and control the quality of the chatbot throughout its development, becoming a complement to dynamic testing. Metrics are an accepted mechanism for assessing and controlling properties of software products and processes [14]. They are complementary to dynamic testing as they can be used at design time, even when the chatbot is not yet functional. They can discover issues (e.g., regarding the design complexity and size) which are not the target of dynamic testing, and can be used to trigger recommendations for the improvement of the chatbot design. However, to our knowledge, there is hardly any proposal for the practical application of metrics to chatbot designs.

We argue that static metrics for chatbots can be useful to detect potential problems related to user experience (e.g., complex conversation flows, hard-to-read chatbot answers); as indicators of chatbot complexity; to compare properties of heterogeneous chatbots; to discover chatbot commonalities and cluster similar chatbots; and to understand how different implementation platforms can impact on the chatbot design. Ultimately, the availability of metrics may have a notable impact on current bot development practices and tools, helping to increase the quality of chatbots.

To pursue this goal, we propose a suite of 20 static metrics for chatbot designs, and an accompanying tool called Asymob that supports their evaluation over heterogeneous chatbot implementations. To avoid reimplementing the metrics for every chatbot implementation platform, Asymob defines the metrics over a neutral design

notation called Conga [28], and provides importers from several platforms into Conga. We report on an evaluation applying the metrics over Dialogflow and Rasa chatbots from public repositories, and over predefined chatbots provided by the implementation platforms. Our experiment reveals quality issues in some chatbots, and shows that the metrics can serve as a basis for comparing chatbots from different sources and built using different technologies.

The rest of the paper is organized as follows. Section 2 explains the basics about chatbots. Section 3 revises related work. Section 4 proposes a suite of chatbot metrics over a neutral chatbot design notation. Section 5 describes tool support, and Section 6 evaluates the metrics over chatbots from several sources (predefined, open-source repositories) and technologies (Rasa, Dialogflow). Finally, Section 7 concludes and outlines lines for future work.

2 AN OVERVIEW OF CHATBOTS

Chatbots are conversational software systems with a natural language interface to existing services, like those in banking or shopping. Figure 1 shows a diagram with their typical working scheme. Normally, the user starts the interaction by providing an *utterance* – a phrase in natural language – (step 1) which the chatbot processes to give a proper response (step 6). The interaction can be using text (e.g., if the chatbot is embedded in a social network like Telegram[1]) or voice (e.g., if the chatbot is deployed on smart speakers like Amazon echo[2]). The processing of the user utterance involves a number of steps, which we detail next.

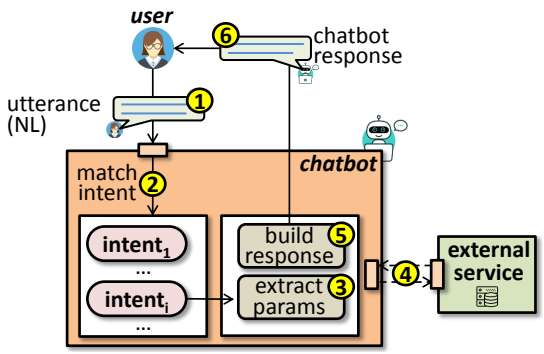


Figure 1: Chatbot working scheme.

Most chatbots are designed around a set of *intents*. These are conversation topics the chatbot aims at recognizing (step 2 in the figure), related to the offered functionality. Depending on the implementation platform, intents are defined either using *regular expressions* (e.g., as in Pandorabots [26]) or with *training phrases* that become interpreted using natural language processing (NLP). Additionally, intents can include *parameters* identifying relevant information pieces to be extracted from user utterances (step 3).

As an example, a chatbot for a cafeteria would define an intent to recognize the users' orders. This intent would be matched by phrases like *"I'd like a medium cappuccino"*, from which the chatbot would extract two parameters: the type of drink (*cappuccino*) and its size (*medium*). Parameters are typed by *entities*, which can be pre-existing in the platform (e.g., dates or numbers) or user-defined for a specific domain (e.g., the type and size of drinks). User-defined entities declare a list of literals (e.g., *medium* and *large* for the size of drinks) with their synonyms.

Upon receiving a user utterance, the chatbot matches the more likely intent, performs some predefined actions to handle the intent, such as accessing an external service (step 4), and composes a text/voice response (step 5) which may incorporate media elements (e.g., images, links) or widgets supported by the social network (e.g., buttons in Telegram). For example, the cafeteria chatbot may need to access an information system to check the availability of discounts and annotate the order, and the response may report the final price. If the chatbot lacks a matching intent for a user utterance, it can trigger a *fallback* intent to ask for clarification.

Typically, conversations are structured into *flows* that intertwine user utterances and chatbot responses. As an example, upon the user utterance *"I'd like a medium cappuccino"*, the chatbot may answer *"Would you like something to eat?"*, leading to a new user interaction (e.g., *"No thanks"*), and so on, according to the defined conversation flow within the chatbot design.

Moore and Arar [23] propose a classification of chatbots depending on their conversation style. *System-centric* chatbots answer user queries or interpret commands by means of 2-turn conversations (i.e., each user turn starts a new conversation and the chatbot lacks state). *Content-centric* chatbots act as an interface for FAQs, typically providing long document-like responses that may not be appropriate for voice-based interfaces or mobile devices with small screens. *Visual-centric* chatbots present buttons and other widgets to facilitate user interaction, in a style borrowed from mobile phones. Finally, *conversation-centric* chatbots mimic human conversations, offering conversation management utterances (e.g., "*What do you mean?*") and short responses. This latter type of chatbots are normally preferred because their conversation style suits a wider variety of devices and engages better in natural conversations.

3 RELATED WORKS ON CHATBOT QUALITY ASSESSMENT

Since the early days of conversational systems [39], researchers have proposed ways for evaluating their quality. For example, PARADISE [37] is an early framework based on the correlation of performance and user satisfaction.

Recently, the popularity of chatbots has raised concerns on proper conversational design. For example, IBM's Natural Conversation Framework [24] proposes conversation patterns [25] and design principles [23, 34]. The latter include guidelines like *recipient design* (i.e., allow multiple conversation paths for different user types), *minimization* (i.e., use concise chatbot answers), and *repair* (i.e., provide support for clarifications). In this line, *Chatbottest* [9] defines guidelines for chatbot design issues in categories like answering, error management, intelligence, navigation, personality and understanding. However, the burden is on the developer to manually test whether the chatbot fulfils the guidelines.

Literature reviews [27, 31] have also identified chatbot quality properties and ways to assess them. Radziwill and Benton align bot quality attributes with the ISO-9241 notion of usability [1] (efficiency, effectiveness and satisfaction), while Peras [27] adds further

[1] https://telegram.org/
[2] https://en.wikipedia.org/wiki/Amazon_Echo

categories (e.g., information retrieval, affect). Generally, the assessment of these quality properties relies on the dynamic execution of the chatbot, on collecting statistical data, or on subjective evaluations [10, 16, 22, 33]. Instead, our goal is to provide metrics that can be calculated automatically and statically on chatbot designs.

Other approaches assess quality via testing [7]. For instance, tools like Botium [3] or OggyBug [13] support test automation. Still, the developer has to provide a set of user utterances and expected chatbot answers within the envisioned conversation flows. To alleviate this burden, some works focus on the generation of challenging test user utterances [4, 5], e.g., via mutation of the training phrases defined for the intents. ChatEval [35] targets testing readability, which can be done statically by applying metrics (e.g., BLEU2 and average cosine similarity [19]) to the chatbot responses, and interactively by requiring the user to complete evaluation tasks. Instead, our goal is to provide complementary assessment mechanisms to testing in the form of metrics, which can be applied prior to deploying the chatbot and can reveal defects in the chatbot design.

As we will see in Section 4.2, some of our metrics profit from the work of the NLP community, which has developed useful readability metrics [19, 30]. For example, Pitler and Nenkova [30] combine lexical, syntactic and discourse features in a highly predictive model of human judgements of text readability. In this model, several linguistic features correlate best with readability judgments. In particular, the average number of verb phrases per sentence, the number of words in the text, and the vocabulary, among others, are associated with human assessments of how well a text is written. More specific to chatbots, Liu et al. [19] identify some weaknesses of metrics for chatbot responses, and provide recommendations for future chatbot evaluation systems.

Overall, we observe a lack of metrics to evaluate statically and automatically quality aspects of chatbot designs, independently of the chatbot implementation platform. Our goal is to fill this gap.

4 CHATBOT DESIGN METRICS

In order to provide a suite of metrics independent from the chatbot implementation technology, we propose using a neutral design notation to represent chatbots, over which the metrics can be computed. In this section, firstly, Section 4.1 introduces the chatbot design notation, and then, Section 4.2 details our proposed metrics.

