Information and Software Technology 147 (2022) 106886

Contents lists available at ScienceDirect

Information and Software Technology

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

SE^{3}M: A model for software effort estimation using pre-trained embedding models

Eliane Maria De Bortoli Fávero [a,*], Dalcimar Casanova [a], Andrey Ricardo Pimentel [b]

[a] Via do Conhecimento, Km 01, S/N - Pato Branco Paraná, 85503-390, Brazil
[b] Rua Evaristo F. F. da Costa, 418 - Jardim das Americas, Curitiba, Paraná, Brazil

ARTICLE INFO

Keywords:
Software effort estimation
Pre-trained model
Context-less embedding
Contextualized embedding
Domain-specific model
BERT

ABSTRACT

Context: Software effort estimation from requirements texts, presents many challenges, mainly in getting viable features to infer effort. The most recent Natural Language Processing (NLP) initiatives for this purpose apply context-less embedding models, which are often not sufficient to adequately discriminate each analyzed sentence. Contextualized pre-trained embedding models have emerged quite recently and have been shown to be far more effective than context-less models in representing textual features.
Objective: This paper proposes evaluating the effectiveness of pre-trained embedding models, to explore a more effective technique for representing textual requirements, which are used to infer effort estimates by analogy.
Method: Generic pre-trained models went through a fine-tuning process for both approaches — context-less and contextualized. The generated models were used as input in the applied deep learning architecture, with linear output. The results were very promising, realizing that contextualized pre-trained embedding models can be used to estimate software effort based only on requirements texts.
Results: We highlight the results obtained to apply the contextualized pre-trained model BERT with fine-tuning, applied in a single repository containing different projects, whose Mean Absolute Error (MAE) value is 4.25 and the standard deviation is only 0.17. This represents a result very positive when compared to similar works.
Conclusion: The main advantages of the proposed estimation method are reliability, the possibility of generalization, speed, and low computational cost. Such advantages are provided by the fine-tuning process, enabling to infer effort estimation for new or existing requirements.

1. Introduction

Estimating software effort is a challenging and important activity in the software development process. This activity depends on the success of other crucial aspects of a project, mainly related to the achievement of time and budget constraints, directly impacting the software product's quality. The success of a particular software project depends heavily on how accurate its effort estimates are [1]. The importance of the estimates accuracy has been explored by studies in the field of software engineering (SE) published in recent years, such as: [2–4] and, [5] These studies seek to explore computational techniques both individually or in combination, seeking to achieve better levels of accuracy for effort estimation techniques.

Among the existing classifications for software effort estimation techniques, the Analogy-based Software Effort Estimation (ABSEE) can fit either agile or traditional models as long as the estimation approach is based on data and previous team experiences to estimate software projects. One challenge that has been presented for using these techniques, especially in agile models, is the lack of project data and their requirements in the early stages of the development process. The basic specification of software requirements used in these models is the user story, which is user needs, usually written informally [6].

Among the main limitations when using textual requirements is the occurrence of a variety of domain-specific information, such as natural language text containing source code, numerical data (e.g. IP addresses, artifact identification codes), among others. A very common aspect is the occurrence of different words, but used in the same context; in this case they should be considered similar (polysemy) because their context is similar. Or, equal words (ambiguous), but applied in different contexts, therefore, should not be represented in the same way.

* Corresponding author.
E-mail addresses: elianedb@utfpr.edu.br (E.M.B. Fávero), dalcimar@utfpr.edu.br (D. Casanova), andrey@inf.ufpr.br (A.R. Pimentel).

https://doi.org/10.1016/j.infsof.2022.106886
Received 19 July 2021; Received in revised form 8 December 2021; Accepted 16 February 2022
Available online 26 February 2022

On the other hand, in the AI world, especially in the Natural Language Process (NLP) field, word embedding methods mainly aim to capture the semantics of a given word in a specific context. This method allows words to be represented densely and with low dimensionality, facilitating machine learning tasks that use textual characteristics. Breakthroughs in training word embedding models for a variety of purposes began in years recent with the emergence of Word2Vec [7] and GloVe [8], enabling models to learn from a very large corpus.

More specifically applied in the generation of software effort estimates, the use of embedding is highlighted in two studies [5,9]. In the first case, Ionescu (2017) explores the use of word embedding generated by a context-less approach (Word2Vec), aggregated with design attributes and textual metrics (e.g. term frequency–inverse document frequency (TF–IDF), from which positive results were obtained. Choetkiertikul et al. (2018) seek to infer estimates from the text of user stories, which were given as input to a deep learning architecture, with an embedding layer as input. However, these initiatives face two main limitations, which make it difficult to solve in the specific field of SE. Are they:

1. **Domain-specific terms present their meanings changed according to the context in which they are used:** some studies were carried out, seeking to develop resources to facilitate textual representation in software engineering a (i); s (f), but no complete solution for the presentation of contexts. Still, regarding the context representation, the textual requirements are usually short, bringing an inherent difficulty in identifying the context to which they refer.
2. **Lack of domain-specific SE data to train smart models:** this aspect makes it very difficult to train deep neural networks, which tend to overfitting themselves in this small training corpus, not reaching generalization. This reality is no different for textual software requirements and becomes more critical when we need these texts to be accompanied by their labels, which should represent the effort required to implement them.

To solve both problems, this work explores the use of contextualized pre-training embedding models (e.g. BERT [10]) to infer an estimate of software effort from textual requirements. Pre-trained models present the concept of transfer learning [11], which allows us to solve these problems because we can train or model on a generic corpus (solve a lack of data problems) and adjust it (solve o problem of the specific meaning of words in different contexts).

As shown, so far no research has explored the application of contextualized pre-trained embedding models following the task of ABSEE. Thus, this approach, entitled Software Effort Estimation Embedding Model (SE^3M) **aims to present a model for the inference of effort estimates by analogy, both for existing and new software requirements, using contextualized pre-trained embedding models, having as input the exclusive use of textual requirements (e.g. user stories), generated in the initial stage of development**.

Pre-trained embedding models make it easy to perform NLP tasks related to SE, without the need for training from scratch, and with a low computational cost. According to Howard e Ruder (2018), this is possible through the fine-tuning technique, eliminating the need for a representative corpus. The fine-tuning approach consists of changing minimal task-specific parameters and is trained on the downstream tasks by simply fine-tuning all pre-trained [10]. Most language representation models (e.g. Word2Vec, ELMo, Glove) are classified as context-less, i.e. each token considers only words on the left or right as part of its context [12]. This makes fine-tuning approaches difficult, where it is relevant to incorporate the context bidirectionally, the reason for using BERT models. BERT is a contextual representation model that solves the one-way constraint mentioned earlier.

Therefore, the approach also aims to infer software effort estimation using pre-trained embedding models, with and without fine-tuning on a SE specific corpus. Thus, with the results of this article, we intend to answer the following research questions (RQ):

- RQ1. Does a generically pre-trained word embedding model show similar results with a software engineering pre-trained model?
- RQ2. Would embedding models generated by context-less methods (i.e. Word2Vec) be effective as models generated by contextualized methods (i.e. BERT)?
- RQ3. Are pre-trained embedding models useful to a text-based software effort estimation?
- RQ4. Are pre-trained embedding models useful to a text-based software effort estimation, both on new and existing projects?
- RQ5. Are the results found generalizable, aiming to generate estimates of effort between projects or companies?

It is worth mentioning that, unlike the more similar approaches [5, 9], the proposed approach proposes to be generalizable, that is, it should allow the generation of estimates, independent of projects and/or companies, both for new requirements, as for existing ones.

The structure of this article to organize as follows. Session 2 presents the background of software effort estimation and word embedding, in sequence (session 3) to present the related works. Session 4 presents the theoretical and practical aspects necessary for the construction of the proposed approach, as well as data collection and pre-processing procedures. Then, in session 5 the initial research questions will be answered and discussed. Section 6 presents the threats to validity, followed by the conclusion and future work.

2. Background

In this section, we first introduce aspects related to software effort estimation (Section 2.1). The following are the concepts of word embedding and the context-less and contextualized paradigms (Sections 2.2 and 2.3).

2.1. Software effort estimation

Various classifications for software effort estimation models have been applied in the last decades, with small differences, according to each author's point of view. According to Shivhare [13], software effort estimation models can subdivide into algorithmic/parametric and non-algorithmic. The first ones are those that use algorithmic models, applied to project attributes and/or requirements to calculate their estimate, presenting themselves as reproducible methods in substitution to non-algorithmic human expert methods [14]. Examples of algorithmic models are COCOMO II [15] and Function Point Analysis [16]. Non-algorithmic ones are those based on Machine Learning techniques (e.g. linear regression, neural networks) and are also called models by analogy, which rely on historical data to generate custom models that can learn from this data [17].

Chiu and Huang [18] point out that the ABSEE estimate deals with the process of identifying one or more historical projects similar to the target project, and from them infer the estimate. In other words, but using the same line of reasoning, Shepperd's [19], says that the basis for the ABSEE technique is to describe (in terms of several variables) the project that must be estimated, and then, to use this description to find other similar projects that have been completed. In this way, the known effort values for these completed projects can be used to create an estimate for the new project.

Therefore, the ABSEE is classified as non-algorithmic. Idri and Abran [1] also classify a technique by analogy as a machine learning technique. These authors further point out that machine learning models have also gained significant attention for effort estimation purposes, as they can model the complex relationship between effort and software attributes (cost factors), especially when this relationship is not linear, and it does not appear to have any predetermined form. Analog-based reasoning approaches have proven to be promising in the field of software effort estimation, and their use has increased among software researchers [20].

Idri and Abran [1] conducted a systematic review of the literature on ABSEE and found that these techniques outperform other prediction techniques. This conclusion to support by most of the works selected in their mapping. The similarity with human reasoning by analogy is among the main advantages of ABSEE.

Thus, the estimation of software effort by analogy is perceived as a very appropriate technique when the input resources are requirements specifications in the unstructured text format, as is the case of this research. Since it is possible to submit textual characteristics, drawn from these specifications, as input to machine learning models (e.g. neural networks).

2.2. Word embedding

Unlike lexical dictionaries (e.g. WordNet), which basically consist of a thesaurus, grouping words based on their meanings [21] and which are usually built with human support, a word embedding model is made up of word representation vectors. Through these vectors, it is possible to identify a semantic relationship between words in a given domain, based on the properties observed in a training body and their automatically created generation [7].

