Received 5 June 2023, accepted 26 June 2023, date of publication 28 June 2023, date of current version 7 July 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3290478

RESEARCH ARTICLE

Toward Automatic Tutoring of Math Word Problems in Intelligent Tutoring Systems

PABLO ARNAU-GONZÁLEZ[1], **ANA SERRANO-MAMOLAR**[2], **STAMOS KATSIGIANNIS**[3], **(Member, IEEE)**, **TURKE ALTHOBAITI**[4], **AND MIGUEL AREVALILLO-HERRÁEZ**[1]

[1]Departament d'Informática, Universitat de Valéncia, Burjassot, 46100 Valencia, Spain
[2]Departamento de Lenguajes y Sistemas Informáticos, Universidad de Burgos, 09006 Burgos, Spain
[3]Department of Computer Science, Durham University, DH1 3LE Durham, U.K.
[4]Faculty of Science, Northern Border University, Arar 91431, Saudi Arabia

Corresponding author: Pablo Arnau-González (pablo.arnau@uv.es)

This work was supported in part by the Ministry of Science and Innovation (Strategic Projects Focused on the Green and Digital Transition) under Project TED2021-129485BC42 and Project TED-2021-129485B-C43; in part by the Ministerio de Cultura e Innovacion (MCIN)/Agencia Estatal de Investigación (AEI)/10.13039/50 1100011033 and "European Regional Development Fund (ERDF)–A way of making Europe" under Project PGC2018-096463-B-I00; in part by the Valencian Regional Government, Spain, under Project AICO/2021/019, and grant CIAPOS/2022/163; in part by the Margarita Salas 2022–2024 Grant awarded by Universitat de València and Universidad de Burgos, funded by the Spanish Ministry of Science, Innovation and Universities through NextGenerationEU funds, under Grant MS21-29; and in part by the Project "AGENCY," funded by the Engineering and Physical Sciences Research Council, U.K, under Grant EP/W032481/1. The authors extend their appreciation to the deanship of Scientific research at Northern Border University, Arar, KSA for funding this research work through the project number "NBU-FFR-2023-0101".

ABSTRACT Math Word Problem (MWP) solving, which involves solving math problems in natural language, is a prevalent approach employed by Intelligent Tutoring Systems (ITS) for teaching mathematics. However, one major drawback of ITS is the complexity of encoding all potential solutions for each problem supported, which is both time-consuming and labour-intensive. In this study, we propose a novel method for automatically converting the statement of a previously unseen MWP into the internal representation of an ITS, thereby simplifying the task of adding new MWPs by only requiring the problem statement. To accomplish this, we propose the use of large pre-trained language models to translate the problem into Python code, which can then be easily imported into an ITS. Experimental results indicate that this approach is effective and suitable for the task, and as language models continue to improve, the accuracy rates are expected to increase further.

INDEX TERMS Math word problems, algebra tutoring, intelligent tutoring systems, automatic code generation.

I. INTRODUCTION

A Math Word Problem (MWP) is a mathematical problem that is expressed in natural language. Solving a MWP is the task of providing a numerical solution to it [3]. For example, "*A has five apples. B has two times the apples of A. How many apples does B have?* $\rightarrow$ Answer: 10 apples". To obtain the numerical solution for the MWPs, it is essential to accurately identify the quantities mentioned in the problem statement and their interrelationships [13]. MWPs have been extensively used in Intelligent Tutoring Systems (ITS) to facilitate the teaching of mathematics and arithmetic skills by emulating human tutor tasks such as, among others, the provision of conceptual schemes needed for problem statement interpretation, the assessment of the validity of the process used to solve the problem, and the adaptation of the problem sequence according to the individual characteristics of the learner [1].

The principal objective of an ITS is to provide support throughout the learning process of a specific subject. However, ITS have always suffered from one major limitation, which is that they are only capable of supervising the resolution process through problems or exercises that have been previously registered in the system's own knowledge

The associate editor coordinating the review of this manuscript and approving it for publication was Gustavo Olague.

representation schema. Therefore, creating more extensive sets of problems demands a considerable amount of time from experts, not only to generate fresh problems but also to analyse all potentially valid solution paths and encoding them into the ITS's internal representation. In the case of ITS designed to supervise the solution of arithmetical MWPs, this task can be partly automated by automatically generating the ITS's internal representation from the problem statement. As a result, the human expert is left with the creative task of developing new problem statements.