4.1 A neutral notation for chatbot designs

Since our aim is to develop metrics for chatbot designs, we need a concrete notation over which to define the metrics. For this purpose, we rely on the chatbot neutral notation we proposed in [28], called Conga. We opt for this neutral notation because, as reported in [28], its definition is based on a thorough revision of 15 widely used chatbot development platforms. This means that the design concepts in Conga can be mapped from and to all these platforms. Hence, by defining the metrics over Conga, they become platform-agnostic as well as significant for many chatbot development platforms. As we will show in Section 5, another practical implication is that one can build importers from different platforms into Conga to perform the measurement of existing chatbots.

Figure 2 depicts the meta-model of Conga. It permits representing a chatbot by a Chatbot object, which contains a set of Intents, Entities, Actions performed by the bot, and conversation Flows. The notation supports multi-language chatbots, and so, each intent can declare a number of TrainingPhrases per definition language. The phrases may refer to Parameters, which are defined at the level of the intent. Parameters are typed either by predefined entities (enumeration PredefinedEntity) or user-defined Entity objects. Entities can be Simple, Regex (regular expressions, a change in this new version of the meta-model) or Composite, and for each language (EntityLanguage), they declare the set of literals and synonyms making up the entity. For example, a chatbot can declare a simple entity for drink sizes with literals small, medium and large in English, and additionally define synonyms regular for medium and big for large.

A chatbot can define one or more Actions of type Text, Image, HttpRequest, HttpResponse and Empty. The two first types are used to compose responses combining text and images. HttpRequest and HttpResponse allow configuring the communication of the chatbot with external services in the backend. The last action type Empty is a wildcard for other platform-specific actions, added in this new version of the meta-model to facilitate transformation between platform-specific definition into Conga (explained in Section 5.2).

Finally, the conversation flow between the chatbot and the users is modelled by Flow objects consisting of user and bot turns (classes BotInteraction and UserInteraction). The user turn refers to the intent to be recognized in the interaction (reference UserInteraction.intent). The bot turn specifies the actions that the bot has to perform (reference BotInteraction.actions).

4.2 A metrics suite for chatbot designs

We propose the suite of metrics for chatbot designs that Table 1 shows. All metrics measure internal attributes of chatbots. We considered three sources when designing the metrics:

- Some of them, like INT (the number of intents) or ENT (the number of user-defined entities), are calculated by taking statistics of concepts from the meta-model in Figure 2. According to [28], these concepts are common in chatbot development frameworks.
- Some other metrics have been adapted from the NLP literature [19, 30] to assess the readability of the chatbot responses or the complexity of the expected user utterances.
- Finally, we use the conversation design principles proposed in [23, 34], and Moore and Arar's classification of chatbots [23], to interpret the value of some metrics such as PATH (the number of conversation paths), FLOW (the number of conversation entry points) and WPOP (the number of words per bot output phrase).

The fourth column of Table 1 classifies the potential impact of the metrics on usability (as defined in the ISO 9241-11) [1] in terms of Effectiveness (i.e., accuracy and completeness with which users achieve their goals), efficiency (i.e., time and resources that users expend to achieve their goals) and Satisfaction (i.e., comfort and acceptability of use). We also classify metrics based on their target: either global design properties, or specific aspects of intents, entities or conversation flows. Non-global metrics can be computed per element (intent, entity, flow) or averaged for all elements of a kind.

4.2.1 Global metrics. We start introducing *global metrics*. These measure the number of intents (INT), entities (ENT) and flows (FLOW, PATH), and include understanding and user experience metrics (CNF, SNT).

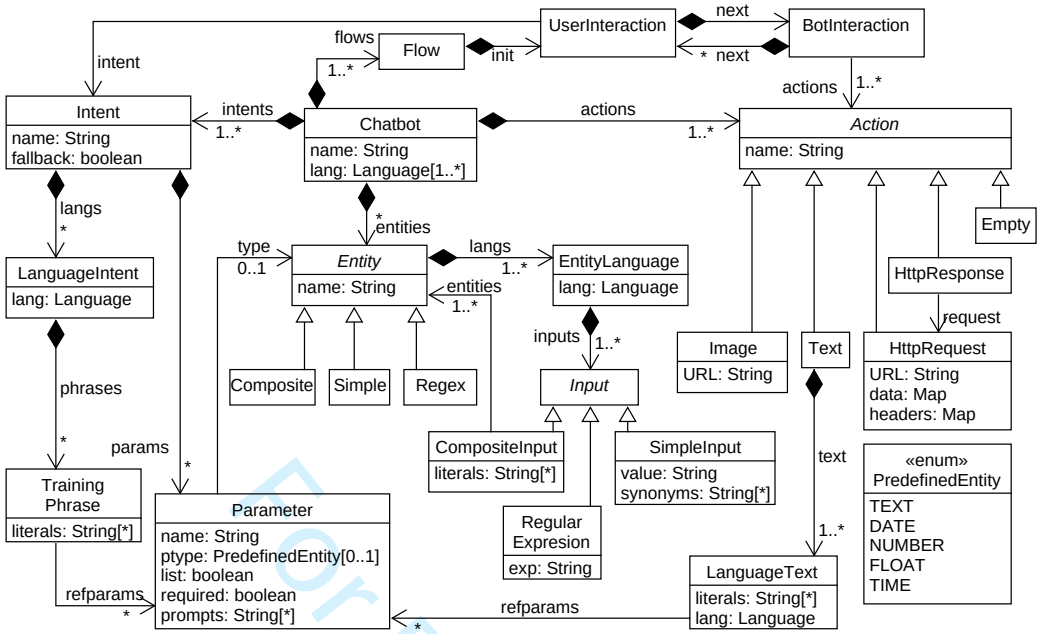


Figure 2: Meta-model for chatbot design (adapted from [28]).

Table 1: Metrics for chatbot designs. Column dimension uses abbreviations for Effectiveness, efficiencY and Satisfaction.

Metric	Description	Type	Dim
Global metrics			
INT	# intents	design size	E
ENT	# user-defined entities	vocabulary size	S
FLOW	# conversation entry points	conversation diversity	E
PATH	# different conversation flow paths	conversation complexity	S,Y
CNF	# confusing phrases [8]	bot understanding	E,S
SNT	# positive, neutral, negative output phrases [33]	user experience	S
Intent metrics			
TPI	# training phrases per intent	topic complexity	E,S
WPTP	# words per training phrase	topic complexity	Y
VPTP	# verbs per training phrase	topic complexity	S,Y
PPTP	# parameters per training phrase	topic complexity	E
WPOP	# words per output phrase	readability	S,Y
VPOP	# verbs per output phrase	readability	S
CPOP	# characters per output phrase	readability	S,Y
READ	reading time of the output phrases [6]	readability	Y
Entity metrics			
LPE	# literals per entity	vocabulary complexity	S
SPL	# synonyms per literal	vocabulary complexity	S
WL	word length	readability	Y,S
Flow metrics			
FACT	# actions per flow	bot response complexity	E,S
FPATH	# conversation flow paths	conversation complexity	S,Y
CL	conversation length	conversation complexity	Y

INT is an indicator of design size and functionality, since each intent contributes to functionality offered to the user. The larger INT is, the more functionality the bot offers, potentially impacting effectiveness. ENT measures the size of the chatbot vocabulary and the conversation topic diversity, which may affect satisfaction.

FLOW counts the number of conversation entry points for users, being an indicator of conversation diversity. Since each entry point might correspond to a functionality, FLOW may impact effectiveness. PATH measures conversation complexity. If PATH=FLOW, all conversations are linear, while if PATH>FLOW, some conversation splits into several paths. As an example, Figure 3 shows two small excerpts of chatbot designs conformant to the CONGA meta-model. The chatbot design (a) depicts a linear flow (i.e., FLOW=PATH=1). Linear flows enable simple conversations, typically request/reply, which may indicate a system-centric chatbot [23]. The chatbot design (b) shows a conversation flow that splits after the bot interaction (FLOW=1, PATH=2). This kind of flows permits non-linear conversations with multiple turns and dialogue alternatives, typical of conversation-centric chatbots [23].

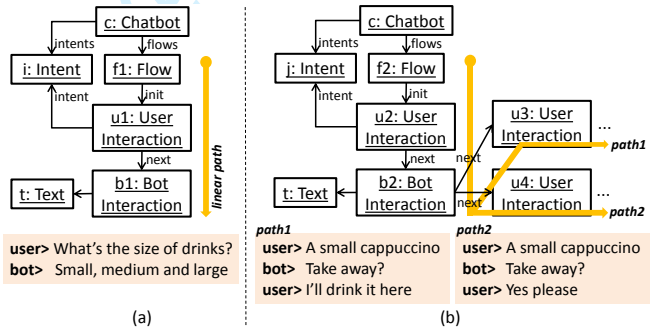


Figure 3: Chatbot design excerpts illustrating (a) a linear conversation flow, (b) a forked conversation flow.

The combined use of FLOW and PATH can help detecting deviations of some design principles. The *recipient* principle [34] advices to design for the target users, from experts (who may give all information at once) to novices (where the bot needs to prompt for more information). In turn, the *repair* principle [34] recommends supporting clarifications in the conversation, and multiple paths may be an indication of this. Moreover, having several paths per

flow potentially results in more natural conversations (impacting satisfaction) but less predictable for the user (likely impacting the user effort or efficiency).