Word embedding is currently being a strong trend in NLP. Word embedding models use neural network architecture to represent each word as a dense vector with low dimensionality and focusing on the relationship between words [7]. Such vectors are used independently to calculate similarities between terms and as a basis of representation for NLP tasks (e.g. text classification, entity recognition, sentiment analysis). The word embedding models emerged to solve some limitations imposed by the bag-of-words (BOW) model, which usually presents sparse and high dimensionality matrices.

Word2Vec, one of the most popular methods for generating word embeddings from a text corpus, is an unsupervised learning algorithm that automatically attempts to learn the relationship between words by grouping words that have similar meanings into similar clusters [22]. In the Word2Vec model, [7], a neural network is trained to represent each word in the vocabulary as an n-dimensional vector. The general idea is that the distance between the vectors representing the word embedding is smaller if the corresponding words are semantically more similar (or related). Pennington et al. [8] add that for the generation of these vectors, the method captures the distributional semantics and co-occurrence statistics for each word in the presented training corpus.

The Word2Vec model internally implements two neural network-based approaches: Common Bag of Words (CBOW) and skip-gram. Both are models for word embedding widely used in information retrieval tasks. The goal of the CBOW model is to predict the current word based on its context, while the skip-gram model is intended to predict the current surrounding words for the given word. For both cases, the model consists only of a single weight matrix (in addition to the word analyzed), which results in training capable of capturing semantic information [7]. After training, each word must be mapped to a low-dimensional vector. Results presented by their authors [7] show that words with similar meanings are much closer to those with different meanings. Generally speaking, the key concept of Word2Vec is to find words that share common contexts in the training corpus, close to the vector space compared to others.

Word2Vec, like other models (e.g. Glove, FastText) is considered a context-less (static) method for generating pre-trained textual representations. This feature means that these models have constraints regarding the representation of the context of words in a text, making sentence-level or even fine-tuning token tasks difficult. Also, according to their authors [7], these models are considered too shallow, as they represent each word by only one layer, and there is a limit to the amount of information they can capture. And finally, these models do not consider word polysemy, that is, the same word to use in different contexts can have different meanings, which is not dealt with by these models.

The Fig. 1 presents a non-exhaustive differentiation between contextualized and context-less models. As we can see, Word2Vec is a form of static word embedding such as Glove [8], Fast Text [23], among others.

2.3. BERT

The BERT is an innovative method, considered the state of the art in pre-trained language representation [10]. BERT models are considered contextualized or dynamic models, and have shown much-improved results in several NLP tasks [11,25–27] as sentiment classification, calculation of semantic tasks of textual similarity and recognition of tasks of textual linking.

This model originated from various ideas and initiatives aimed at textual representation that have emerged in the area of NLP in recent years, such as: coVe [28], ELMo [26], ULMFiT [11], CVT [29], context2Vec [30], the OpenAI transformer (GPT and GPT-2) [27] and the Transformer [12].

BERT is characterized as a dynamic method, mainly because it has an attention mechanism, also called Transformer [10], which allows analyzing the context of each word in a text individually, including checking if each word has been previously used in a text with the same context. This allows the method to learn contextual relationships between words (or subwords) in a text.

BERT consists of several Transformer models [12] whose parameters are pre-trained on an unlabeled corpus like Wikipedia and BooksCorpus [31]. It can say that for a given input sentence, BERT “looks left and right several times” and outputs a dense vector representation for each word. For this reason, BERT is classified as a profoundly two-way model because it learns two representations of each word, one on the right and one on the left, and this learning to repeat n times. These representations are concatenated to obtain a final representation to use in future tasks.

The pre-processing model adopted by BERT accomplishes two main tasks: masked language modeling (MLM) and next sentence prediction (NSP). In the MLM task, the authors argue [10] that it is possible to predict a particular masked word from the context. For example, let us say we have a phrase: “I love reading data science articles”. We want to train a contextualized language model. In this case, you need to replace “data” with “[MASK]”. It is a token to indicate that it is missing. We will then train the model so that it can predict “date” as the missing token: “I love reading articles from [MASK] science”.

This technique aims to make the model learn the relationship between words, improving the level of learning, avoiding a possible “vicious cycle”, in which the prediction of a word to base on the word itself. Devlin et al. [10] used 15%–20% of words as masked words.

The task of NSP is to learn the relation between sentences. As with MLM, given two sentences (A and B), we want to know if B is the next sentence after A in the corpus or if it would be any sentence.

With this, BERT combines the pre-training tasks of both MLM and NSP, making it a task-independent model. For this, their authors provided pre-trained models in a generic corpus but allowing fine-tuning. It means that instead of taking days to pre-training work, it only takes a few hours. According to the authors of BERT [10], a new state of the art has been achieved in all NLP tasks. The authors have attempted (e.g. Question Answering (QA) and Natural Language Inference (NLI)), among others.

3. Related works

When performing a systematic mapping focusing on effort estimation on analogy-based, some studies are found, as [5,9,32–37], that obtain the effort estimate from texts.

It was observed in most of the studies presented the application bag-of-words approach, considering word-level features (e.g. TF, TF–IDF, part-of-speech (POS) tag), not applying specific knowledge about the text structure, that means, ignoring aspects of context. Only two studies [5,9] differ from these attributes, as they apply word embedding models, but none of them use pre-trained embedding models.

Ionescu [5] proposed a machine learning-based method for estimating effort for software development, using as input the requirement

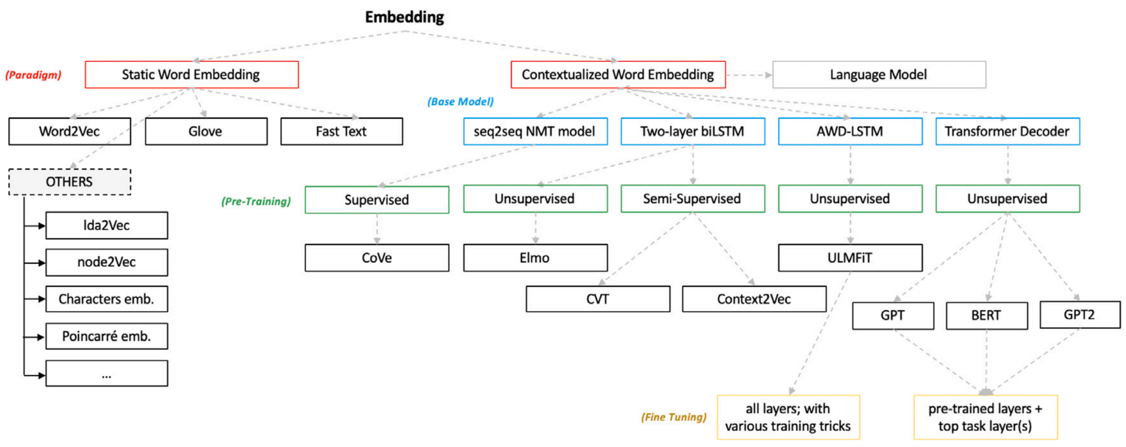

Fig. 1. Differentiation between contextualized and context-less (static) embedding models.
Source: adapted of Haj-Yahia et al. [24]

texts and project management metrics. The authors applied an original statistical pre-processing method to try out better results. First, a custom vocabulary was made. It was done using the standard deviation of the effort of those requirements where each word appears in the training set. For each requirement, a percentage of your words is maintained based on this statistic. The resulting requirements are concatenated with available project metrics. A modified TF–IDF calculation was also used, and numerical data were produced to form a bag-of-words, which is used as input to a linear regression algorithm.

Choetkiertikul et al. [9] proposed the use of the deep learning. Two neural network models were to combine into the proposed deep learning architecture: The Long Short Term Memory (LSTM), which are long term memories and the recurrent highway network. The model is trainable from beginning to end with raw input data that has only gone through a pre-processing step. The model learns from the story point estimated by previous projects to predict the effort of new user stories. This proposal [9] uses context-less word embedding as input to the LSTM layer. As input data, the title and description of the requirements report were combined into a single sentence.

The embedding vectors generated in this first layer serve as input to the LSTM layer, which then generates a representation vector for the full sentence. It should be to note that this process of embedding vectors generation for each sentence, becomes computationally expensive. For this reason, the authors pre-train these layers, and then make these models available for use. This sentence vector is fed into the recurrent highway network, which transforms the sentence vector several times before producing a final vector that represents each sentence. Finally, the sentence vectors undergo the simple linear regression, predicting their effort.

The possible bottleneck of this approach is the difficulty to feeding the model with new data. This would fine-tune models, making them increasingly accurate. This difficulty occurs because with each new insertion into the dataset, the pre-training process, and consequently, its cost needs to be to repeat. Besides, the method by Choetkirtkul et al. [9] realizes inter-project prediction, which is not repository independent.

As a differential, our method seeks to make effort estimation in a generic way, that is, using a single requirements repository, independent of projects or repositories. A method of contextualized word embedding (i.e. BERT) allows a deeply learning in the context of each selected word, solving problems of polysemy and ambiguity. When feeding the model with new training data, the adjustment can be made in a few hours, being cheaper computationally than the generation of a pre-trained model from zero.

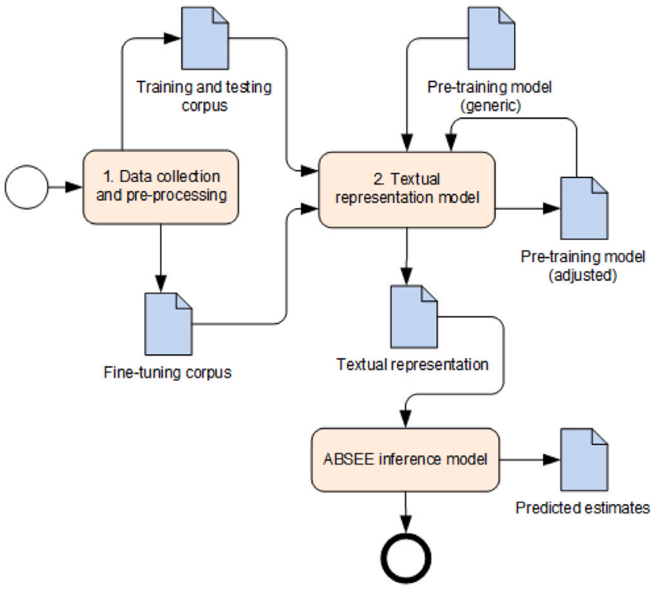
Training and testing corpus
Pre-training model (generic)
1. Data collection and pre-processing
2. Textual representation model
Pre-training model (adjusted)
Fine-tuning corpus
Textual representation
ABSEE inference model
Predicted estimates

Fig. 2. General architecture of the proposed model.