Hypergraph-Based Intelligent Tutoring System (HINTS) was developed during the past decade [2], and was a milestone in the advancement of ITS for algebraic and arithmetic word problem-solving. The system adopted a non-guided approach and gave the learner greater autonomy by letting them define new quantities as required, without imposing any restriction on the solution path. To enable supervision, the expert had to encode the problem solution in the form of a bipartite graph with operands on one partition and quantities on the other partition. This encoding was arduous and time-consuming and had to be completed for each individual exercise. The automation of this task was not feasible with the available resources at the time, but advances in Natural Language Processing (NLP) and machine learning have provided with some promising potential solutions for addressing this issue.

In recent years, the field of NLP and, specifically, text generation, has seen significant advances due to the emergence of Large Language Models (LLMs). These tools and algorithms have achieved results that are close to human-like in many tasks. Large models of code, i.e. large language models trained on source code from a programming language, have recently shown great potential in code completion and code synthesis tasks based on natural language input [32].

Various large code models, such as Salesforce's CodeGen [21] and Facebook's InCoder [9], have recently been released. These models are trained on a variety of Python code repositories, mainly for the task of code synthesis, although they are also trained for other code-related tasks, such as docstring generation, infilling, or return type inference. Such models are now widely used in commercial applications, with the most notable being GitHub's CoPilot,[1] a code development assistant integrated into the development environment that is trained on billions of lines of code and can turn natural language prompts into coding suggestions across dozens of languages.

In this work, we aim to make progress towards the automation of the process of encoding the problem solution from a problem statement written in natural language. To achieve this goal, we present a two-stage process. As a first step, we use a large code model to generate an intermediate representation of the problem solution in the form of Python source code.

[1]https://copilot.github.com/

This intermediate representation serves as a bridge between the natural language problem statement and the solution specification as a bipartite graph, which is done in a second stage, by using compiler technology. This automation has multiple relevant benefits and learning implications. For example, it would allow learners to specify the problem statement they want to be supervised on; and experts to easily add new exercises to the ITS by simply providing the problem statement, which saves the effort of formulating the problem solution in the required specification format.

Our experimental evaluation on a publicly available dataset of 1,000 MWPs showed that the proposed method is able to correctly encode up to 39% of the provided problems. These findings align with other results presented in the state-of-the-art literature on similar MWP solving approaches, while the proposed method can also facilitate the automation of MWP encoding for ITS.

II. RELATED WORK

The resolution process of MWP constitutes a field that has been extensively researched in the literature [34]. In a first stage of the evolution of MWP solving methods, early works employed handcrafted features and probabilistic approaches [3], [14], [20], [26], heavily relying on human interventions and therefore being able to resolve a limited number of pre-defined scenarios. This was followed by a second stage that focused on using templates for matching problems into semantic expressions [11], [24] to enable easier quantitative reasoning. More recent works moved towards generating a tree that could solve the problem [30], [36] by transforming the derivation of the arithmetic expression into an equivalent tree structure step-by-step in a bottom-up manner. Another recent approach consists of using a sequence-to-sequence (Seq2Seq) methodology for the generation of the mathematical formula needed for solving the problem [6], [27]. Various works improved upon the Seq2Seq approach using template-based approaches [29], reinforcement learning [12], [28], or group attention [16]. To this end, the solving of MWPs was formulated as a reasoning pipeline that transforms natural language descriptions of MWPs into mathematical formulas that can be used to solve the problem following an encoder/decoder approach [30], [33], [36], achieving further improvement in MWP solving performance by incorporating hard constraints into the decoder [25], [35].

With the emergence of Deep Learning, researchers have exploited the generalisation capabilities of deep models for providing different solutions. Graph convolutional networks (GCNs) were used by Li et al. [17] and Zhang et al. [36] in order to model the relations between different quantities in the problem, following a graph-to-tree approach, whereas Jie et al. [13] proposed a deductive reasoner on top of a BERT-like [8] transformer, which showed impressive results, as it is capable of accurately identifying quantities and the operation relations between them while improving the current state of the art on the SVAMP [23] MWP dataset. The use of