The CNF metric measures the semantic distance between the training phrases of different intents, identifying similar phrases that may confuse the bot to make it identify a wrong intent [8]. Since this may cause errors, the metric is related to effectiveness and satisfaction.

Finally, SNT measures the sentiment of the chatbot output phrases, classifying them into positive, negative and neutral. This is related to satisfaction, since a bot that outputs mostly negative phrases may cause a negative user experience [33].

4.2.2 Intent metrics. Intent metrics measure quality properties of each intent with respect to the expected user utterances and the bot output phrases.

Related to user utterances, TPI counts the number of training phrases in the intent definition. The larger TPI is, the more precise the intent recognition might be, but this also may indicate a complex intent. WPTP measures the length of the training phrases in words. Long phrases are not adequate or even possible in social networks (e.g., Twitter restricts message length), and so, large WPTP values might be problematic. VPTP measures the number of verbs per training phrase. This is an indication of interaction complexity, since composite phrases with several verbs can be more difficult to elaborate for the user [30]. PPTP measures the number of information items (i.e., parameters) the user needs to provide, and the larger PPTP is, the more complex is the intent domain concept.

Regarding chatbot outputs, WPOP measures the number of words per bot output phrase. According to the *minimization* principle [23, 34], the bot answers should be concise. Large phrases are more difficult to understand and can be problematic in social networks. The latter is more concretely targeted by CPOP, as high values may require scrolling (e.g., in mobile devices) and long reading times (with the risk that the user does not complete the reading [23]). Long outputs are especially problematic for voice-based chatbots, since speaking takes longer than reading [23]. Hence, large CPOP values may decrease user satisfaction and efficiency. Similarly, VPOP is another indicator of the complexity of the chatbot responses, given by the number of verbs per output phrase. Finally, READ measures the expected reading time of the bot output responses (a metric related to efficiency). This is calculated as the ratio between the number of words per output phrase, and the number of words that an average person can read per minute [6].

4.2.3 Entity metrics. Entity metrics target user-defined entities representing domain concepts. LPE and SPL are indicators of the complexity of the concepts managed by a chatbot, impacting satisfaction. High LPE and SPL values signal elaborate concepts, but since SPL counts synonyms, a large number may improve recognition in user utterances (better satisfaction). A narrow vocabulary (low SPL) may constrain the way users communicate with the chatbot, and may lead to frustration if the chatbot does not recognize important parameters within user utterances. WL measures the length of words, and similar to CPOP, it contributes to readability and may impact user satisfaction and efficiency.

4.2.4 Flow metrics. Flow metrics consider features of the conversation flows. FACT measures the bot actions (presenting images, text, calling backends) in each conversation flow. The more actions, the more sophisticated tasks can be achieved. Moreover, rich controls help to reduce the user cognitive load and speed up the completion of the intended task. Hence, FACT may impact effectiveness and satisfaction. FPATH measures the number of possible paths per conversation flow. High values signal complex conversations (i.e., more natural-sounding but less predictable). The PATH global metric is calculated by adding up FPATH for each flow. Finally, CL measures the length of each path within a flow, as the number of bot and user turns. This is an indicator of conversation complexity. Longer paths require more time to complete – which affects efficiency – and are typical of conversation-centric chatbots [23].

5 ARCHITECTURE AND TOOL SUPPORT

We have developed a tool called Asymob supporting the automatic measurement of chatbot designs specified with Conga. Next, Section 5.1 presents the architecture of Asymob, including its main features, underlying technologies, and steps required to compute the metrics. Then, Section 5.2 details the conversion of chatbots implemented in two mainstream platforms into Conga.

5.1 Overview of Asymob

We have built a Java framework called Asymob for measuring chatbot designs. The framework is available at https://github.com/ASYM0B/tool. Asymob has a modular architecture that facilitates adding new metrics in multiple programming languages, like Java, Python and Perl. To support chatbots from different platforms, it relies on the neutral chatbot design notation Conga, introduced in Section 4.1. Hence, Asymob computes the metrics on Conga models, independently of any chatbot implementation platform. To measure chatbots from a specific platform, an importer from the platform into Conga must be provided. Currently, Asymob has importers from Dialogflow and Rasa. Section 5.2 will provide more details about these two importers.

To simplify the implementation of new metrics, Asymob supports third-party technologies such as Stanford CoreNLP [20], TensorFlow [2] and Deep Java Learning [11]. Stanford CoreNLP is an NLP library that Asymob uses to perform sentiment and syntactic analysis of the chatbot training and output phrases. The implementation of metrics SNT and VPTP make use of this library. Asymob relies on Deep Java Learning and TensorFlow to detect confusing phrases between intents, using the cosine similarity algorithm. The CNF metric is based on this algorithm.

Figure 4 shows the architecture of Asymob, and the steps to measure a chatbot design. First, the user selects a set of metrics (label 1) and a chatbot (label 2). Then, the Asymob core configures the *metrics database* with the selected metrics, and converts the provided chatbot into a Conga model (label 3, see Section 5.2). Next, the *metric engine* applies the selected metrics to the Conga model, and stores the results in a meta-data file (label 4). On request (label 5), the user can obtain a report with the results in several formats like plain text, Excel and LaTeX (label 6).

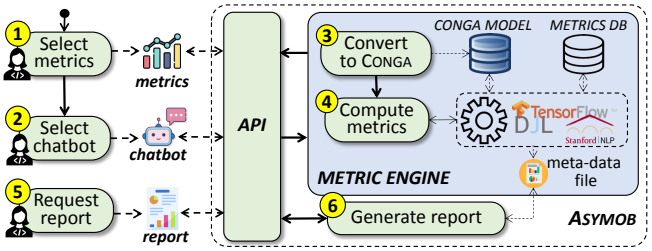


Figure 4: Architecture of ASYMOB.

5.2 Importing chatbots into CONGA

In the following, we provide details of the importers that we have built to convert chatbots from two representative and widely used chatbot platforms (Dialogflow and Rasa) into CONGA.

5.2.1 From Dialogflow to CONGA. Dialogflow is a low-code development platform to create chatbots using a graphical interface within the browser. Chatbots so defined can be exported as JSON files, which our importer is able to convert into CONGA models.

In the JSON-based representation of a Dialogflow chatbot, the file *Agent.json* describes global chatbot features, like its name, definition languages, or connection data to external services (the *webhook*). The latter include details such as the URL, headers, and authentication credentials. Our importer creates a CONGA Chatbot object using the agent name and languages, and an HttpRequest action with the webhook data.

Entities in Dialogflow can be predefined or user-defined. The latter are described either by a regular expression, a list of literals with synonyms, or a composite entity. Each user-defined entity becomes exported as a JSON file containing the entity name and configuration information (if it is a regular expression or a composite entity), and one file per definition language with the corresponding literals. Our importer converts these files into CONGA Entity objects.

Intents in Dialogflow have a name, training phrases, responses, parameters, and an indication of whether they are fallback or enable a webhook, among other features. Intents are exported into JSON files. For each intent definition file, our importer creates a CONGA Intent object with its Parameters and TrainingPhrases, as well as the necessary Actions to compose each response. We currently support text and image responses, and convert other custom responses into Empty actions. Anyhow, this does not affect the defined metrics.

Finally, Dialogflow controls the conversation flow via contexts. These can be input/output to intents, and can store relevant conversation state. Our importer uses the contexts and the responses of the related intents to generate CONGA Flow objects.

5.2.2 From Rasa to CONGA. Rasa is a framework to develop chatbots using Python, markdown and YAML. The definition of a Rasa chatbot comprises several files. The *config.yml* file defines configuration properties, like the chatbot language or the used NL prediction model. The *data/nlu.md* file contains training data to identify the intents correctly, with its entities and synonyms or regular expressions. As an example, the *data/nlu.md* file in Listing 1 defines an intent called order (lines 1–4). The parameters in the training phrases can be defined within brackets and followed by the entity name in parenthesis (e.g., [cappuccino](type)), or with curly brackets (e.g., [medium]{"entity": "size", "value": "medium"}).

```
1  ## intent:order
2  - I'd like a [medium]{"entity": "size", "value": "medium"} [cappuccino](type)
3  - I want a [small]{"entity": "size", "value": "small"} [latte](type)
4  - Can I order a [large]{"entity": "size", "value": "large"} [black](type) coffee?
5  ## synonym:small
6  - little
7  - short
8  ## synonym:medium
9  - regular
10 - median
11 ## synonym:large
12 - big
13 - extra
```

Listing 1: Example of data/nlu.md Rasa file

```
1  ## story1
2  * order
3      - utter_confirm_order
```

Listing 2: Example flow from data/stories.md Rasa file

The listing also declares synonyms for literals small (lines 5–7), medium (8–10) and large (11–13).