4. Construction approach

The main objective of this research is to evaluate the efficacy of contextualized pre-training embedding models aiming to effort estimation from requirement texts.

A requirement in this context can be a case of use, a user story, or any software requirement, provided that the data is in text format, and aligned with the target effort. For this work, user stories serve as the input to the proposed model and are composed of their description and their effort, provided in story points. The textual format of this data requires some basic pre-processing (e.g. removal of special characters and stopwords) before their use in the models.

For a comparison, pre-trained models generated from two approaches for modeling language will be applied: context-less and contextualized models. Fig. 2 presents the steps that make up the overall architecture of the proposed model.

1. **Data collection and pre-processing:** in this step, the data collection and preparation procedures are performed for later use during the *feature learning* step, which will generate a context vector (i.e. numerical representation) for a given requirement

text. For the proposed model, two corpus of texts containing software requirements will be required: *corp_SE* e *corpPret_SE* (as shown in Table 2). One of them will be the fine-tuning corpus, in which the texts are not labeled. The other corpus will be used during the training and testing stages of the inference model, in which each text will be labeled with its respective efforts. The texts for both corpus go through basic pre-processing procedures (e.g. removal of special characters and *stopwords*). It is important to inform that, for this case, in which representations of sentences are obtained from embeddings, all words are important to obtain the context of a sentence. Therefore, only non-text elements (e.g. numbers, HTML tags) were removed.

2. ***Textual representation model*:** this step consists in applying methods of deep learning to the textual characteristics [38], aiming to generate the vector representation for each of the texts. Therefore, these methods do not use manual activity for the generation of characteristics. To achieve this goal, methods to generate context-less embedding (e.g. Word2Vec) and contextualized embedding (e.g. BERT) are used. Therefore, this step comprises the fine-tuning procedure of the generic pre-trained models for both approaches, which are given as input to the inference model. For fine-tuning, an unlabeled corpus (corpPret_SE - according to Table 2) is applied together with generic pre-trained models for each embedding approach (*word2vec_base* and *BERT_base* - according to Table 3). As an output, two pre-trained adjusted models are generated: *word2vec_SE* and *BERT_SE* (according to Table 6. Then, the pre-trained and adjusted models are used to extract the textual representations for each requirement that makes up the training and testing corpus, using appropriate pooling techniques (e.g. mean, sum) applied to embedding for each word. This textual representation is given by a matrix containing the number of samples from the training and test corpus concerning the number of dimensions of the respective embedding (context-less and contextualized). The vector representation of each requirement is given as an input to the ABSEE inference model, aiming to learn and infer new estimates.
3. **Inference model for ABSEE**: the textual representation for the set of training and testing requirements is submitted as input to the inference model. This model is composed of deep learning architecture, which is considered to be quite simplified when compared to VGG-type models [39], Resnet [40], among others. It is important to note that the concept of deep learning is not related to the number of layers of neural networks that make up the architecture, but rather to the fact that this architecture executes deep learning of the text's characteristics, through the process of learning features, which begins by representing the texts using an embedding model. Therefore, the characteristics learned during this process are applied through the embedding layer of the deep learning architecture used. Since the network entry is a sequence of words (100 words are considered for each text), an LSTM layer has the function of processing that sequence, generating a representation for the sentence (that is, for each text). Subsequently, two dense layers with nonlinear activation functions are used (the dimensions of the hidden layers are 50 and 10 respectively), ending with a linear regression layer. This is an even smaller architecture, when compared to the most similar work found [9], excluding the role of recurring networks [41,42] and applying a single sequence feedforward after the representation layer. This is possible due to the feature learning methods applied in the previous step. A simplified architecture also aims to not mask the results generated from the expected inputs to the network.

After performing an inference of the estimates for each textual requirement for the training and testing sets, metrics for performance evaluation were applied, identified in the learning model used (according to Section 4.1), in order to analyze the feasibility of the application of the model developed. This evaluation process was carried out using applicable statistical and graphic techniques, always related to the real development environment.

4.1. Evaluation metrics

The evaluation metrics used to evaluate the model performance, refer to the distance between the test set values and predicted values. For this, some metrics were selected, which have been recommended for the evaluation of regression-based software effort prediction models [9,22,43]. They are Mean Absolute Error (MAE), Median Absolute Error (MdAE), and Mean Square Error (MSE).

$$MAE = \frac{1}{N}\sum_{i=1}^{N} |actual\_eff - estimated\_eff_{\mathrm{i}}| \tag{1}$$

where N is the number of textual requirements (e.g. user stories) that make up the test set used to evaluate model performance. Actual_eff is the current effort measure, and estimated_eff$_{\mathrm{i}}$ is the estimated effort measure for a given textual requirement i. We also used the Median Absolute Error (MdAE), suggested as a more robust metric for large outliers [9]. MdAE is defined as:

$$MdAE = median|actual\_eff_{\mathrm{i}} - estimated\_eff_{\mathrm{i}}| \tag{2}$$

The Mean Squared Error (MSE) metric, represented by the equation below, was also applied:

$$MSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(actual\_eff_{\mathrm{i}} - estimated\_eff_{\mathrm{i}})^2} \tag{3}$$

Seeking to avoid bias towards underestimation and looking for a stable metric to compare effort estimation models [44], the Standardized Accuracy (SA) metric was also included [43,45].

$$SA = 1 - (MAE/MAE_{\mathrm{rguess}}) \times 100 \tag{4}$$

where MAErguess is the MAE of a large number (e.g. 1000 runs) of random guesses. SA measures the comparison against random guessing. Predictive performance can be improved by decreasing MAE or increasing SA.

4.2. Data collection and pre-processing

In order to obtain and prepare the data that makes up the training and testing corpus, and the fine-tuning corpus, the data collection, and pre-processing steps were carried out using API's for the NLP, based on models previously established by the literature. These steps precede the process of representing textual characteristics, that is, the generation of the context vector for each requirement.

Thus, a specific corpus to the SE context (*corp_SE*) was used to create the training and testing corpus. This is composed of textual requirements, more specifically user stories [9], labeled according to their respective development efforts. It is important to highlight that, despite being referred to as user stories, the textual requirements do not have a standard structure. Regarding the effort attributed to each requirement, it is worth mentioning that no single measurement scale was adopted (ex. *Fibonacci*). The *corp_SE* (Table 2) is considered by the authors to be the first dataset where the focus is on the level of requirements (e.g. user stories) and not just on the project level, as in most datasets available for SE researches.

The textual requirements, as well as the effort given to each of them, were obtained from large open sources project management systems (e.g. Jira), totaling 23.313 requirements (Table 1), which were initially offered by [46]. Subsequently, Choetkiertikul et al. [9] used the same dataset to carry out his research, aiming to estimate software effort by analogy. Despite the difficulty in obtaining the real effort to implement a software requirement, the authors claim to have been able to obtain the implementation time based on the situation (*status*) of the

Table 1
Number of textual requirements (user stories) and description of each of the 16 projects used in the experiments [9].

Project ID	Description	Requirements/project
0	Mesos	1680
1	Usergrid	482
2	Appcelerator Studio	2919
3	Aptana Studio	829
4	Titanium SDK/CLI	2251
5	DuraCloud	666
6	Bamboo	521
7	Clover	384
8	JIRA Software	352
9	Moodle	1166
10	Data Management	4667
11	Mule	889
12	Mule Studio	732
13	Spring XD	3526
14	Talend Data Quality	1381
15	Talend ESB	868
Total		**23.313**

Table 2
Corpus used in the experiments carried out.

Corpus	Specification	Aplication	Labeled
corp_SE	Contains 23.313 user stories and consists of 16 large open source projects in 9 repositories (Apache, Appcelerator, DuraSpace, Atlassian, Moodle, Lsstcorp, Mulesoft, Spring e Talendforge).	Train and test	Yes
corpPret_SE	It consists of more than 290 thousand texts of software requirements of different projects (ex. *Apache, Moodle, Mesos*).	Fine-tuning	No

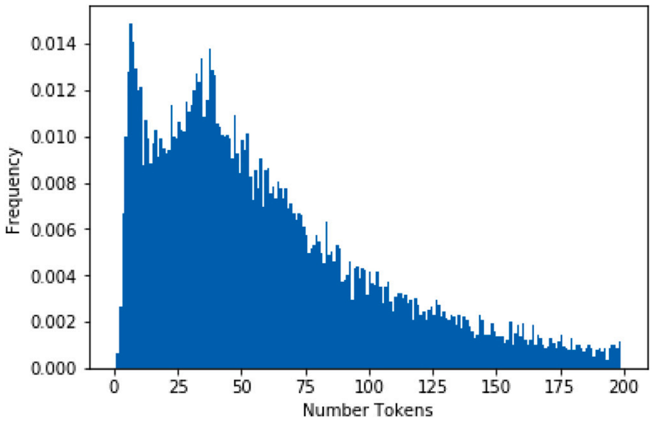

Fig. 3. Histogram representing the number of words in each sentence of the training and testing corpus.

Table 3
Pre-trained models used in the performed experiments.

Pre-trained model before fine-tuning	Specification
word2vec_base	Trained on a *corpus* from *Wikipedia* using the *Word2Vec* [7] algorithm. For this, the following hyperparameters were used: number of dimensions of the hidden layer = 100; method applied to the learning task = *cbow*.
BERT_base	*Bert_base uncased*: 12 layers for each *token*, 768 hidden layers, 12 heads of attention, 110 million parameters. The *uncased* specification means that the text was converted to lower case before *tokenization* based on *WordPiece*, in addition, removes any accent marks. This model was trained with English texts (Wikipedia) with lowercase letters.

requirement. Thus, the effort was obtained beginning from the moment when the situation was defined as "in progress" until the moment when it was changed to "resolved". Thus, Choetkiertikul et al. [9] applied two statistical tests (*Spearman's* and *Pearson* correlation) [47], which suggested a correlation between the story points assigned and their real effort. Therefore, this same dataset (referenced by corpus) was applied to the research proposed by this paper. It is known that these story points were estimated by human teams and, therefore, may contain biases, and in some cases, may not be accurate, which may cause some level of inaccuracy in the models.

In this way, it is important to say that there is variability in the attribution of the effort metric in story points, which can be defined by Planning Poker (PP) or another estimate method. The Choetkiertikul et al. [9] work performed normalization of story points, considering several factors, such as the number of requirement comments, work hours, among others. Therefore, to the SE^3M model, normalized story points have been used.