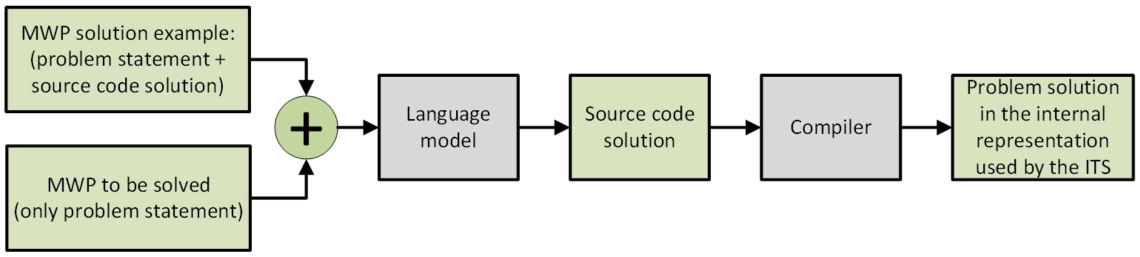

MWP solution example: (problem statement + source code solution)
MWP to be solved (only problem statement)
+
Language model
Source code solution
Compiler
Problem solution in the internal representation used by the ITS

FIGURE 1. Overview of the proposed end-to-end method for solving MWPs and integrating them in a given ITS.

pre-trained language models such as BERT led to significant improvement in the quality of tree expressions for MWP solving [15], [19].

The current trend in the field of NLP is to constantly increase the number of parameters of Language Models [4], [7], which has led to the emergence of "Large Language Models" (LLM). These models have been trained in vast amounts of text from the internet, and excel at human-like production of text in a wide variety of tasks with only minimal examples [4]. Among these tasks, LLM trained or fine-tuned on source code have demonstrated impressive capabilities in code generation, leading researchers into exploring LLM for code, or simply Large Code Models [9], [21]. LLM are trained specifically on enormous collections of source code and can generate programs from both natural language descriptions and other pieces of code. These models can be of great help to professional coders, such as by auto-completing code segments, creating docstrings for a given function body and signature, or recommending unit tests for a codebase.

By leveraging Large Code Models, this work aims to take a step further on the state of the art in the field of MWP solving for integration in ITS.

III. METHODOLOGY

The HINTS or similar ITS can supervise and support a student through the resolution process of any algebraic/arithmetic MWP, as long as the problem itself is registered in the system and properly encoded using the ITS's own knowledge representation schema. The still remaining main challenge of this approach is to process the problem expressed in natural language and represent it using the internal representation schema, i.e. the hypergraph. To this end, we propose processing the MWP using a Large Language Model that can output a source code solution, and then compiling the generated source code to the ITS's internal knowledge representation schema. An overview of the proposed methodology is illustrated in FIGURE 1, where computational processes are shown in grey boxes and the green colour is used to represent inputs and outputs.

A. INTERMEDIATE SOURCE-CODE REPRESENTATION

Given a natural language problem statement S composed by n words w_i, $S = \{w_0, w_1, \ldots, w_{n-1}\}$, our proposal is to generate a Python script that first defines a series Q_S of m quantities from the problem S, therefore, $Q_S = \{q_0, q_1, \ldots, q_{m-1}\}$, and then computes the numerical answer to the problem y_S. By proposing a solution expressed in source code, we can construct a graph exploiting the mathematical relation between the identified quantities. Moreover, the source code representation allows the naming of the identified quantities, so that the generated graph can be used to semantically match a user input to the correct quantities, allowing the system to assign quantities to variables with a simple command, such as "*let x be the number of rabbits*". The generated code, is therefore, an intermediate representation of the MWP, that is simple to understand and that can be converted to the internal representation schema of HINTS, or any other arithmetic ITS for that matter, automatically.

In order to produce the Python source code that can solve the posed problem, we propose to initialise the language model's prompt with one example of how the model is expected to provide the output code, followed by the posed unsolved problem, as shown in FIGURE 2a. To this end, the problem statements are introduced to the model as a code comment followed by a definition of a function called `sol()` with no input parameters. The `sol()` function defines all the known quantities as entries in a Map-like structure. In this particular case, we use the dictionary data structure that is defined as a Python built-in. Furthermore, in the initialised prompt, we solve the example `sol()` function so that one operation is defined per line until the definition of the solution and the return statement of the said solution. This allows the compiler to identify the quantities that have to be calculated (can be one or more, since Python allows to return tuples) as the ones returned by the `sol()` function, and also establishes the relationships between the quantities in the problem. Then, the output of the model consists of the source code of the `sol()` function for the unsolved MWP, as shown in FIGURE 2b. Finally, the generated `sol()` function is compiled into an XML file following the HINTS's internal knowledge representation. It must be noted that this last step can be customised in order to work with internal representations used by various ITS.