The file *domain.yml* defines the chatbot intents, entities and actions. Actions can be text, images, buttons, or custom actions defined in the Python file *actions.py*. Finally, the file *data/stories.md* specifies the conversation flows. Listing 2 shows a flow example, by which matching the intent order triggers the response utter_confirm_order.

We have built an importer that reads the chatbot language from the *config.yml* file and creates CONGA intents and entities from the *data/nlu.md* file, CONGA actions from the *domain.yml* file, and CONGA flows from the *data/stories.md* file. As in the case of Dialogflow, our importer from Rasa supports text and image responses, and converts Rasa custom actions into CONGA empty actions.

6 EVALUATION

We have used ASYMOB to perform an empirical study to assess the suitability of our metrics to detect quality issues and compare bots. We aim at answering the following research questions (RQs):

RQ1 Can the defined metrics detect quality issues in real chatbots?
RQ2 Can the defined metrics be used to compare heterogeneous chatbots?

Next, Section 6.1 describes the experiment setting, Sections 6.2 and 6.3 answer the RQs, and Section 6.4 discusses threats to validity.

6.1 Experiment setting

We have analysed 6 Dialogflow chatbots and 6 Rasa chatbots built by third parties, available at https://github.com/ASYM0B/evaluation. Table 2 shows the metric results, with some extreme values marked in bold. Chatbots are categorised depending on their implementation platform (**D**ialog**f**low or **R**as**a**) and their source (**G**ithub or **P**redefined natively on the platform). We used ASYMOB to import the chatbots into the CONGA format (cf. Section 5.2) and obtain the metrics.

6.2 RQ1: Detection of quality issues

Some metric values reveal design issues. The CPOP of the FAQ-RASA-NLU chatbot is 285 characters. This indicates poor accessibility and

Table 2: Summary of the evaluation. Columns use abbreviations for Dialogflow (DF), Rasa (RS), Github (G) and Predefined (P).

Chatbot			Global metrics						Intent metrics								Entity metrics			Flow metrics		
Name	Plat.	Src	INT	ENT	FLOW	PATH	CNF	SNT (%)	TPI	WPTP	VPTP	PPTP	WPOP	VPOP	CPOP	READ	LPE	SPL	WL	FACT	FPATH	CL
bikeShop	DF	G	5	1	4	4	3	38 50 12	2.60	3.48	0.81	0.60	14.00	2.60	55.20	12.00	2.00	5.50	6.64	2.00	1.00	2
googleChallenge	DF	G	32	34	32	32	**1442**	19 59 22	6.94	9.62	1.76	0.94	19.81	3.49	105.16	16.00	3.15	4.54	11.37	1.00	1.00	1
mysteryAnimal	DF	G	62	37	62	62	770	0 0 0	6.52	4.34	1.30	2.27	0.00	0.00	0.00	0.00	163.84	3.89	9.21	3.00	1.00	1
Car	DF	P	77	14	61	117	**4188**	0 0 0	9.70	6.80	1.29	2.25	0.00	0.00	0.00	0.00	14.93	3.60	11.41	1.00	1.92	2
Dining-Out	DF	P	9	15	4	14	148	20 61 19	94.67	3.81	0.76	8.33	8.50	2.56	31.44	7.00	**1255.00**	2.39	11.36	1.25	3.50	3
Easter-Eggs	DF	P	6	0	6	6	0	10 51 **39**	7.17	6.23	1.31	0.00	7.46	1.63	37.54	6.00	0.00	0.00	0.00	1.00	1.00	1
05_event_bot	RS	G	17	0	1	20	167	40 60 0	3.82	2.05	0.12	0.00	15.00	2.75	87.00	12.00	0.00	0.00	0.00	1.02	**20.00**	6
FAQ-RASA-NLU	RS	G	8	0	7	7	4	15 34 **51**	3.38	4.67	1.00	0.00	54.56	3.56	**285.56**	46.00	0.00	0.00	0.00	1.00	1.00	1
small-talk-rasa-stack	RS	G	87	0	86	92	**8467**	24 62 14	22.51	3.82	1.13	0.00	6.66	1.81	28.48	5.00	0.00	0.00	0.00	1.00	1.07	**16**
concertbot	RS	P	6	2	1	1	0	25 75 0	0.00	0.00	0.00	0.00	2.25	0.25	12.25	1.00	0.00	0.00	0.00	3.50	1.00	2
formbot	RS	P	8	1	2	9	37	17 83 0	35.63	4.25	0.92	1.25	5.00	1.50	22.50	4.00	0.00	0.00	0.00	1.16	4.50	7
moodbot	RS	P	6	0	2	4	22	20 80 0	10.50	2.10	0.44	0.00	3.00	1.00	11.25	2.00	0.00	0.00	0.00	1.10	2.00	3

readability, as large answers require scrolling in mobile devices, long reading times (46 seconds for this bot), and cannot be fully displayed on social networks like Twitter due to their message length constraints. As an example, Fig. 5 shows a chatbot response deployed on Telegram using a mobile phone, which requires scrolling as the response has more than 30 lines. This is an example of a content-centric chatbot to access a FAQ. However, according to [23], conversation-centric chatbots with short answers and a natural conversation style are usable in more platforms. The googleChallenge chatbot has the same problem to a lesser extent (CPOP is 105, READ is 16).

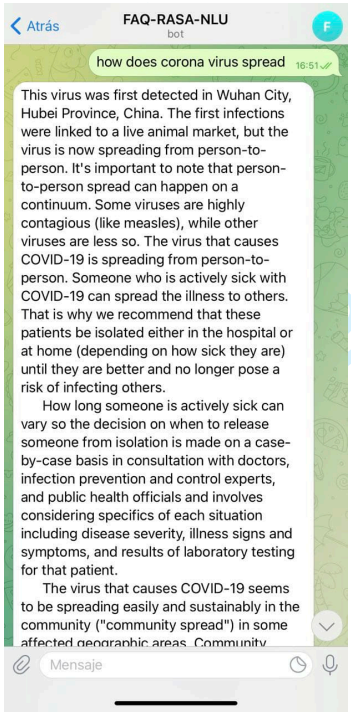


Figure 5: Large response from FAQ-RASA-NLU in a mobile in Telegram.

The flow metrics reveal some complex conversations. The bot 05_event_bot has a single FLOW with 20 paths (PATH and FPATH are 20). Another indicator of conversation complexity is the conversation length CL. The chatbot with the highest CL value (16) is small-talk-rasa-stack.

The sentiment of the bot responses may affect the user experience. In this respect, FAQ-RASA-NLU and Easter-Eggs have 51% and 39% of negative responses (third value of column SNT). Also related to user experience, the high CNF values in bots small-talk-rasa-stack, Car and googleChallenge (8467, 4188 and 1442) may signal chatbot understanding problems due to the existence of similar training phrases in different intents, which may confuse the bots. For example, Car has similar training phrases in different intents, such as *"turn down the heater for each seat in the car"* and *"turn off the heating in my car"*. Other bots with confusing training phrases are small-talk-rasa-stack (*"I am very bored"* / *"I'm bored of you"*), googleChallenge (*"What is the time duration for completing Masters in Artificial Intelligence?"* / *"Completion period for masters in AI?"*), Dining-Out (*"now cafe"* / *"find cafe"*), and bikeShop (*"Can you fix my road bike?"* / *"Can you service my bike?"*). Hence, CNF provides useful information to detect intents that a chatbot may mismatch, without resorting to intensive dynamic testing.

Overall, we can answer RQ1 positively, since our metrics could detect issues regarding readability (CPOP), conversation complexity (FLOW, CL), user experience (SNT) and bot understanding (CNF).

6.3 RQ2: Comparing chatbots

Metrics also serve to compare or classify chatbots based on their design style [23]. For instance, some chatbots like Car, mysteryAnimal and small-talk-rasa-stack are very detailed and complex according to their number of intents (INT), flows (FLOW) and paths (PATH). Instead, others like bikeShop and moodbot are simpler.

Interestingly, two chatbots have no output phrases, one for being a predefined template bot that the developer needs to complete (Car), and the other because a backend API generates the output dynamically (mysteryAnimal). Likewise, chatbots Easter-Eggs, 05_event_bot, small-talk-rasa-stack and FAQ-RASA-NLU lack a domain-specific vocabulary, since ENT is 0. This might be explained as being general-purpose (e.g., for small talk) or simple bots (e.g., 05_event_bot).

Regarding conversations, some chatbots have linear conversations where FLOW=PATH (e.g., FAQ-RASA-NLU, concertbot), while others support complex conversations where FLOW<PATH (e.g., Car, Dining-Out, 05_event_bot). Additionally, the conversation length of some bots is limited to one user-bot interaction (CL=1), and hence, they can be classified as system-centric [23]. Within this set, bots providing long responses (like FAQ-RASA-NLU) are likely content-centric. Other bots allow longer, more elaborate conversations (CL>1). Bots with non-linear conversations (FLOW<PATH) and multiple turns (CL>1) can be classified as conversation-centric [23].