Typically, user stories are measured on a scale based on series of *Fibonacci* [48], called *Planning Poker* (e.g. 1, 2, 3, 5, 8, 13, 21, 40, 100) [6]. As there is no standardized use of this scale among the projects used to create *corp_SE*, there was no way to approximate the story points provided. Therefore, 100 possible predictions were considered, distributed over the *corp_SE*.

In this way, the effort estimate was treated as a regression problem.

For the fine-tuning process, which makes up the proposed model, a corpus of specific texts from SE, the *corpPret_SE* (shown in the Table 2 is used. It is not labeled, therefore the training carried out is unsupervised, and is composed of texts with specifications of requirements, obtained from open source repositories, according to the procedure described by [9].

While exploring the data available in *corp_SE*, some relevant aspects that interfere in the inference model's settings were observed. The histogram of Fig. 3 allows evaluating the maximum number of words to be considered per text. The average number of words per text, accompanied by its standard deviation, is 53 ± 108.6.

When analyzing the frequency of distribution of the effort size in the training and testing corpus (*corp_SE*), it is noted that most of the samples have smaller efforts (between 1 and 8 story points).

The cross-validation *10-fold* method was applied in order to partition the *corp_SE* to carry out the experiments. For each subset of data, the mean and standard deviation for the metrics applied in the performance evaluation were obtained, as described in Section 4.1.

4.3. Textual representation model

The purpose of the procedures described in this section is to generate models to represent the texts that make up the training and test corpus. These representation models will be obtained from the generic and adjusted pre-trained embedding models, that must consider the diversity of existing contexts. Thus, the representation models (according to Fig. 2) serve as input to the proposed sequential architecture.

Thus, for the context-less approach, a pre-trained model called *word2vec_base* was used, whose specifications are shown in Table 3. As a contextualized model, a generic BERT model (*BERT_base uncased*) had previously been pre-trained and made available by its authors [10] for free use in NLP tasks, as specified in Table 3.

The *BERT_base* model, as well as the other pre-trained BERT models, offers 3 components [10]:

- A *TensorFlow checkpoint (bert_model.ckpt)* that contains pre-trained weights (consisting of 3 files).
- A vocabulary file (vocab.txt) for mapping *WordPiece* [49] for word identification.

Table 4
Example of formatting input texts for pre-training with BERT.

Entry of two sentences	Entry of a sentence
[CLS] The man went to the store. [SEP] He bought a gallon of milk. [SEP]	[CLS] The man went to the store. [SEP]

Table 5
Example application of *tokenizer* provided by BERT.

Input sentence	"Here is the sentence I want embedding for."
Text after *tokenizer*	['[CLS]', 'here', 'is', 'the', 'sentence', 'i', 'want', 'em', '##bed', '##ding', '##s', 'for', '.', '[SEP]']

- A configuration file (*bert_config.json*) that specifies the model's hyperparameters.

Then, these two generic models go through a fine-tuning process, as presented in Section 4.4.

4.4. Fine-tuning

It is worth noting that the fine-tuning process consists of the use of a pre-trained embedding model (trained on a generic corpus) in an unsupervised way, which is adjusted, that is, retrained on a known corpus that is specific to the area of interest. In this case, the fine-tuning was performed on the generic models *word2vec_base* and *BERT_base*, using corpus *corpPret_SE* (shown in Table 2).

The pipeline of Fig. 4) was used to adjust and generate the representation of the texts *corp_SE*, applying the BERT model.

A fine-tuning of the generic model *word2vec_base* was performed using specific methods for this purpose provided by the *Gensim* library in the *Python* language. This process generated the *word2vec_SE* model. This adjusted model was used to generate the average representation (see Section 4.5) for each requirement text of the training and testing corpus (corp_SE).

The fine-tuning process of the pre-trained BERT model consists of two main steps [10]:

1. **Preparation of data for pre-training:** initially the input data is generated for pre-training. This is done by converting the input sentences into the format expected by the BERT model (using *create_pretraining_data* algorithm). As BERT can receive one or two sentences as input, the model expects an input format in which special tokens mark the beginning and end of each sentence, as shown in Table 4. In addition, the *tokenization* process needs to be performed. BERT provides its own *tokenizer*, which generates output as shown in Table 5.
2. **Application of the pre-training method:** the method used for pre-training by BERT (*run_pretraining*) became available by its authors. The necessary hyperparameters were informed, the most important being:
    - *input_file*: directory containing pre-formatted pre-training data (as per step 1).
    - *max_seq_length*: defining the maximum size of the input texts (set at 100).
    - *batch-size*: maximum lot size (set at 32, per use guidance of the pre-trained model *BERT_base*.

For the generic BERT model (Table 3), we opted for its version *Uncased_L-12_base*, here called *BERT_base* (Table 3). The fine-tuning process for the BERT model was performed as shown the 4.

The entire process, from data preparation to fine-tuning the BERT model, used the algorithms available in the repository https://github.com/google-research/bert, in which the authors [10] provides the full *framework* developed in the *Python* language.

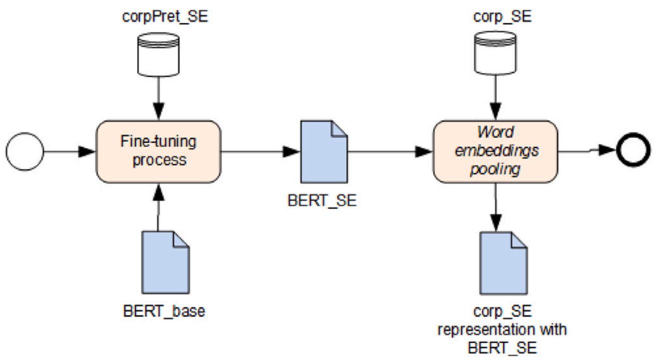

Fig. 4. Pipeline of the fine-tuning process of the *BERT_base* model and generation of the textual representation for the corp_SE.

Table 6
Adjusted pre-trained model applied to the performed experiments.

Pre-trained model	Specification
BERT_SE	Consists of the *BERT_base* model after fine-tuning with the corpus *corp_SE*.

After performing the fine-tuning, a new pre-trained models is available, as shown in the Table 6, which will compose the experiments.

It is noteworthy that the proposed model requires pre-training only for the embedding layer. This allows, for example, this pre-trained model is available for other software engineering tasks, or even for different effort estimation tasks. Thus, this pre-trained model may undergo successive adjustments, according to the need of the task to which it will be applied.

4.5. Obtaining characteristics

As for the contextualized embedding model, the textual representations generated by the BERT model (according to Fig. 4) present a different structure from the context-less embedding models (e.g. Word2Vec). This is primarily because the number of dimensions of the embedding vectors is not defined by the user, but by the model itself. Therefore, the number of dimensions of word embedding for the model *BERT_base* is defined in 768. In addition, each word in this model is represented by 12 layers (standard for *BERT_base*), with the need to pool [50] the embedding of some of the layers for each word. In order to define the pooling strategies to be applied, the article by [10] was taken as a basis.

Thus, one of the proposed strategies was used, considering that there were no significant differences between the results obtained with the other tested strategies. Thus, for the models *BERT_base* and *BERT_SE*, the strategy chosen was to use the penultimate layer of each word to generate a vector of average embedding for each sentence. More details on pooling strategies can be found in [50,51].

4.6. Exploratory data analysis

Before presenting the results of the effort estimation, it is first important to highlight some observable aspects regarding the vector of textual representation obtained after the embedding layer. These sentence embedding were generated from the models specified in Tables 3 and 6, that is, generic and fine-tuned models for both approaches (without context and contextualized).

In order to show the characteristic of the generated embedding, the *t-Distributed Stochastic Neighbor Embedding* (t-SNE) algorithm was applied. This algorithm has been used to represent complex data graphically and in smaller dimensions, while preserving the relationships

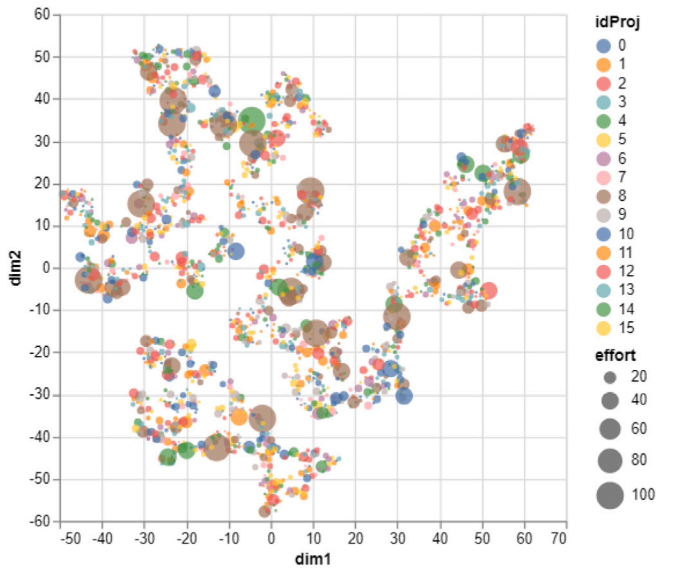

Fig. 5. Embedding generated by *BERT_SE*. The points represent the effort for each requirement, according to its size. The larger the point size, the greater the effort.

Table 7
Contextual similarity between grouped texts.

ID	Text
Texto 18994	For the **sqlserver and postgresql connection** cannot show the structure correctly.1. **create apostgresql sqlserver connection set** or not set the catalog parameter(does not set the textbfschema). 2. check the structure, only one **schema** show under each catalog. in fact i have several. please check it same issue as **informix db**.
Texto 17552	Sqoop - unable **to create job using merge command** as a user, i need to use xd **sqoop** module to support the merge command. currently, the sqoop runner create final arguments method forces the requirement for **connect**, **username and password** options which are not valid for the merge option. a check of the module type to not force these options being assigned to **sqoop arg list** would be preferred.
Texto 15490	Need **to create a persistent-job-registry** in order to hook up the **to get access** to all the jobs available the job registry has to be shared. Currently, the only **implementation** is the mapjobregistry. testability. the admin will need to see all **jobs created** by its containers.

between neighboring words [52], which greatly facilitates their understanding.