The pre-trained LLMs used in this work are (i) Saleforce's CodeGen [21] and (ii) Facebook's InCoder [9]. For CodeGen, the "mono" version was used, which is a transformer trained on The Pile [10], a general text corpus, fine-tuned on

```
""" A book has 3 chapters. The first chapter is 91 pages long
the second chapter is 23 pages long and the third chapter is
25 pages long. How many more pages does the first chapter have
than the second chapter? """
def sol():
    context = dict()
    context['number of chapters'] = 3
    context['number of pages first chapter'] = 91
    context['number of pages sencond chapter'] = 23
    context['number of pages third chapter'] = 25
    context['pages more first chapter'] = (
        context['number of pages first chapter']
        - context['number of pages second chapter']
    )
    return context['pages more first chapter']


""" An industrial machine worked for 5 minutes. It can make 4 shirts a minute.
How many shirts did the machine make? """
def sol():
```

(a) Prompt example and problem to be solved

```
    context = dict()
    context['number of minutes'] = 5
    context['shirts per minute'] = 4
    context['number of shirts'] = (
        context['number of minutes']
        * context['shirts per minute']
    )
    return context['number of shirts']
```

(b) Generated code

FIGURE 2. (a) Example of an input, with an initialised prompt containing an example of a solved problem together with the statement of the problem to be solved; and (b) output source code generated by the model.

BigQuery[2] [21], a corpus containing source code in a variety of popular programming languages, and further fine-tuned in BigPython [21], a corpus containing source code written exclusively in Python. The CodeGen model is trained as a Causal Language model, which means it is trained to predict the next token based exclusively on the previous tokens. InCoder is trained on a variety of languages. Although the training corpus is not fully disclosed, the authors mention that it is trained on code extracted from GitHub repositories and StackOverflow answers. The InCoder model is trained to perform the tasks of type inference, variable naming, docstring generation, and multi-region infilling. In this work, the 350 Million, 2 Billion, 6 Billion, and 16 Billion parameters model variations of the CodeGen model, and the 1 Billion and 6 Billion parameters model variations of the InCoder model were used for the experimental evaluation.

B. CONVERSION TO HINTS INTERNAL REPRESENTATION SCHEMA

The process explained above is able to generate a solution for a given problem, but the ultimate goal of our research is to be able to provide scaffolding for tutoring a problem proposed by the student. To this end, once an intermediate source code representation has been obtained, this needs to be translated into the knowledge representation used by the ITS. In the particular case of HINTS, it relies on a hypergraph representation of the problem solution [2]. This representation takes into account the problem quantities and creates hyperedges which represent the relations that exist between the quantities. In the specific case of HINTS, hyperedges link at least three different nodes: 2 operands, and 1 result node. Nevertheless, it is not limited to ternary relations. This representation is aligned with the code produced by the LLM used in this work but needs to be translated from one format to another.

The translation is approached by using compiler technology. In particular, a combination of the jflex and java cup tools has been used to generate the output in the format required by HINTS. This is a common combination typically used in compiler construction courses [22] to teach the syntax-driven approach. First, a lexical analyser generated by using jflex identifies the basic elements of the language, namely the keywords, symbols and quantity names. Then, a syntax analyser created by using java cup is used to parse the output of the lexical analyser, determine if the code conforms to the expected syntax specification and generate the translated output in HINTS's syntax.

Both the lexical and syntax analyser are created from specifications that encode the structure of the source and target languages, taking into account the following structure:

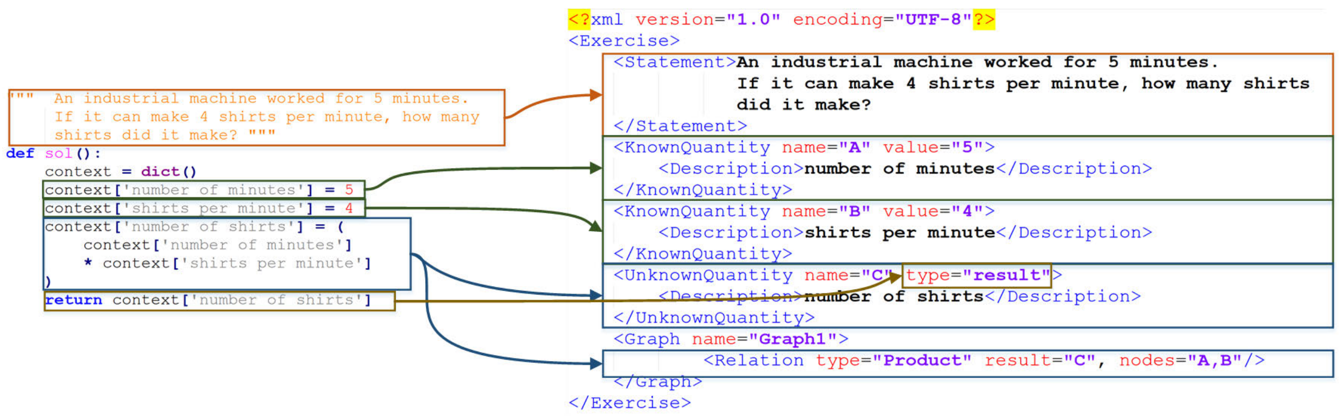
- The function starts with the heading *def sol():* and the first line is always *context=dict():*
- The last line of the function is a sentence with the syntax *return context['*name*']*, where name refers to the quantity that represents the problem's answer.
- All other lines in the source code generated define a quantity each. There are two syntactically different forms to do this. When the value is explicitly provided in the problem statement, the line follows the syntax *context['*name*']=*value, where name and value denote a quantity name and the value is explicitly given. When a quantity is defined in terms of other quantities, a mathematical expression is provided instead of a value, with each quantity expressed within a *context['*name*']* structure.

Our experimental evaluation showed that the source code generated by the proposed approach always conformed to the specified syntax. Therefore, the translation step proceeded with no errors in all cases. FIGURE 3 displays a translation example, showing both the input source code and the output XML file in the format required by HINTS.

IV. RESULTS

The performance of the proposed approach relies heavily on the performance of the code generation models that are available. As mentioned before, LLM trained on code generation are at their early stages but are rapidly progressing towards generating better and more human-like code. In this work, the maturity of these models is evaluated against a common benchmark dataset for MWP solving, i.e. SVAMP [23]. SVAMP is a collection of 1,000 MWPs, expressed in natural

[2]https://console.cloud.google.com/marketplace/details/github/github-repos



```
"""  An industrial machine worked for 5 minutes.
     If it can make 4 shirts per minute, how many
     shirts did it make? """
def sol():
    context = dict()
    context['number of minutes'] = 5
    context['shirts per minute'] = 4
    context['number of shirts'] = (
        context['number of minutes']
        * context['shirts per minute']
)
    return context['number of shirts']
```

```
<?xml version="1.0" encoding="UTF-8"?>
<Exercise>
    <Statement>An industrial machine worked for 5 minutes.
               If it can make 4 shirts per minute, how many shirts
               did it make?
    </Statement>
    <KnownQuantity name="A" value="5">
        <Description>number of minutes</Description>
    </KnownQuantity>
    <KnownQuantity name="B" value="4">
        <Description>shirts per minute</Description>
    </KnownQuantity>
    <UnknownQuantity name="C" type="result">
        <Description>number of shirts</Description>
    </UnknownQuantity>
    <Graph name="Graph1">
            <Relation type="Product" result="C", nodes="A,B"/>
    </Graph>
</Exercise>
```

(a) Example of input to the translator

(b) Example of output from the translator

FIGURE 3. Translation from generated source code to HINTS internal representation schema.

language (English), along with the numerical answer and an algebraic expression of the solution the problem.

The performance of the examined models is evaluated under two different scenarios: (i) Problem solving with unknown answer, in which we attempt to generate the graph for problems in SVAMP directly and uniquely from the problem statements, and (ii) problem solving with known answer, in which we attempt to generate and choose the correct graph from the problem statement, knowing the solution.