Metrics are also helpful to compare implementation platforms. First, all entity metrics of the analysed Rasa chatbots have value 0. This is so as entities in Rasa are not defined explicitly, but via a Python method that returns whether an entity accepts a given String. The concertbot bot has 0 training phrases because Rasa bots

can be trained interactively. We observe that bots in Rasa define fewer entities (ENT) than in Dialogflow. In general, the analysed Dialogflow bots are more detailed in terms of functionality (INT), vocabulary (ENT) and intent recognition (TPI). Conversations in the Dialogflow bots tend to be linear (PATH=FLOW) while in Rasa they split in several paths (PATH>FLOW), denoting less predictability.

Finally, metrics can be used to compare open-source and predefined bots. In Rasa, the predefined bots are simpler than the Github ones, reflected on lower values of INT, FLOW and PATH. This does not happen with the Dialogflow bots.

Overall, we can answer RQ2 affirmatively. Our metrics permit comparing chatbot complexity and size regarding intents, flows and paths; enable classification of chatbots along Moore and Arar's taxonomy [23]; and – being defined over Conga – they can be applied to different chatbot technologies and chatbot sources.

6.4 Threats to validity

Given the limited size of the experiment, we cannot claim differences or similarities between implementation platforms or predefined/open-source bots, for which we would need a larger scale experiment. Instead, our goal was to hint at the usefulness of the defined static chatbot metrics.

Another limitation of our evaluation is that it relies on custom-made importers from existing platforms into Conga. Since Rasa is a framework, it permits programming some aspects of chatbots in different ways. For example, one may train the model on the fly instead of using training phrases, or even change the conversation flow using Python. All these variants may affect the metric values.

7 CONCLUSIONS AND FUTURE WORK

The increasing relevance of chatbots demands support for assessing their quality prior to testing. With this aim, we have proposed a suite of metrics that can be evaluated statically on chatbot designs, independently of their implementation platform. We have demonstrated the feasibility of our proposal by building the Asymob tool, which we have used to evaluate existing heterogeneous chatbots.

In the future, we plan to extend our evaluation to get a panorama of the features of open-source chatbots and derive metric thresholds. Our metrics could be correlated with development metrics like effort, and validated with usability metrics collected dynamically. We plan to extend our tool to cluster chatbots by similarity, and enable semantic clustering by representing chatbots using a bag-of-words model. The latter can be useful to provide a search facility over chatbot repositories. Technically, we aim at embedding Asymob as a web service to let the community profit from its metrics.

ACKNOWLEDGMENTS

Work funded by the Spanish Ministry of Science (RTI2018-095255-B-I00) and the R&D programme of Madrid (P2018/TCS-4314).

REFERENCES

[1] ISO 9241-11. 1998. Ergonomic requirements for office work with visual display terminals (VDTs). Part II guidance on usability.
[2] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. Gordon Murray, B. Steiner, P. A. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng. 2016. Tensorflow: A system for large-scale machine learning. In *OSDI*. USENIX Association, 265–283.
[3] Botium. [n. d.]. `https://www.botium.ai/`. last access in 2021.
[4] J. Bozic and F. Wotawa. 2019. Testing chatbots using metamorphic relations. In *ICTSS (LNCS, Vol. 11812)*. Springer, 41–55.
[5] S. Bravo-Santos, E. Guerra, and J. de Lara. 2020. Testing chatbots with Charm. In *QUATIC (CCIS, Vol. 1266)*. Springer, 426–438.
[6] M. Brysbaert. 2019. How many words do we read per minute? A review and meta-analysis of reading rate. *J. of Memory and Language* 109 (2019), 104047.
[7] J. Cabot, L. Burgueño, R. Clarisó, G. Daniel, J. Perianez-Pascual, and R. Rodríguez-Echeverría. 2021. Testing challenges for NLP-intensive bots. In *BotSE*. IEEE, 31–34.
[8] D. Cer, Y. Yang, S.-Y. Kong, N. Hua, N. Limtiaco, R. St. John, N. Constant, M. Guajardo-Céspedes, S. Yuan, C. Tar, Y.-H. Sung, B. Strope, and R. Kurzweil. 2018. Universal sentence encoder. *arXiv preprint arXiv:1803.11175* (2018), 7.
[9] Chatbottest. [n. d.]. https://chatbottest.com/. last access in 2021.
[10] D. Coniam. 2014. The linguistic accuracy of chatbots: usability from an ESL perspective. *Text & Talk* 34, 5 (2014), 545–567.
[11] Deep Java Library. [n. d.]. `https://djl.ai/`. last access in 2021.
[12] Dialogflow. [n. d.]. https://dialogflow.com/. last access in 2021.
[13] M. B. dos Santos, A. P. C. C. Furtado, S. C. Nogueira, and D. D. Moreira. 2020. OggyBug: A test automation tool in chatbots. In *SAST*. ACM, 79–87.
[14] N. E. Fenton and S. Lawrence Pfleeger. 1996. *Software metrics - a practical and rigorous approach (2. ed.)*. International Thomson.
[15] FlowXO. [n. d.]. https://flowxo.com/. last access in 2021.
[16] J. Jiang and N. Ahuja. 2020. Response quality in human-chatbot collaborative systems. In *SIGIR*. ACM, 1545–1548.
[17] C. Lebeuf, M.-A. D. Storey, and A. Zagalsky. 2018. Software bots. *IEEE Softw.* 35, 1 (2018), 18–23.
[18] Lex. [n. d.]. https://aws.amazon.com/en/lex/. last access in 2021.
[19] C.-W. Liu, R. Lowe, I. Serban, M. Noseworthy, L. Charlin, and J. Pineau. 2016. How NOT to evaluate your dialogue system: An empirical study of unsupervised evaluation metrics for dialogue response generation. In *EMNLP*. ACL, 2122–2132.
[20] C. D. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. J. Bethard, and D. McClosky. 2014. The Stanford CoreNLP natural language processing toolkit. In *ACL: System Demonstrations*. 55–60.
[21] Microsoft Bot Framework. [n. d.]. https://dev.botframework.com/. last access in 2021.
[22] S. Möller, R. Englert, K.-P. Engelbrecht, V. V. Hafner, A. Jameson, A. Oulasvirta, A. Raake, and N. Reithinger. 2006. Memo: Towards automatic usability evaluation of spoken dialogue services by user error simulations. In *ICSLP*. ISCA.
[23] R. J. Moore and R. Arar. 2018. Conversational UX Design: An Introduction. In *Studies in Conversational UX Design*. Springer, 1–16.
[24] R. J. Moore and R. Arar. 2019. *Conversational UX Design: A Practitioner's Guide to the Natural Conversation Framework*. ACM, New York, NY, USA.
[25] R. J. Moore, E. Young Liu, S. Mishra, and G.-J. Ren. 2020. Design systems for conversational UX. In *CUI*. ACM, 45:1–45:4.
[26] Pandorabots. [n. d.]. https://home.pandorabots.com/. last access in 2021.
[27] D. Peras. 2018. Chatbot evaluation metrics: Review paper. In *ESD*. Varazdin Development and Entrepreneurship Agency, 89–97.
[28] S. Pérez-Soler, E. Guerra, and J. de Lara. 2020. Model-driven chatbot development. In *ER (LNCS, Vol. 12400)*. Springer, 207–222.
[29] S. Pérez-Soler, S. Juárez-Puerta, E. Guerra, and J. de Lara. 2021. Choosing a chatbot development tool. *IEEE Software* 38, 4 (2021), 94–103.
[30] E. Pitler and A. Nenkova. 2008. Revisiting readability: A unified framework for predicting text quality. In *EMNLP*. ACL, 186–195.
[31] N. M. Radziwill and M. C. Benton. 2017. Evaluating quality of chatbots and intelligent conversational agents. (2017). http://arxiv.org/abs/1704.04579
[32] Rasa. [n. d.]. https://rasa.com/. last access in 2021.
[33] R. Ren, J. W. Castro, S. T. Acuña, and J. de Lara. 2019. Evaluation techniques for chatbot usability: A systematic mapping study. *Int. J. Softw. Eng. Knowl. Eng.* 29, 11&12 (2019), 1673–1702.
[34] E. A. Schegloff. 2007. *Sequence Organization in Interaction*. Cambridge University Press.
[35] J. Sedoc, D. Ippolito, A. Kirubarajan, J. Thirani, L. Ungar, and C. Callison-Burch. 2019. Chateval: A tool for chatbot evaluation. In *NAACL-HLT (Demonstrations)*. ACL, 60–65.
[36] A. Shevat. 2017. *Designing bots: Creating conversational experiences*. O'Reilly.
[37] M. A. Walker, D. J. Litman, C. A. Kamm, and A. Abella. 1997. PARADISE: A framework for evaluating spoken dialogue agents. In *ACL/EACL*. Morgan Kaufmann Publishers / ACL, 271–280.
[38] Watson. [n. d.]. https://www.ibm.com/cloud/watson-assistant/. last access in 2021.
[39] J. Weizenbaum. 1966. ELIZA - A computer program for the study of natural language communication between man and machine. *Commun. ACM* 9, 1 (1966), 36–45.