Thus, Fig. 5 shows sentence embedding generated from the *BERT_SE* for each textual requirement. Only 100 instances of requirements were included for each of the 16 projects analyzed (Table 1) (represented by "idProj"). The *t_SNE* algorithm reduced the 768-dimension model (BERT standard) to just two dimensions, which allowed for visual analysis of the representations obtained and, based on these representations, some conclusions were reached.

Table 7 present examples of requirement texts extracted from 2 different groups, according to Fig. 5. The texts, in each of the tables, have minimum distances between them.

In this example (Table 7), the context of the requirements presented is related to connection and database operations. Among the highlighted words (in bold), one can find, for example, "sqoop". This word is part of an application that transfers data between relational databases and *Hadoop*.[1] Therefore, the identification of similar contexts is more clearly perceived, even with very different words, as is the case of "sqoop", "sqlserver" and "persistent". These are different words, but part of the same context. Thus, although the groupings did not demonstrate clear groups, either by project or by effort, the groupings demonstrated, at a certain level, a representation of requirements from the same context, even if from different projects and/or efforts.

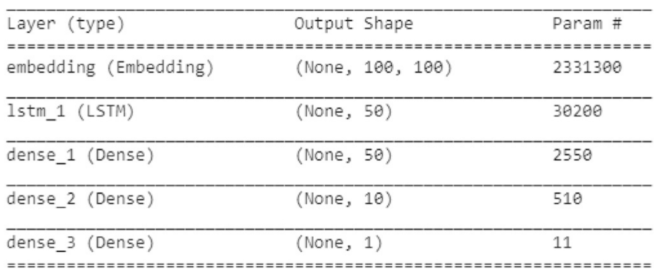
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 100, 100)          2331300
_________________________________________________________________
lstm_1 (LSTM)                (None, 50)                30200
_________________________________________________________________
dense_1 (Dense)              (None, 50)                2550
_________________________________________________________________
dense_2 (Dense)              (None, 10)                510
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 11
=================================================================
```

Fig. 6. Architecture *deep learning* with the pre-trained embedding layer using Word2Vec.

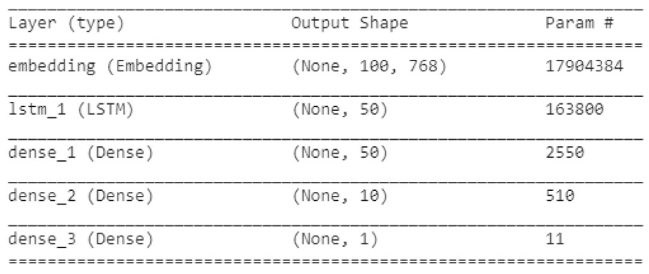
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 100, 768)          17904384
_________________________________________________________________
lstm_1 (LSTM)                (None, 50)                163800
_________________________________________________________________
dense_1 (Dense)              (None, 50)                2550
_________________________________________________________________
dense_2 (Dense)              (None, 10)                510
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 11
=================================================================
```

Fig. 7. Architecture *deep learning* with the pre-trained embedding layer using BERT.

4.7. Effort estimation model settings

In this section, the stages of the SE^3M model will be presented, in which representation vectors for each textual requirement are obtained according to the procedures presented in Section 4.5, are given as a parameter to the layers *dense* of the architecture *deep learning*, as shown in Figs. 6 and 7.

The *embedding()* layers (Figs. 6, and 7) are represented by the vector of pre-trained weights for each sentence. These vectors are processed by two dense nonlinear layers, followed by a linear regression layer. The output is the estimate of the predicted effort (e.g. story points).

Each instance of textual requirement submitted to the *Embedding ()* layer is represented by an average vector representing each sentence. A sentence consists of a maximum of 100 words, each of which is represented by 100 text embedding for the *Word2Vec* models and 768 dimension embedding according to BERT models, as justified below.

The maximum number of words per text was defined based on the representation of the histogram shown in Fig. 3, which shows that this number would include most of the sentences used in the training and test database, with reduced data loss.

As presented in the architecture specification SE^3M (Section 4), it is a very simple architecture, post layer of embedding, consisting only of two dense non-linear layers and a linear regression layer. The fact that the *deep learning* architecture is very simplified was purposeful, considering that the objective of this thesis is to present how to infer effort estimates in software projects by analogy using pre-trained embedding models, verifying whether these models are promising for text-based

[1] Hadoop is an open-source software platform for the storage and distributed processing of large databases. The services *Hadoop* provided include storage, processing, access, governance, security, and data operations, making use of powerful hardware architectures, which are usually on loan.

Table 8
Description of performed experiments.

Experiments	Pre-trained model	Sequential network architecture
E1	*word2vec_base*	Embedding() + LSTM + Dense (not-linear) + Dense (not-linear) + Dense (linear)
E2	*word2vec_SE*	Embedding() + LSTM + Dense (not-linear) + Dense (not-linear) + Dense (linear)
E3	*BERT_base*	Embedding() + LSTM + Dense (not-linear) + Dense (not-linear) + Dense (linear)
E4	*BERT_SE*	Embedding() + LSTM + Dense (not-linear) + Dense (not-linear) + Dense (linear)
E5	*BERT_SE*	Embedding() + LSTM + Dense (not-linear) + Dense (not-linear) + Dense (softmax)

software effort estimation. In this way, a more robust architecture could mask the results generated at each of the different network entrances.

A textual requirement also referred to in this study as a sentence, is represented by an average vector of the word embedding that composes this sentence. This average vector is generated from each of the generated models (Tables 3 and 6), considering the particularities of each applied approach (without context and contextualized), as presented in Section 4.5. Therefore, each generated embedding model is represented as a matrix, where each line represents an average embedding vector for a given textual requirement. Thus, each embedding model has the same number of samples, and what varies is the dimension applied. Previous tests were performed using the *grid search* method. For the embedding models without a context, tests were performed with dimensions 50, 100, and 200 with the best results presented by dimensions 50 and 100. As there was no significant variation for the MAE between both dimensions using the same neural network architecture, the number of dimensions of embedding was fixed at 100. For the *transformers* BERT the standard dimension defined for the *BERT_base* model was used, that is, equal to 768.

For the training of the learning model, it was necessary to configure some hyperparameters. Therefore, the *Adam* optimizer [53], learning rate 0.002 was used and the size of the *batch_size* was 128. Twenty epochs and an early stopping mechanism were defined, in which, if the MAE value remains stable for 5 epochs, the best results are averaged.

4.8. Experiment settings

The results obtained in this study will be presented below to make a comparative analysis with the most similar study [9] we identified, in addition to answering the following research questions defined initially.

5. Results and discussions

The results will be presented below, to make a comparative analysis with the most similar study [9], in addition to answering the following research questions defined initially.

RQ1. Does a generically pre-trained word embedding model show similar results to a software engineering pre-trained model?

To answer this question, experiments (**E1**, **E2**, **E3** and **E4**) were performed, with pre-trained models with and without fine-tuning, for context-less and contextualized approaches, a task that aimed to determine effort estimation. The approach consists of using a pre-trained embedding model as the only source of input in the deep learning architecture used.

Table 9 presents the average values for the MAE, MdAE, and MSE measurements obtained after the application of cross-validation *10-fold* during the execution of the proposed sequential architecture, in which each model was used as input for pre-trained embedding available for each of the experiments (according to Table 8).

As can be seen in Table 9, the models that underwent fine-tuning (with SE suffix), regardless of the approach used, gave better results for MAE, MSE, and MdAE. Thus, it is clear that a pre-trained embedding model, in which fine-tuning is performed with a specific corpus of the task domain, presents better performance than a pre-trained model with a generic corpus, as is the case with *word2vec_base* and *BERT_base*.

Table 9
Evaluation of the results obtained for experiments E1, E2, E3, and E4. In bold are the best results for each pre-trained embedding model (Model) in both approaches (context-less and contextualized. With the exception of Pred(25). For the other metrics used, the lower the value, the better the result.

Approach	Model	MAE	MSE	MdAE	Pred(25)
Context-less	word2vec_base	4.66 ± 0.14	100.26 ± 7.04	2.9	68.8 ± 2.91
	word2vec_SE	4.36 ± 0.31	89.9 ± 14.37	2.5	67.5 ± 1.45
Context.	BERT_base	4.52 ± 0.09	100.95 ± 7.3	2.7	68.1 ± 2.48
	BERT_SE	**4.25 ± 0.17**	**86.15 ± 1.66**	**2.3**	**68.4 ± 1.57**

This can be proven by comparing the generic context-less embedding model (*word2vec_base*) to the same fine-tuned model (*word2vec_SE*), where a 5% improvement is seen about the MAE value for the second model. Likewise, when applying the contextualized embedding model with fine-tuning (*BERT_SE*), an improvement of 7.8% for the MAE was observed concerning the generic model *BERT_base*. Thus, it is noted that, in general, the results achieved improved after adjusting the models with a specific corpus of the domain.

If the MAE value obtained for the best model in Table 9 (*BERT_SE*), which was 4.25, is applied in practice, this will indicate that, for a given user story, if the real effort is 5 story points, the estimated effort value could be between 1 and 9 story points. This is one reason why human participation in the estimation process is indicated to calibrate these effort values to new requirements.

In terms of MSE, the improvement was of 3.5% for the *word2vec_SE* model, when compared to *word2vec_base*, and 13.5% for MdAE. When evaluating MSE and MdAE for the contextualized approach, the fine-tuned model outperforms the generic model by 14.6% and 14.8%, respectively.

Thus, we conclude that the representation of the training and test model (*corp_SE*), when generated from pre-trained and adjusted embedding models, improves the performance of activities such as the ABSEE. To this end, it is estimated that the greater the volume and diversity of samples in the corpus used in fine-tuning (*corpPret_SE*, the better the results can be. Therefore, this fact must be considered when giving continuity to this task in future studies.

RQ2. Would embedding models generated by context-less methods (i.e. Word2Vec) be effective as models generated by contextualized methods (i.e. BERT)?

As stated by Ruder [11], "it only seems to be a question of time until pre-trained word embedding (i.e. word2vec and similar) will be dethroned and replaced by pre-trained language models (i.e. BERT) in the toolbox of every NLP practitioner". Thus, the objective of this study was to analyze if context-less embedding models are effective in effort estimates, as compared to contextualized embedding models in a specific corpus.

As can be seen in Table 9 the results obtained by the contextualized models (*BERT_base* and *BERT_ES*), surpass the results of the models without context.

When comparing MAE values, the *BERT_base* model shows a 3% improvement over *word2vec_base*. Likewise, for the values of MdAE and MSE, there was an improvement of 0.7% and 6.9%, respectively.