A. PROBLEM SOLVING WITH UNKNOWN RESULT

In order to evaluate the MWP solving performance, the following methodology was used: A random MWP was first selected from the SVAMP dataset and the corresponding `sol()` function was implemented manually, following the specifications outlined in Section III. This problem was then used as the example problem for all of the language models' prompts to ensure a fair evaluation across the different models. Subsequently, each model was prompted by providing the randomly chosen solved example problem, followed by the statement of the problem to be solved, as shown in FIGURE 2a. For each of the 1,000 problems in the SVAMP dataset, each model generated one potential solution, which was then evaluated by running the generated Python code and comparing the result to the expected solution. In order to carry out the inference process, the Softmax's temperature parameter (t) was fine-tuned to values 0.1, 0.3, and 0.5. This parameter is used to control the level of randomness of a model's predictions, with higher values resulting in a model becoming more random and less confident in its predictions, while lower values lead to more certain predictions. To ensure a fair evaluation, each parameter number variant of CodeGen and InCoder was tested three times, with the aforementioned temperature values used in each run.

Results shown in TABLE 1 and FIGURE 4 indicate that out of all the examined models, the 16B parameter CodeGen model achieved the best performance on the SVAMP dataset, reaching an accuracy of 39.1%, which was higher than the accuracy of all other models that were tested. One important observation that can be made from and FIGURE 4 and TABLE 1 is that there is a direct correlation between the number of parameters a model has and its performance. In fact, the models with more parameters achieved higher accuracy in most cases, as demonstrated by a Pearson's ρ value of 0.77.

TABLE 1. Performance on the SVAMP dataset in terms of accuracy for the best performing temperature (t) for each model.

Model	Temperature	Accuracy
InCoder-1B	0.1	0.061
InCoder-6B	0.3	0.174
CodeGen-350M	0.1	0.088
CodeGen-2B	0.1	0.272
CodeGen-6B	0.3	0.335
CodeGen-16B	0.3	**0.391**

Moreover, this relationship between model parameters and accuracy is also clearly displayed in FIGURE 4 that shows the accuracy of the InCoder and CodeGen models, respectively, for different temperature values. This figure reveals that the accuracy of a model tends to increase as the number of parameters in the model increases. However, for the CodeGen model, the relationship between the number of parameters and the achieved accuracy is not linear, but rather logarithmic. This means that while increasing the number of parameters can lead to an improvement in accuracy, the rate of improvement decreases as the number of parameters increases. Similar conclusions cannot be extracted for the InCoder model due to insufficient data points.

Finally, FIGURE 4 also reveals that the temperature parameter had minimal effects on the achieved accuracy. Overall, results suggest that the number of parameters in a model is a crucial factor in determining its performance, and increasing the number of parameters can lead to an improvement in accuracy, albeit at a diminishing rate.

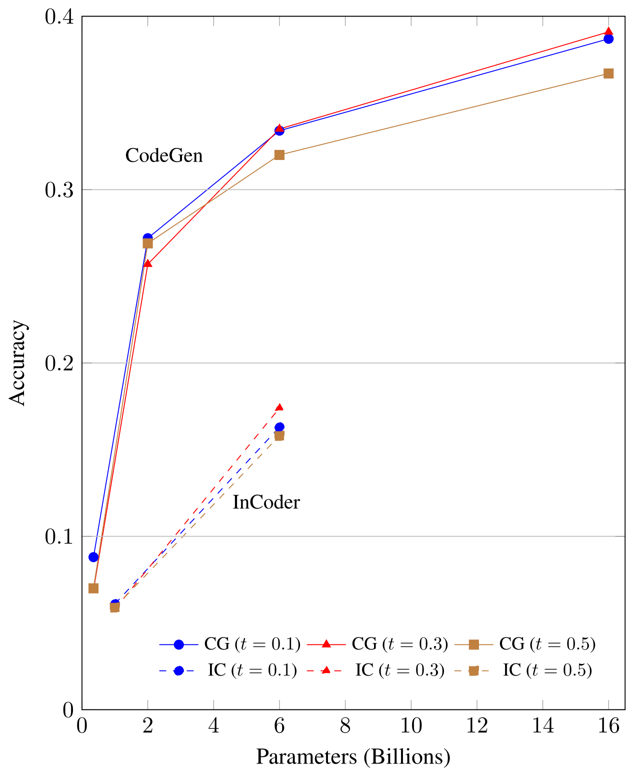


FIGURE 4. Accuracy obtained for different temperatures (t) and number of parameters for each model. CG: CodeGen, IC: InCoder.