Asymob: a platform for measuring and clustering chatbots

Jose María López-Morales
Universidad Autónoma de Madrid
Madrid, Spain
JoseMaria.LopezM@uam.es

Pablo C. Cañizares
Universidad Autónoma de Madrid
Madrid, Spain
Pablo.Cerro@uam.es

Sara Pérez-Soler
Universidad Autónoma de Madrid
Madrid, Spain
Sara.PerezS@uam.es

Esther Guerra
Universidad Autónoma de Madrid
Madrid, Spain
Esther.Guerra@uam.es

Juan de Lara
Universidad Autónoma de Madrid
Madrid, Spain
Juan.deLara@uam.es

ABSTRACT

Chatbots have become a popular way to access all sorts of services via natural language. Many platforms and tools have been proposed for their construction, like Google's Dialogflow, Amazon's Lex or Rasa. However, most of them still miss integrated quality assurance methods like metrics. Moreover, there is currently a lack of mechanisms to compare and classify chatbots possibly developed with heterogeneous technologies.

To tackle these issues, we present Asymob, a web platform that enables the measurement of chatbots using a suite of 20 metrics. The tool features a repository supporting chatbots built with different technologies, like Dialogflow and Rasa. Asymob's metrics help in detecting quality issues and serve to compare chatbots across and within technologies. The tool also helps in classifying chatbots along conversation topics or design features by means of two clustering methods: based on the chatbot metrics or on the phrases expected and produced by the chatbot. A video showcasing the tool is available at https://www.youtube.com/watch?v=8lpETkILpv8.

CCS CONCEPTS

• **Human-centered computing** → **Natural language interfaces**; • **General and reference** → **Metrics**; • **Social and professional topics** → **Quality assurance**;

KEYWORDS

Chatbot design, metrics, quality assurance

ACM Reference Format:
Jose María López-Morales, Pablo C. Cañizares, Sara Pérez-Soler, Esther Guerra, and Juan de Lara. 2022. Asymob: a platform for measuring and clustering chatbots. In *Proceedings of ICSE (ICSE'22)*. ACM, New York, NY, USA, Article 4, 5 pages. https://doi.org/xx.xxx/xxx_x

ICSE'22, May 21 - 29, 2022, Pittsburgh, USA

ACM ISBN 978-1-4503-8713-2/22/04…$15.00
https://doi.org/xx.xxx/xxx_x

1 INTRODUCTION

Chatbots are increasingly used to access all sorts of services, including leisure (e.g., shopping, booking flights or hotels), customer services, professional support (e.g., banking) and information services (e.g., weather) [18]. Their success is due to their natural language conversational interface, which can be used through many channels such as social networks, web apps, or intelligent speakers.

The popularity of chatbots has triggered the emergence of a plethora of platforms, libraries and tools for their construction [13]. Some prominent examples are Google's Dialogflow[1], Amazon's Lex[2], IBM's Watson[3], Rasa[4] or Pandorabots[5], to name a few.

The quality of chatbots is critical for their success. In this respect, some researchers have proposed techniques for testing chatbots [3, 5] and guidelines for their design [10]. However, most tools lack static quality assurance mechanisms that can be used at design time to assess desired chatbot properties. Likewise, there is a lack of tools to compare, cluster and classify chatbots along design features or conversation topics. Such tools would enable a better understanding of the current chatbot landscape, the comparison of chatbots across implementation technologies (e.g., Dialogflow, Rasa) and provenance (e.g., open source repositories, proprietary platforms), and the extraction of valuable data for chatbot analysis.

In order to address these challenges, we present the web platform Asymob for chatbot measurement and clustering. The tool features a repository where chatbots developed using different technologies (currently Dialogflow and Rasa) can be uploaded. It offers a suite of 20 metrics that measure aspects of design size, complexity, and user experience. It also enables the clustering and comparison of chatbots based on these metrics; as well as on conversation topics extracted from the bot expected and issued phrases. The envisioned users of our tool are chatbot designers and developers.

In the rest of the paper, Section 2 introduces the basic notions of chatbots, Section 3 presents the Asymob platform, reporting on a preliminary evaluation, Section 4 compares with related work, and Section 5 concludes with a summary and open research lines.

2 AN OVERVIEW OF CHATBOTS

Chatbots offer a conversational interface via natural language to software services. They are typically powered by natural language

[1] https://dialogflow.com/
[2] https://aws.amazon.com/en/lex/
[3] https://www.ibm.com/cloud/watson-assistant/
[4] https://rasa.com/
[5] https://home.pandorabots.com/

processing (NLP) technologies that provide good understanding capabilities on sets of predefined topics, called *intents*. Intents are expected conversation topics, which reflect the functionality of the chatbot. Frequently, intents are defined via training phrases that illustrate the different ways a user may approach the chatbot. For example, a chatbot for a pizzeria may have two main intents, one for ordering (expecting phrases like *"A small margherita, please"*) and another for getting information about the available pizza types (expecting utterances like *"What pizzas are available?"*).

Intents may define *parameters*, whose value is extracted from the user utterances. For example, when ordering a pizza, the user should specify the type of pizza (e.g., hawaiian) and the size (e.g., medium), via phrases like *"I'd like a medium hawaiian pizza"*. Parameters may be tagged as mandatory, in which case, the chatbot will request their value if absent from the user phrase. Parameters are typed by *entities*, which can be either user-defined (e.g., for pizza types) or pre-defined (e.g., for numbers or dates).

Conversations are defined by *flows* of expected intents and resulting bot *actions*. The latter normally involve an output phrase, but may also include other elements like images or widgets specific to the deployment channel (e.g., buttons in the Telegram social network). In addition, the chatbot may need to access an external service to manage the intent. For example, in the pizzeria, the chatbot needs to access an information system to store the order.

3 THE ASYMOB PLATFORM

Asymob is a web platform providing static chatbot quality assurance. Next, Section 3.1 describes its architecture, Sections 3.2–3.4 detail its functionality, and Section 3.5 reports on a preliminary evaluation.

3.1 Overview and architecture

Asymob[6] permits *uploading* chatbots of heterogeneous technologies, which then are *measured* using a suite of 20 metrics. Asymob provides *statistics* of the metrics across all chatbots in the repository. In addition, users can *query* the repository to search for chatbots within certain metric bounds and *compare* them against each other according to their metric values. The platform also allows *clustering* chatbots by metric values, or by the conversation topics as given by the words used in training phrases, bot responses and entities.

Fig. 1 shows the architecture of Asymob. Its functionality is offered via a web interface, which interacts with a service layer via a REST API. The *presentation layer* is implemented in HTML and JavaScript, and supports the interactive presentation of metrics and clusters using the libraries Plotly[7] and Cytoscape[8].

The *service layer* (the Asymob core) implements the functionality related to measuring and clustering chatbots. This core has an extensible design, which makes it easy to add new types of metrics, clustering criteria and chatbot technologies. To support the uniform handling of chatbots from heterogeneous technologies, the core relies on a neutral chatbot design notation called Conga [11]. This way, our platform enables the contribution of importers from specific chatbot implementation platforms into Conga, and the measurement and clustering are applied to Conga

[6]http://miso.ii.uam.es/asymobService
[7]https://plotly.com/
[8]https://cytoscape.org/

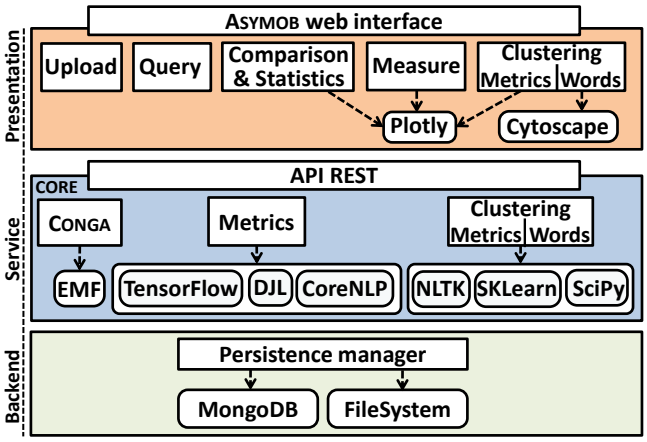


Figure 1: Architecture of Asymob.

models. Section 3.2 will provide more details about Conga and the available importers. Then, Section 3.3 will present the current list of supported metrics, built upon established technologies such as TensorFlow[9], CoreNLP[10] and the Deep Java Library (DJL)[11]. Next, Section 3.4 will focus on the chatbot clustering functionality, developed using Python libraries like NLTK[12], SKLearn[13] and SciPy[14].

An additional *backend layer* provides persistence. This stores the uploaded chatbots in the filesystem of the machine where the Asymob core is deployed, and uses mongoDB[15] for storing the data produced in the service layer (i.e., the metric values and the information required for conducting clustering).