When comparing how these metrics between the models with fine-tuning (*Word2vec_ES* and *BERT_ES*), as improvements of the contextualized model with no context were 2.5%, 4.2% and 8% for the values of MAE, MSE, and MdAE, respectively. Looking at Pred(25), the BERT_SE results are very close to those obtained by BERT_base, but with a much smaller standard deviation, which gives the method a greater reliability in the executed predictions.

Table 10
Example of requirement texts that have polysemy — same words and different contexts.

Text 22 810	**Add** that contact as a favorite notice that the images for contacts (driven by remote URL) change unexpectedly. Under the covers, all that is happening is that the data of the list view is refreshed.
Text 23 227	**Add** qparam to skip retrieving metadata and graph edges if the qparam is not there, use current default behavior.

Table 11
Example of requirement texts that have ambiguity — different words in the same context.

Text 18	**Create** new titanium studio splash screen there is a placeholder image...
Text 22 810	**Build** the corporate directory app for ios...

Table 12
Mean Absolute Error (MAE) obtained for *BERT_SE* compared to the Deep-SE model [9].

Method	SE^3M (multi-repositories)	*Deep-SE* (cross-repository)
MAE	4.25 ± 0.17	3.82 ± 1.56

It is believed that this result is since methods context-less (ex. *Word2Vec*) allow representing a single context for a given word in a set of texts. This aspect causes a lot of information to be lost. In an effort estimate, based on the requirements texts (e.g. user stories), a contextualized strategy certainly produces better results. Unlike the *Word2vec* approach, BERT methods offer this dynamic context, allowing the actual contexts of each word to be represented in each text.

This aspect is important, as textual software requirements (i.e. user stories, use cases) generally have short texts and little vocabulary, which means that many words are common to the field of software engineering and are repeated in many texts. This is the importance of identifying different contexts of use for each word to differentiate them. Contextualized methods like BERT guarantee each word's dynamic treatment, intrinsically addressing polysemy problems and ambiguity.

This is possible because the contextualized models use a model of deep representation of each word in the text (that is, 12 or 24 layers), unlike the models of embedding context-less, which present a superficial representation (of a single layer) for a word. In addition, contextualized models use an attention model that allows verifying whether the same word occurred previously in the same context or not (example in Table 10), or whether different words can present the same context (e.g. create, implement, generate) - example in Table 11.

RQ3. Are pre-trained embedding models useful to a text-based software effort estimation?

To answer this question regarding the perspectives of contextualized pre-trained models applied to ABSEE, it is necessary to observe whether MAE, MSE, and MdAE obtained good results. As shown in Table 9, the best MAE value was 4.25, which means that a software effort of 6 will be predicted between 2 and 10.

When questioning whether this result is good or bad, can be observed that, considering the small number of samples in the training set and tests, the high degree of imbalance between classes, and the high variability of the text, this result is quite adequate. It is precisely due to the perception that, at least currently, there is little data to estimate the effort involved in software development that motivated these researchers to investigate pre-trained embedding models, to solve the proposed problem.

When comparing the best MAE results, obtained through the *BERT_SE* model, with the MAE results given by the Deep-SE [9] model, the latter of which was the study found to be most similar to the present research, some aspects stand out, as shown below.

Table 13
Number of textual requirements allocated to each Planning Poker class.

Label values	1	2	3	5	8	13	20	40	100	Total
Textual requirements	4225	3406	4809	4725	3588	1238	706	451	165	**23.313**

As can be seen in Table 12, the MAE obtained by *BERT_SE* was slightly higher than *Deep-SE*. However, one should note that to obtain this result, [9] inferred the effort estimates between projects (e.g. Moodle/Titanium, Mesos/Mule). Therefore, there is a large chance that two projects share a similar context, which would make predictability easier for projects in different contexts. This statement is reflected directly in the standard deviation of the MAE values (e.g. 5.37, 6.36, 5.55, 2.67, 4.24) for the Deep-SE between projects, which is 1.56. One can see that some values of MAE are relatively low (e.g. 2.67), while others are higher (ex. 6.36). This means that there may be a higher variation for the estimates inferred by Deep-SE, which, in the worst case, can cause a requirement whose effort is 7 story points, to return 1 or 13 story points. Thus, it is suggested that if the Deep-SE model is applied using a single repository approach, as well as the SE^3M, the MAE values may be even higher.

SE^3M, on the other hand, uses a single repository approach, that is, all requirements are independent of project or repository, aiming to generalize the model. Thus, although the SE^3M MAE is close to the Deep-SE value, the standard deviation obtained is smaller (0.17), or almost nonexistent (ex. 3.87, 4.21, 4.15, 4.12, 3.97, 4.25, 4.01), which can be proven by observing the pattern of MAE values obtained by the model. In this sense, it can be said that the proposed method is more generic and applicable to different problems, demonstrating a greater degree of reliability than Deep-SE. This is mainly due to the ability of the BERT *Transformer* mechanism (attention mechanism) to resolve long-term dependency and "vanishing gradient" [54,55], which presents itself as limitations in recurrent network architectures, as is the case with RHN and LSTM, used by [9]. The "vanishing gradient" is the loss of relevant context information, used to identify the semantics of a given word in a text. Therefore, the attention mechanism allows for one to ignore irrelevant information and focus on what is relevant, making it possible to connect two related words, even if they are not located one after the other.

In order to reinforce the positive trend of applying pre-trained embedding models in the process of inferring effort estimation, the **E5** experiment was performed. In this case, the same set of training and testing data was used, applying a classification layer (with Softmax activation) instead of linear regression. This required some modifications to the corpus. First, there was a need to make the corpus more homogeneous and bulky concerning the existing labels. Then the closest estimates were grouped, considering the series of *Fibonacci* proposed for the *Planning Poker* [6] estimation method. As a result of this process, the corpus only had 9 possible labels: 1, 2, 3, 5, 8, 13, 20, 40, 100. Therefore, compared to the regression problem, each label had its data volume increased and balanced to the existing classes, mainly for the smaller labels (between 1 and 8), as shown in Table 13.

The Fig. 8 shows the confusion matrix generated for the experiment **E5**. One can see that the greatest confusion occurs between the lowest efforts (1, 2, 3, 5, and 8). Considering the MAE value of the regression experiment (4.25), it is possible to understand this bias in the confusion matrix; after all, as explained previously, an effort of 4 story points could be 1 or 8. For example, class zero (0) errors are located at most up to class 4, that is, if the prediction is not fully assertive, it is at least approximate. Another example that can be mentioned is class 3, in which 77% of the correct answers are related to class 2, 3, or 5, errors that are considered acceptable within one a software development process.

One can also observe that, similarly, the larger classes (13, 20, 40, and 100) became more confused with each other and, in very

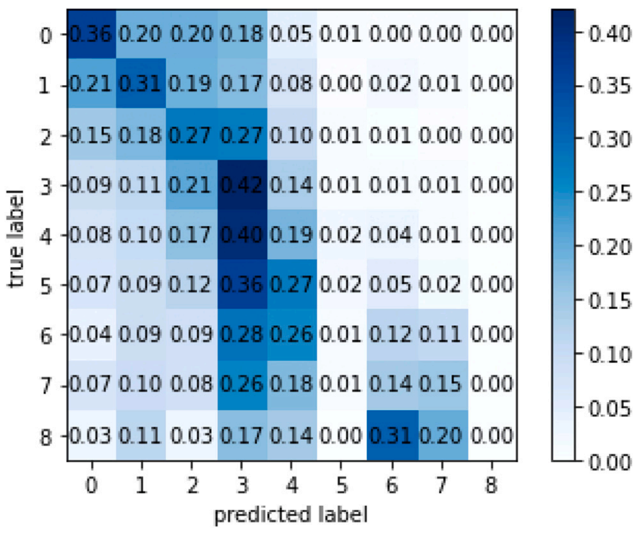

Fig. 8. Confusion matrix for the E5 experiment using the 9 classes from *Planning Poker*.

low percentages there was confusion among the smaller classes. This aspect leads us to suggest that human intervention at the end of the process is needed, to approve/modify the estimate generated, according to the user's working reality. Thus, considering its application for the end-user, the proposed method would be classified as semi-automated.

Another aspect to be observed in the results of Fig. 8 is an indication that the larger the corpus representing each of the labels, the better the results. This study argues that effective techniques for increasing data in texts can improve this result. Another alternative would be to collect a more significant number of real samples, complementing the pre-training and fine-tuning corpus *corpPret_SE*, as well as sets for training and testing (*corp_SE*). Thus, the generated embedding will be more representative, that is, they can better represent each situation found in the requirements.

RQ4. Are pre-trained embedding models useful to a text-based software effort estimation, both on new and existing projects?

Another aspect that can be observed is the indication of the SE^3M model to estimates new and non-existing requirements. For this, an additional experiment was carried out containing the same configuration as **E4**, and only changing the type of data partitioning. In this experiment *cross validation* was applied by projects to evaluate the results of inferring the estimates for each project, that is, for completely new projects. Thus, during each of the iterations, one of the projects was considered a target for the estimates (test set), while the others were considered a source (training set). Table 14 shows the results obtained for the MAE in each project.

When considering the cross-project estimation approach, if target projects are considered (UG, ME, AP, TI, AS, TI, MS, MU), as listed in Table 9 of the article by [9], the authors were able to an average MAE value of 3.82 for the Deep-SE model, while the SE^3M model presented a MAE of 3.4 (Table 15). It is observed that this comparison was made only with the results obtained in the mode between repositories of [9], and for the results obtained with SE^3M, all 15 projects were considered as source projects remaining, while the target projects are the same chosen by the authors.

It is still observed that the authors of Deep-SE used a source project for training and another target project for tests, that is, the diversity of characteristics that the learning method needs to deal with is limited only by the domain of a single project. In other words, the model needs to deal with less variability of data, which supposedly can facilitate learning, since some important relationships between these data can be discovered more easily by the learning method. However, the proposal presented here uses all projects (except one) for training, that is, the variety of relationships that the model must deal with is much greater when compared to the Deep-SE approach. From a certain point of view, this is good, as the learning method should learn a better generalization, for any type of problem presented. On the other hand, however, learning is hampered due to the curse of dimensionality, in which many important relationships can be more difficult to be inferred automatically, mainly due to the very small corpus. The explanation for this is that in a smaller corpus, basic relationships are easily learned, whereas more specific relationships are often not sufficiently representative.