B. PROBLEM SOLVING WITH KNOWN RESULT

The second examined scenario relies on the correct generation of the graph in a setting in which the student has both the problem statement and the solution to the problem. This scenario is common in a learning context, since the main objective is that the student is capable of actually reasoning through the problem, rather than simply giving a numerical answer. Many textbooks include the problem solution together with the statement, or in a separate section, to facilitate the self-correction from students. In this scenario, since the response is known, we can generate k samples and obtain the graph by selecting the Python script that returns a result matching the provided solution. This scenario was validated similar to the previous one. A random problem was picked and manually solved using Python (the same problem as in the previous evaluation) and the model was asked to provide the source-code solution to each of the problems in the SVAMP dataset. The difference is that in this case, the model was configured to produce k different alternative solutions to the problem.

TABLE 2 shows the percentage of problems that were correctly solved by generating the k different solutions. Large models, especially those with more than 2 Billion parameters, are particularly costly to run, therefore, this experiment was limited to exploring the top $k = 10$ samples. The best performance was achieved by generating $k = 10$ samples using the Codegen-6B model for a temperature $t = 0.5$,

TABLE 2. Performance of the examined models in terms of accuracy for different values of k.

Model	# Params	Temperature	$k = 1$	$k = 2$	$k = 5$	$k = 10$
InCoder	1B	0.1	0.059	0.070	0.086	0.100
		0.3	0.057	0.082	0.121	0.152
		0.5	0.057	0.088	0.139	0.183
	6B	0.1	0.171	0.208	0.253	0.284
		0.3	0.175	0.249	0.354	0.433
		0.5	0.165	0.256	0.396	0.502
CodeGen	350M	0.1	0.085	0.107	0.134	0.154
		0.3	0.074	0.114	0.182	0.244
		0.5	0.066	0.112	0.200	0.283
	2B	0.1	0.267	0.317	0.374	0.411
		0.3	0.265	0.361	0.484	0.571
		0.5	0.259	0.377	0.539	0.648
	6B	0.1	0.335	0.380	0.432	0.467
		0.3	0.336	0.431	0.546	0.623
		0.5	0.322	0.443	0.592	**0.692**
	16B	0.1	0.387	0.433	0.488	0.523
		0.3	0.391	0.482	0.587	0.654
		0.5	0.367	0.487	0.612	0.691

Note: M: Million, B: Billion. Best performance in bold.

leading to correctly solving 69.2% of the problems in the SVAMP dataset, whereas the second best performance was marginally worse, reaching an accuracy of 69.1% using the Codegen-16B model with $k = 10$ and $t = 0.5$.

The study of the effect of the temperature parameter on the performance of each model is presented in FIGURE 5. The results show that generating multiple samples with higher temperatures dramatically improves the accuracy. This trend is visible in all the examined models, being consistent with similar findings in the field of code synthesis [5]. In the case of the best performing Codegen-6B and Codegen-16B models, FIGURE 5 indicates that, with a temperature setting of 0.5 and generating 10 potential solutions for each problem, we ensure that at least one of these solutions will successfully solve the problem for approximately 70% of the evaluated problems. Therefore, the results suggest that generating multiple samples with higher temperatures can effectively improve the accuracy of the model's performance.

TABLE 2 clearly illustrates that higher temperature settings lead to the generation of better problem resolution graphs, especially if the solution is already known. Consequently, we can calculate the minimum number of samples that need to be generated in order to produce a solution that returns a correct result. This analysis is reflected in FIGURE 6 which depicts a cumulative histogram that shows which percentage of problems were actually solved by generating k samples. The histogram shows that despite having an initially lower accuracy, inference with a higher-temperature setting consistently performs better than the lower temperature counterparts. This result indicates that higher temperature settings can help generate more diverse samples, which can ultimately lead to better solutions for a given problem.