3.2 Handling heterogeneous chatbots

Asymob provides static mechanisms to assess chatbot quality. Since there are many chatbot development tools, Asymob implements those mechanisms over a neutral, technology-agnostic chatbot design notation called Conga, and provides importers from different technologies into Conga. This permits reusing the functionality of Asymob with chatbots of heterogeneous technologies.

Conga [11] is designed based on an analysis of 15 popular chatbot tools, so its primitives can be mapped from/to all of them. Conga supports the concepts explained in Section 2 (intents, parameters, entities, conversation flows, bot actions). Intents can be defined via training phrases, and user-defined entities as a list of words with synonyms, a regular expression, or a set of strings and other entities. Possible chatbot actions include sending text, images, HTTP requests to external services, or presenting widgets like buttons.

Currently, there are importers from Rasa and Dialogflow chatbots into Conga. Rasa is a framework to develop chatbots using Python, markdown and YAML. Dialogflow is a lowcode development platform to create chatbots using a graphical web interface, and the chatbots can be exported as JSON files.

[9]https://www.tensorflow.org/
[10]https://stanfordnlp.github.io/CoreNLP/
[11]https://djl.ai/
[12]https://www.nltk.org/
[13]https://scikit-learn.org/
[14]https://scipy.org/
[15]https://www.mongodb.com/

Table 1: Metrics for chatbot designs.

Metric	Description	Type
Global metrics		
INT	# intents	design size
ENT	# user-defined entities	vocabulary size
FLOW	# conversation entry points	conversation diversity
PATH	# different conversation flow paths	conversation complexity
CNF	# confusing phrases	bot understanding
SNT	# positive, neutral, negative output phrases	user experience
Intent metrics		
TPI	# training phrases per intent	topic complexity
WPTP	# words per training phrase	topic complexity
VPTP	# verbs per training phrase	topic complexity
PPTP	# parameters per training phrase	topic complexity
WPOP	# words per output phrase	readability
VPOP	# verbs per output phrase	readability
CPOP	# characters per output phrase	readability
READ	reading time of the output phrases	readability
Entity metrics		
LPE	# literals per entity	vocabulary complexity
SPL	# synonyms per literal	vocabulary complexity
WL	word length	readability
Flow metrics		
FACT	# actions per flow	bot response complexity
FPATH	# conversation flow paths	conversation complexity
CL	conversation length	conversation complexity

Overall, users of Asymob can register on the platform or use a generic user. When uploading a chatbot to the platform, the user must specify the chatbot implementation technology (Dialogflow, Rasa or Conga), its visibility (private, so that only the owner can see the chatbot, or public, if other users can see it), and its version (to enable version control for the chatbot). Then, the proper importer is automatically applied to the chatbot, and both the original chatbot and the resulting Conga model are stored.

3.3 Measuring chatbots

Asymob includes a metrics engine to analyse static characteristics related to the correct design of chatbots. This provides a suite of 20 metrics, which we proposed in [6], covering both *global* design aspects and specific features concerning the design of *intents*, *entities* and conversation *flows*. The metrics are summarized in Table 1 and explained next.

Global metrics capture global properties of the chatbot. Specifically, Asymob measures the number of intents (INT), user-defined entities (ENT), conversation entry points (FLOW), conversation flow paths (PATH), confusing phrases (CNF), and output phrases with positive, neutral or negative sentiment (SNT). Confusing phrases refer to similar training phrases (i.e., with small semantic distance) defined by different intents. They are problematic, since a chatbot may confuse them and end up identifying a wrong intent. Additionally, a chatbot that mostly outputs phrases with negative sentiment, may impact negatively the user experience. Overall, global metrics are useful to assess the chatbot design size (INT), the vocabolary size (ENT), the conversation diversity and complexity (FLOW and PATH), potential problems in bot understanding (CNF) and user experience (SNT).

Intent metrics measure quality and complexity aspects of intents, namely, the number of training phrases per intent (TPI), words/verbs/parameters per training phrase (WPTP/VPTP/PPTP), words/verbs/characters per output phrase (WPOP/VPOP/CPOP) and average reading time of the output phrases (READ). Overall, these metrics quantify the complexity and readability of phrases. Large phrases are difficult to understand, are problematic in social networks with constrained message length, and may require scrolling in mobile devices with small screens. For example, high CPOP and READ values entail long reading times, which is especially problematic for voice-based chatbots, as speaking takes longer than reading [10].

Entity metrics analyse the user-defined entities, which represent domain concepts. They are useful to obtain a measurement of their complexity and readability. Entity metrics include the number of literals per entity (LPE), the synonyms per literal (SPL) and the length of words (WL). These are indicators of the complexity of the concepts and the width of the vocabulary of the chatbot.

Flow metrics are concerned with the complexity of the conversation flows and the sophistication of the bot responses. They comprise the number of actions per flow (FACT), the number of conversation paths (FPATH) and the conversation length (CL).

When a chatbot is uploaded, Asymob computes its metrics and displays their value in a table and also in interactive graphs that compare these values with statistics of the chatbots in the repository. Fig. 2 shows the graph for metric INT. The left bar displays statistics of the chatbot repository, and the bar to the right displays the metric value for the uploaded chatbot. We observe that the new chatbot can be considered large, since it has 25 intents, while the average number of intents of the chatbots is around 10 (with a median of 6). The computed metrics are persisted to speed up the generation of statistics when new chatbots are uploaded, and to facilitate the following functionalities.

Figure 2: Displaying the value of metric INT.

First, Asymob offers statistics of the metrics of all chatbots in the repository (average, minimum, maximum, median and 1st and 3rd quartiles). They are displayed as a table, as a graph, and side-by-side with the metric values of a specific chatbot, as Fig. 2 shows.

Additionally, Asymob permits comparing chatbots based on a set of metrics, as Fig. 3 shows. The x-axis displays the selected

metrics (ENT, INT and FLOW in the figure), and the y-axis shows their value for the selected chatbots (4 bots in this case). We can see that mysteryAnimal stands out in the three metrics, meaning that it has more vocabulary (entities), conversation alternatives (intents) and conversation flows. This comparison can also be performed for several versions of the same chatbot to reason about the evolution between chatbot versions in terms of metrics.

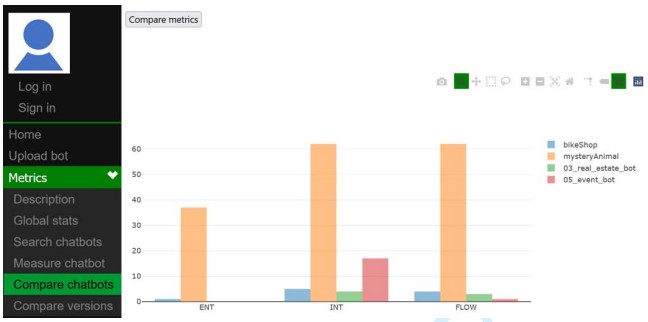


Figure 3: Metric-based comparison of chatbots.

The platform also includes a metric-based chatbot search facility. This permits users to specify the lower and upper limits for the value of some metrics of interest, and Asymob displays the chatbots in the repository with metric values within these boundaries. This is useful to obtain sets of chatbots with certain characteristics, for example simple chatbots with few intents, and no defined entities; or complex chatbots with many intents and complex conversation flows.

3.4 Clustering chatbots

Asymob supports the automated classification of chatbots based on two disjoint criteria: metric values, or the chatbot vocabulary.

Metrics-based clustering is useful to identify groups of chatbots with (dis)similar design features. For this purpose, the user can select one or more metrics, and the chatbots become classified based on the metric values. For example, clustering by metric INT (i.e., number of intents) would create groups of chatbots with similar size complexity, whereas if the user performs the clustering using metrics FLOW and PATH, then the chatbots would be grouped according to the complexity of their conversations.

Technically, the platform implements the K-means algorithm for clustering the chatbots based on the value of the selected metrics. The user can also select the number of clusters to create (i.e., the k-value), or otherwise, it is automatically computed using the silhouette coefficient, as supported by SKLearn.

Asymob visualizes the resulting clusters in a table and graphically, as Fig. 4 shows. The graph can display two or three dimensions, so if the user selects more than three metrics, then the platform reduces the number of dimensions using the principal component analysis (PCA). The graphic represents each chatbot as a dot, and uses a different colour for each cluster of chatbots. In Fig. 4, there are two clusters of 3 and 26 chatbots.

Vocabulary-based clustering classifies chatbots by their vocabulary, which is useful to identify chatbots targeting analogous topics. For example, chatbots for booking flights are likely to be

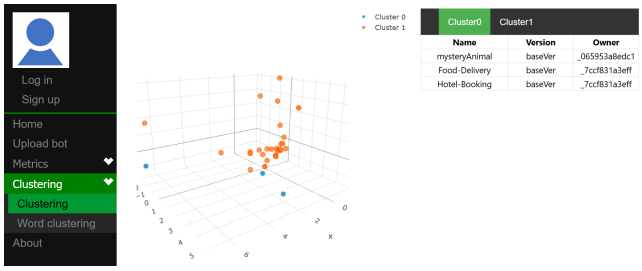


Figure 4: Metrics-based clustering.

in the same cluster, as their vocabulary is similar. We foresee this clustering to be useful as a way to search for chatbots similar to a given one, or by different topics; and as a way to present and organize a large set of chatbots within a repository.