Table 14
MAE values when estimating the effort for each project in relation to the others. The number of requirements by the project (Num. requirement), the average (Avg), and standard deviation (Std) of the effort by a requirement in each project are presented.

ID project	Description	Num. requirement	Avg	Std	MAE
AS	Appcelerator Studio	2919	5.63	3.32	2.50
AP	Aptana Studio	829	8.01	5.95	4.18
BB	Bamboo	521	2.41	2.14	2.76
CV	Clover	384	4.6	6.54	3.87
DM	Data Management	4667	9.56	16.6	7.78
DC	DuraCloud	666	2.12	2.03	3.79
JI	JIRA Software	352	4.43	3.51	3.13
ME	Mesos	1680	3.08	2.42	3.39
MD	Moodle	1166	15.54	21.63	11.99
MU	Mule	889	5.08	3.49	3.51
MS	Mule Studio	732	6.39	5.38	3.51
XD	Spring XD	3526	3.69	3.22	3.16
TD	Talend Data Quality	1381	5.92	5.19	4.04
TE	Talend ESB	868	2.16	1.49	3.42
TI	Titanium SDK/CLI	2251	6.31	5.09	3.49
UG	Usergrid	482	2.85	1.40	3.24

Table 15
Mean Absolute Error (MAE) in estimating effort for new projects.

Source projects	Target projects	MAE (Deep-SE)	MAE (SE^3M)
	UG	1.57	3.24
	ME	2.08	3.39
	AP	5.37	4.18
15 remaining projects	TI	6.36	3.49
	AS	5.55	2.50
	TI	2.67	3.49
	MS	4.24	3.51
	MU	2.70	3.51
Avg		3.82	**3.40**

In the case of software projects, each project used as training and testing data is in fact another domain (when compared to another project), which can be composed of several sub-domains, such as application areas of the project, programming languages (e.g. Java, Python, C), databases (e.g. SQL Server, Oracle, MySQL, MongoDB), application modalities (e.g. web, mobile, desktop), development teams with different characteristics (e.g. beginner, full), among others. Therefore, each sub-domain can be composed of different relationships, since different aspects can change the context of one project in relation to the other, which leads, for example, to the use of different terms for the same purpose. Thus, there is a need for a balanced volume of representative samples from each sub-domain, so that the model can learn properly.

Although the results of this study are not very different from that obtained in the article by [9], the proposed approach presents a much simpler and potentially more robust network architecture. This is possible because part of the *feature learning* process previously performed, which extracts the contextualized textual representation for new project requirements, is performed only once, during the process of generating the pre-trained model (*BERT_SE*). This model is then employed as a parameter in the embedding layer of the sequential architecture used. This aspect is important, considering that there is the need to feed the training corpus with new cases of requirements, as well as with the cases' respective efforts, which makes it possible to increase precision in effort estimates for new projects.

RQ5. Are the results found generalizable, aiming to generate estimates of effort between companies?

In our approach, the results show that even a new project can have its effort predicted without any pre-existing data (as shown in RQ4). Although the MAE still does not deliver a perfect result, a good estimate of the effort can be achieved and used, in a semi-automated way, by companies, as explained in the RQ3 response.

Thus, it is presumed that, since the SE^3M model is used by different companies, variability in the development effort will occur, considering the team experience, the technology used, among other factors that affect this metric. In this regard, it is not expected that the effort predictions are extremely accurate when you are starting to use the SE^3M model. Therefore, it is necessary a period of adaptation and evaluation of the amount of effort that 1 story point means to the team that will perform it. The adaptation model must occur by the feedback of real story points obtained by the team/company.

It is important to mention that the proposed model can, with some efficiency, differentiate a simple requirement of a complex, just making an adaptation of the value of a story point to the team/company, considering the development features of the working team.

Additionally, when considering applying the model to multiple companies, it is necessary to consider the possibility that the existing requirements contain different metrics (e.g. function points, story points) since the dataset would be fed by requirements from different companies. To answer this question, it is suggested convert these metrics into a standard form, which, if obtained from the estimator, could be converted into the format to be used by the user. This conversion of software effort metrics is proposed by [56].

6. Threats to validity

Thus as several research in the area of SE that apply machine learning techniques for use with texts, this paper propose a kind of software effort estimation analogy-based using requirement texts. This is a difficult approach, especially regarding the availability of real data. These data should reflect the reality of software projects in different areas, levels of complexity, uses of technologies, among other attributes. Typically, these difficulties are related to the volume of data, the language in which the data is given, the quality of the data (e.g. text formats, fullness, nonexistent labels, among others). This way, after accomplished a search for textual requirements corpus, we decided to use the same applied by [9], which provided the first dataset at the level of requirements for the realization of research in SE area. Since the model proposed in this article (SE^3M) aims to compare results obtained for estimating software effort with the Deep-SE model, proposed by [9], it was defined by using the same dataset.

Therefore, actions to containing or reducing threats to validity carried out by [9], were adopted and maintained for this work. They are:

- It is used the actual project requirements data, which were obtained from large and different open source projects.
- The story points that accompany each textual requirement, were first estimated by human teams, and therefore, may not be accurate in some situations. Choetkiertikul et al. [9] performed two tests to mitigate this threat: one with the original story points, the other with normalized and adjusted story points. With that it was verified that the proportions of story points attributed to each requirement were adequate.
- It was observed that project managers and analysts determine the estimate for a new requirement, based on their comparison with requirements already implemented in the past, and thus carry out the estimate consistently. In this way, the problem is indicated for a machine apprentice, since the training and testing database presents the description of the requirements, accompanied by their respective efforts in story points. Thus, new requirements can be estimated, as long as they have these two attributes.

In order to perform the experiments, appropriate metrics were applied to evaluate regression models, that are commonly used to evaluate models of software effort estimates [9] which attest to the validity of the model. In addition, different forms of data partitioning (*cross-validation*) were used to validate the results obtained.

To compare the results obtained in this work with those obtained by [9], considering that our implementation may not present all the details that Deep-SE presents, our model was tested using the same corpus as the authors. Thus, it was possible to state that our results are consistent.

Aimed at validating the possibility of generalization, it should be noted that the training and testing corpus is composed of 23.313 requirements from sixteen open source projects, which differ significantly in size, complexity, developer team and community [9]. It is observed that open source projects do not present the same aspects as commercial projects in general, especially in relation to the human resources involved, which requires more research. It would be prudent to test the SE^3M model with a database of commercial projects only (with data available containing the same attributes used in this research) and then with all types of projects (commercial and open source) to check for significant differences.

7. Conclusions and future work

The main objective of this research is to evaluate if pre-trained embedding models are promising for the inference of text-based software effort estimation, evaluating two approaches to embedding: contextless and contextualized. The study obtained positive results for the pre-trained models aiming the ABSEE task, particularly the contextualized models, such as BERT. As predicted in the literature, the contextualized methods demonstrate the best performance numbers. In addition, we show that fine-tuning the embedding layers can help to improve the results. All of these results can be improved, especially if trained with more data and/or using some effective data augmentation.

The researchers observed that the database was a limitation of this research, particularly because it is not very bulky, which prevents the results from being even better, especially when using cutting-edge NLP techniques (e.g. pre-trained models, fine-tuning and deep learning). When observing the volume of data used in fine-tuning tasks for specific domains, such as [57,58], it is clear that even though the domain's base is considered to be light, they contain billions of words. On the other hand, the domain-specific database applied in the fine-tuning of the *BERT_SE* model, has around 800 thousand words. Thus, it was observed that the results obtained with the use of BERT could be improved if there was a more voluminous and diverse set of data, in which it was possible to better adjust the model for different problems or even train a model of its own (from scratch), which would present an even greater level of adjustment according to the domain area.

Thus, this study argues that the SE^3EM model analyzed in this article can best adapt to different project contexts (for example, agile development). Estimation of story points or another similar estimation metric (e.g. use case points or function points) has a fine granularity (i.e. they are assigned to each user requirement). But this same inference method can be applied to estimate a coarser granularity element. An example would be a sprint in agile models [59], which is estimated by the sum of the smaller tasks that compose it.

Compared with the results obtained by Choetkiertikul et al. [9], the most similar work considering the objectives, note that the SE^3M used a pre-trained contextualized embedding layer, which went through a fine-tuning process, without the need to add any noise to the texts. In addition, the proposed architecture is more simplified, with just one recurring layer. As a result of the feature learning process, which applies a contextualized embedding approach, we have a more generalizable and multi-project method, which can be applied even in new projects. Thus, the method has more flexibility regarding the format of the input and multi-project texts, allowing to perform the inference to any new requirement, even during the initial stage of development.

Using the pre-trained BERT model (even the generic one), there is no need for prior training of a specific corpus or that requires a large volume of data. This has the advantage of involving no pre-training cost (which takes days [11]). Rather, there is only the need of fine-tuning, which takes a few hours [10]. That is, there is no need for training from scratch, as performed by [9].

Thus, we provide the BERT_SE model, which can be used in various SE tasks, allowing further adjustments, if necessary. In addition, the SE^3M model can be applied generically, being reliable and with low cost, and providing a computationally fast solution. All this, due to the fine-tuning process.

The results demonstrated that this is a promising research field with many available resources and room for innovation. Therefore, several research possibilities are presented as future work:

- Collect more textual requirement data to balance the corpus against existing labels, and thereby increase the number of contexts to improve results.
- update BERT_base vocabulary, including specific vocabulary, and then perform the fine-tune.
- perform fine-tuning, also with model BERT_large, and compare the results.
- studying and applying effective data augmentation techniques in order to balance the number of samples in each existing effort class, and thus obtain possible improvements in the results.
- studying and applying different combinations of the layers in the BERT model, in order to evaluate the performance of the model regarding fine-tuning and its effects on ABSEE.
- studying and applying pre-trained "light" model (e.g. ALBERT [60]) to the SE^3M model, and evaluate its performance aiming to carry out ABSEE.

CRediT authorship contribution statement

Eliane Maria De Bortoli Fávero: Conceptualization, Investigation, Methodology, Visualization, Data curation, Software, Writing – review & editing. **Dalcimar Casanova:** Investigation, Methodology, Visualization, Writing – original draft, Validation, Supervision, Writing – review & editing. **Andrey Ricardo Pimentel:** Visualization, Supervision, Validation, Writing – review & editing.