For this kind of clustering, Asymob stores all the relevant words that appear in the training phrases, chatbot responses and user-defined entities of each chatbot, along with their frequency of occurrence. Stop words such as prepositions, articles and conjunctions are discarded. Then, the similarity of two bots is given by the cosine-similarity of their *bag of words* vectors [9]. Note that each chatbot has to be compared with all the other ones, which becomes time-expensive as the repository grows. To reduce this time, Asymob calculates this similarity as a backend process when a chatbot is uploaded, and caches the result in a database.

In the front-end, users can select a set of chatbots and a similarity threshold for the agglomerative clustering algorithm. The results are shown in a table and an interactive hierarchical graph. The first graph layer has a node per cluster, and clicking on a node shows the chatbots it contains. Fig. 5 shows the chatbots within a cluster. The width of the edges conveys the similarity of two chatbots. Clicking on a chatbot displays its metrics on the right.

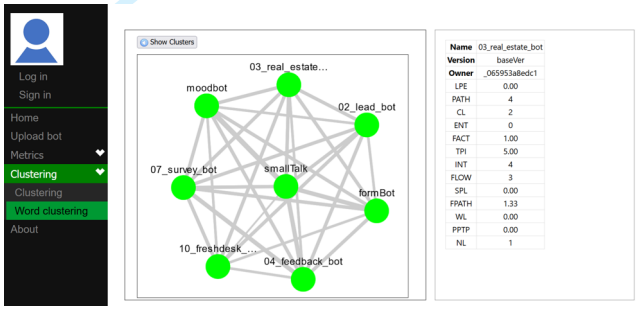


Figure 5: Vocabulary-based clustering.

3.5 Preliminary evaluation

We evaluated the cost of uploading a chatbot, which implies its measurement and extracting its bag-of-words for clustering. The latter requires updating a global vocabulary index when the chatbot introduces new words. We found this to require constant time, in the order of 100ms. At this point, a backend process compares and caches the cosine similarity between the bag-of-words vector of the uploaded chatbot and all the rest, which we found to grow

linear with the number of chatbots (around 99ms per bot). Being a backend process, it does not affect responsiveness, but we are currently working on its parallelization. In the future, we plan to perform more detailed scalability experiments to detect possible bottlenecks in our architecture and optimize where needed.

4 RELATED WORK

Most approaches to assess chatbot quality rely on testing. Some development platforms (e.g., Dialogflow, Lex, Watson) integrate a web chat console to test chatbots manually. There are also dedicated testing tools like Botium [3] and OggyBug [16] which automate the testing of chatbots built with different technologies. Still, developers need to define concrete test conversation cases. To alleviate this burden, Bottester [19] simulates the user interactions, and other works generate challenging test user utterances automatically [4, 5, 15]. Compared to these works, Asymob provides complementary assessment mechanisms to testing in the form of metrics that are collected statically (i.e., without deploying the chatbot) and can reveal defects on several quality aspects of the chatbot design.

Additionally, some development platforms (e.g., Dialogflow, Watson, Bot Framework) provide chatbot analytics. This information is collected dynamically when the chatbot is in production, while Asymob targets the design time.

Another popular way to evaluate chatbots is by means of user studies [8, 17]. These typically evaluate user satisfaction and chatbot performance, and require the recruitment and participation of users [14]. Asymob complements these studies with chatbot information that can be gathered automatically and inexpensively.

The support of static means for quality assessment – like those in Asymob – is less frequent. Next, we discuss some exceptions. Dialogflow performs some chatbot validations (e.g., detecting intents with similar training phrases) categorized by severity. Almansor and Hussain [1] use fuzzy logic to detect inappropriate responses based on the sentiment and length of utterances, and Gao et al. [7] use machine learning to predict the popularity of chatbots based on static metrics (e.g., number of intents, conversation flow length). These two works use metrics supported by Asymob, showing that our tool could enable the use of artificial intelligence for prediction.

Finally, our tool takes inspiration from services available in repositories of other artefacts, like meta-models (e.g., MDEForge [2]). However, to the best of our knowledge, Asymob is the first proposal of a chatbot repository featuring metrics and clustering.

5 CONCLUSIONS AND FUTURE WORK

This paper has presented Asymob, the first platform enabling measuring and clustering chatbots. The tool fills a gap on current practice, which is providing automatic means for assessing chatbot quality prior to their deployment and dynamic testing. The tool comprises a repository of chatbots, static metrics that can be homogeneously evaluated on heterogeneous technologies, and chatbot clustering facilities based on chatbot metrics and vocabulary.

We are currently building converters from other technologies (e.g., Pandorabots, Lex) into Conga. We are also improving the tool with visualization mechanisms able to capture a large amount of informations, e.g., heatmaps and dendograms for clusters. In the future, we plan to use Asymob to evaluate open source chatbots to get a panorama of their features and derive metric thresholds. We also plan to exploit our clustering techniques to provide search facilities over chatbot repositories. Finally, we would also like to integrate Asymob's services within chatbot development tools like the Conga web IDE [12].

ACKNOWLEDGMENT

This work has been funded by the Spanish Ministry of Science (RTI2018-095255-B-I00, "MASSIVE") and the R&D programme of Madrid (P2018/TCS-4314, "FORTE").

REFERENCES

[1] E. Almansor and F. Hussain. 2021. Fuzzy prediction model to measure chatbot quality of service. In *FUZZ-IEEE*. IEEE, 1–4.
[2] F. Basciani, J. di Rocco, D. di Ruscio, L. Iovino, and A. Pierantonio. 2016. Automated clustering of metamodel repositories. In *CAiSE (LNCS)*, Vol. 9694. 342–358.
[3] Botium. [n. d.]. `https://www.botium.ai/`. ([n. d.]). last access in 2021.
[4] J. Bozic and F. Wotawa. 2019. Testing chatbots using metamorphic relations. In *ICTSS (LNCS)*, Vol. 11812. Springer, 41–55.
[5] S. Bravo-Santos, E. Guerra, and J. de Lara. 2020. Testing chatbots with Charm. In *QUATIC (CCIS)*, Vol. 1266. Springer, 426–438.
[6] Pablo C. Cañizares, Sara Pérez-Soler, Esther Guerra, and Juan de Lara. 2022. Automating the Measurement of Heterogeneous Chatbot Designs. In *Proc. SAC*. ACM, 1–8.
[7] M. Gao, X. Liu, A. Xu, and R. Akkiraju. 2021. Chatbot or Chat-Blocker: Predicting chatbot popularity before deployment. In *DIS*. ACM, 1458–1469.
[8] J. Jiang and N. Ahuja. 2020. Response quality in human-chatbot collaborative systems. In *SIGIR*. ACM, 1545–1548.
[9] M. McTear, Z. Callejas, and D. Griol. 2016. *The Conversational Interface. Talking to Smart Devices.* Springer.
[10] R. J. Moore and R. Arar. 2019. *Conversational UX Design: A Practitioner's Guide to the Natural Conversation Framework.* ACM, New York, NY, USA.
[11] S. Pérez-Soler, E. Guerra, and J. de Lara. 2020. Model-driven chatbot development. In *ER (LNCS)*, Vol. 12400. Springer, 207–222.
[12] S. Pérez-Soler, E. Guerra, and J. de Lara. 2021. Creating and Migrating Chatbots with Conga. In *ICSE Companion*. IEEE, 37–40.
[13] S. Pérez-Soler, S. Juarez-Puerta, E. Guerra, and J. de Lara. 2021. Choosing a chatbot development tool. *IEEE Softw.* 38, 4 (2021), 94–103.
[14] R. Ren, J. W. Castro, S. Teresita Acuña, and J. de Lara. 2019. Evaluation techniques for chatbot usability: A systematic mapping study. *Int. J. Softw. Eng. Knowl. Eng.* 29, 11&12 (2019), 1673–1702.
[15] E. Ruane and *et al.* 2018. BoTest: A framework to test the quality of conversational agents using divergent input examples. In *IUI Companion*. ACM, 64:1–64:2.
[16] M. B. dos Santos, A. Cavalcanti, S. C. Nogueira, and D. D. Moreira. 2020. OggyBug: A test automation tool in chatbots. In *SAST*. ACM, 79–87.
[17] B. A. Shawar and E. Atwell. 2007. Different measurements metrics to evaluate a chatbot system. In *NAACL-HLT-Dialog*. ACM, 89–96.
[18] Amir Shevat. 2017. *Designing bots: Creating conversational experiences.* O'Reilly.
[19] M. Vasconcelos, H. Candello, C. S. Pinhanez, and T. dos Santos. 2017. Bottester: Testing conversational systems with simulated users. In *IHC*. ACM, 73:1–73:4.