Declaration of competing interest

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

References

[1] A. Idri, M. Hosni, A. Abran, Systematic literature review of ensemble effort estimation, J. Syst. Softw. 118 (2016) 151–175.
[2] M. Kazemifard, A. Zaeri, M.A. Nematbakhsh, F. Mardukhi, Fuzzy emotional COCOMO II software cost estimation (FECSCE) using multi-agent systems, Appl. Softw. Comput. J. 11 (12) (2011) 2260–2270.
[3] M. Azzeh, Adjusted case-based software effort estimation using bees optimization algorithm, Knowl.-Based Intell. Inform. Eng. Syst. (2011) 315–324.
[4] V.K. Bardsiri, D.N.A. Jawawi, A.K. Bardsiri, E. Khatibi, LMES: A localized multi-estimator model to estimate software development effort, Eng. Appl. Artif. Intell. 26 (10) (2013) 2624–2640.
[5] V.-S. Ionescu, An approach to software development effort estimation using machine learning, in: Intelligent Computer Communication and Processing (ICCP), 2017 13th IEEE International Conference on, IEEE, 2017, pp. 197–203.
[6] M. Cohn, Agile Estimating and Planning, Pearson Education, 2005.
[7] T. Mikolov, K. Chen, G. Corrado, J. Dean, Efficient estimation of word representations in vector space, 2013, arXiv preprint arXiv:1301.3781.
[8] J. Pennington, R. Socher, C. Manning, Glove: Global vectors for word representation, in: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), 2014, pp. 1532–1543.
[9] M. Choetkiertikul, H.K. Dam, T. Tran, T.T.M. Pham, A. Ghose, T. Menzies, A deep learning model for estimating story points, IEEE Trans. Softw. Eng. (2018).
[10] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova, Bert: Pre-Training of Deep Bidirectional Transformers for Language Understanding, North American Association for Computational Linguistics (NAACL), 2019.
[11] J. Howard, S. Ruder, Universal language model fine-tuning for text classification, 2018, arXiv preprint arXiv:1801.06146.
[12] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, Ł. Kaiser, I. Polosukhin, Attention is all you need, in: Advances in Neural Information Processing Systems, 2017, pp. 5998–6008.
[13] J. Shivhare, S.K. Rath, Software effort estimation using machine learning techniques, in: Proceedings of the 7th India Software Engineering Conference, ACM, 2014, p. 19.
[14] E. Kocaguneli, G. Gay, T. Menzies, Y. Yang, J.W. Keung, When to use data from other projects for effort estimation, in: Proceedings of the IEEE/ACM International Conference on Automated Software Engineering, ACM, 2010, pp. 321–324.
[15] B. Boehm, C. Abts, S. Chulani, Software development cost estimation approaches—A survey, Ann. Softw. Eng. 10 (1-4) (2000) 177–205.
[16] S. Choi, S. Park, V. Sugumaran, A rule-based approach for estimating software development cost using function point and goal and scenario based requirements, Exp. Syst. Appl. 39 (1) (2012) 406–418.
[17] D. Manikavelan, R. Ponnusamy, Software cost estimation by analogy using feed forward neural network, in: Information Communication and Embedded Systems (ICICES), 2014 International Conference on, IEEE, 2014, pp. 1–5.
[18] N.-H. Chiu, S.-J. Huang, The adjusted analogy-based software effort estimation based on similarity distances, J. Syst. Softw. 80 (4) (2007) 628–640.
[19] M. Shepperd, C. Schofield, B. Kitchenham, Effort estimation using analogy, in: Proceedings of IEEE 18th International Conference on Software Engineering, IEEE, 1996, pp. 170–178.
[20] A. Idri, I. Abnane, A. Abran, Missing data techniques in analogy-based software development effort estimation, J. Syst. Softw. 117 (2016) 595–611.
[21] C. Fellbaum, The Encyclopedia of Applied Linguistics, WordNet, 2012.
[22] S. Raschka, Python Machine Learning, Packt Publishing Ltd, 2015.
[23] P. Bojanowski, E. Grave, A. Joulin, T. Mikolov, Enriching word vectors with subword information, Trans. Assoc. Comput. Linguist. 5 (2017) 135–146.
[24] Z. Haj-Yahia, A. Sieg, L.A. Deleris, Towards unsupervised text classification leveraging experts and word embeddings, in: Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, 2019, pp. 371–379.
[25] A.M. Dai, Q.V. Le, Semi-supervised sequence learning, in: Advances in Neural Information Processing Systems, 2015, pp. 3079–3087.
[26] M.E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, L. Zettlemoyer, Deep contextualized word representations, 2018, arXiv preprint arXiv:1802.05365.
[27] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever, Improving language understanding by generative pre-training, 2018, URL https://s3-us-west-2.amazonaws.com/openai-assets/researchcovers/languageunsupervised/languageunderstandingpaper.pdf.
[28] B. McCann, J. Bradbury, C. Xiong, R. Socher, Learned in translation: contextualized word vectors, in: Advances in Neural Information Processing Systems, 2017, pp. 6294–6305.
[29] K. Clark, M.-T. Luong, C.D. Manning, Q.V. Le, Semi-supervised sequence modeling with cross-view training, 2018, arXiv preprint arXiv:1809.08370.
[30] O. Melamud, J. Goldberger, I. Dagan, context2vec: Learning generic context embedding with bidirectional lstm, in: Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning, 2016, pp. 51–61.
[31] Y. Zhu, R. Kiros, R. Zemel, R. Salakhutdinov, R. Urtasun, A. Torralba, S. Fidler, Aligning books and movies: Towards story-like visual explanations by watching movies and reading books, in: Proceedings of the IEEE International Conference on Computer Vision, 2015, pp. 19–27.
[32] P. Abrahamsson, I. Fronza, R. Moser, J. Vlasenko, W. Pedrycz, Predicting development effort from user stories, in: 2011 International Symposium on Empirical Software Engineering and Measurement, IEEE, 2011, pp. 400–403.
[33] I. Hussain, L. Kosseim, O. Ormandjieva, Approximation of COSMIC functional size to support early effort estimation in Agile, Data Knowl. Eng. 85 (2013) 2–14.
[34] K. Moharreri, A.V. Sapre, J. Ramanathan, R. Ramnath, Cost-effective supervised learning models for software effort estimation in agile environments, in: 2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC), 2, IEEE, 2016, pp. 135–140.
[35] C. Zhang, S. Tong, W. Mo, Y. Zhou, Y. Xia, B. Shen, ESSE: an early software size estimation method based on auto-extracted requirements features, in: Proceedings of the 8th Asia-Pacific Symposium on Internetware, ACM, 2016, pp. 112–115.
[36] M. Ochodek, Functional size approximation based on use-case names, Inform. Softw. Technol. 80 (2016) 73–88.

[37] T.E. Ayyildiz, A. Koçyigit, A case study on the utilization of problem and solution domain measures for software size estimation, in: 2016 42th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), IEEE, 2016, pp. 108–111.
[38] Y. Bengio, A.C. Courville, P. Vincent, Unsupervised feature learning and deep learning: A review and new perspectives, 1 (2012) 2012. CoRR, arXiv:1206.5538.
[39] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, 2014, arXiv preprint arXiv:1409.1556.
[40] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 770–778.
[41] A. Graves, Supervised sequence labelling, in: Supervised Sequence Labelling with Recurrent Neural Networks, Springer, 2012, pp. 5–13.
[42] H. Sak, A. Senior, F. Beaufays, Long short-term memory recurrent neural network architectures for large scale acoustic modeling, in: Fifteenth Annual Conference of the International Speech Communication Association, 2014.
[43] F. Sarro, A. Petrozziello, M. Harman, Multi-objective software effort estimation, in: 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE), IEEE, 2016, pp. 619–630.
[44] M. Shepperd, S. MacDonell, Evaluating prediction systems in software project estimation, Inform. Softw. Technol. 54 (8) (2012) 820–827.
[45] T. Menzies, E. Kocaguneli, B. Turhan, L. Minku, F. Peters, Sharing Data and Models in Software Engineering, Morgan Kaufmann, 2014.
[46] S. Porru, A. Murgia, S. Demeyer, M. Marchesi, R. Tonelli, Estimating story points from issue reports, in: Proceedings of the The 12th International Conference on Predictive Models and Data Analytics in Software Engineering, 2016, pp. 1–10.
[47] D. Zwillinger, S. Kokoska, CRC Standard Probability and Statistics Tables and Formulae, CRC Press, 1999.
[48] T.C. Scott, P. Marketos, On the Origin of the Fibonacci Sequence, MacTutor History of Mathematics, 2014.
[49] B. Zhang, Google's neural machine translation system: Bridging the gap between human and machine translation, 2016, arXiv preprint arXiv:1609.08144.
[50] G. Lev, B. Klein, L. Wolf, In defense of word embedding for generic text representation, in: International Conference on Applications of Natural Language to Information Systems, Springer, 2015, pp. 35–50.
[51] J. Alammar, The Illustrated BERT, ELMo, and co.(How NLP Cracked Transfer Learning), December 3 (2018) 2018.
[52] L.v.d. Maaten, G. Hinton, Visualizing data using t-SNE, J. Mach. Learn. Res. 9 (Nov) (2008) 2579–2605.
[53] D.P. Kingma, J. Ba, Adam: A method for stochastic optimization, 2014, arXiv preprint arXiv:1412.6980.
[54] S. Hochreiter, Untersuchungen zu dynamischen neuronalen Netzen, Diploma Tech. Univ. München 91 (1) (1991).
[55] Y. Bengio, P. Simard, P. Frasconi, Learning long-term dependencies with gradient descent is difficult, IEEE Trans. Neural Netw. 5 (2) (1994) 157–166.
[56] R. Pressman, B. Maxim, Engenharia de Software-8ª Edição, McGraw Hill Brasil, 2016.
[57] I. Beltagy, K. Lo, A. Cohan, SciBERT: A pretrained language model for scientific text, in: Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), 2019, pp. 3606–3611.
[58] J. Lee, W. Yoon, S. Kim, D. Kim, S. Kim, C.H. So, J. Kang, BioBERT: a pre-trained biomedical language representation model for biomedical text mining, Bioinformatics 36 (4) (2020) 1234–1240.
[59] M. Paasivaara, S. Durasiewicz, C. Lassenius, Using scrum in distributed agile development: a multiple case study, in: 2009 Fourth IEEE International Conference on Global Software Engineering, IEEE, 2009, pp. 195–204.
[60] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, R. Soricut, Albert: A lite bert for self-supervised learning of language representations, 2019, arXiv preprint arXiv:1909.11942.