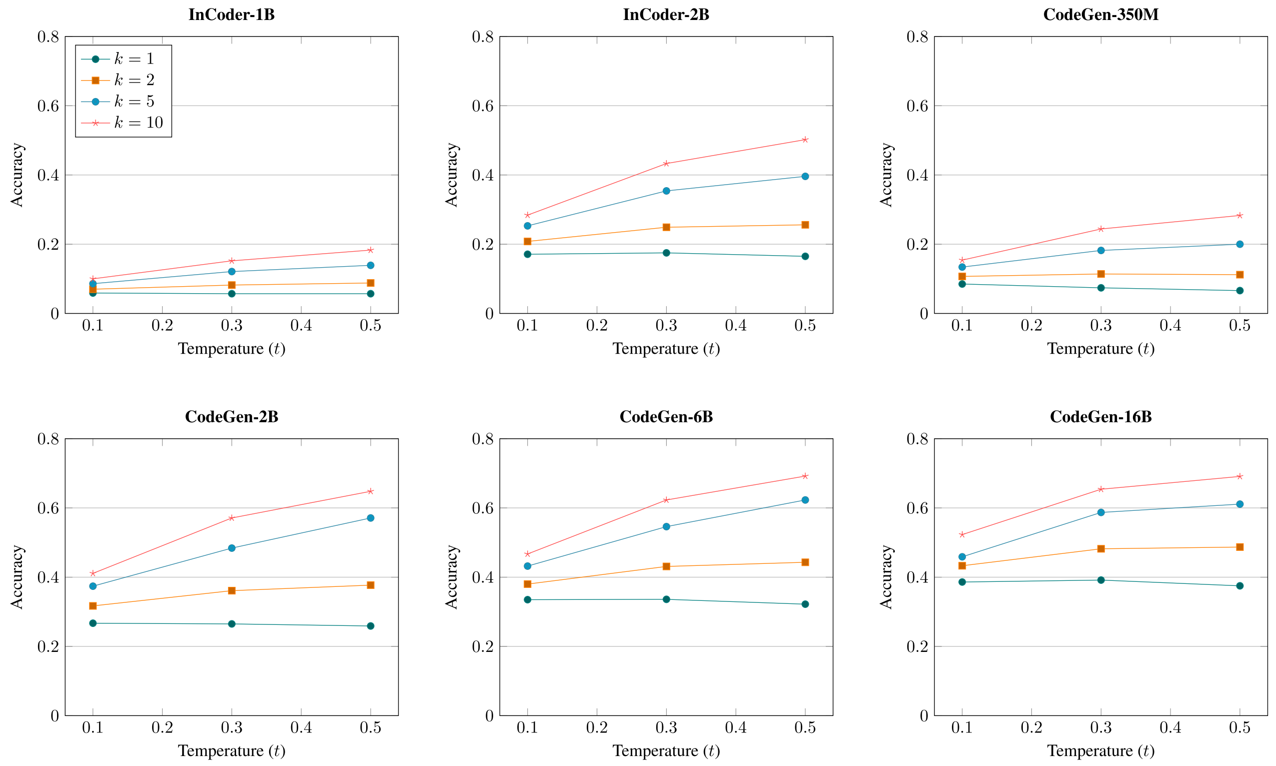


FIGURE 5. Accuracy vs. temperature (*t*) for different values of *k* for the examined models.

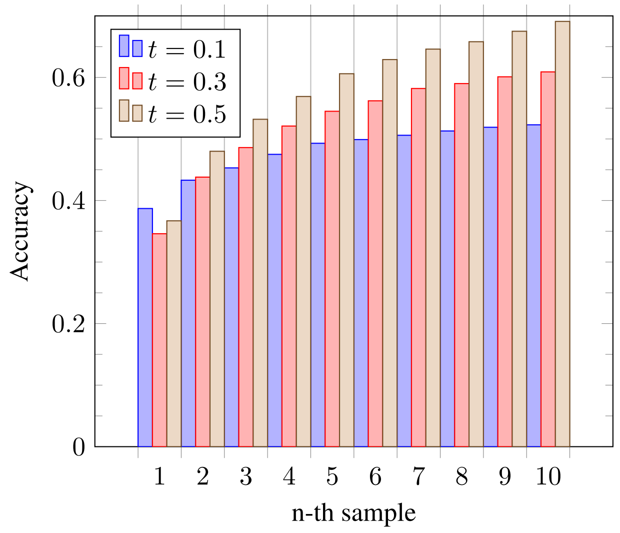


FIGURE 6. Histogram displaying the least amount of samples for solving a problem.

C. COMPARISON TO STATE OF THE ART

The results presented in this study directly compete with the latest state-of-the-art methods for solving MWPs. Our approach to solving problems with a known answer demonstrated a maximum accuracy of 69.5%, outperforming state-of-the-art methods, as shown in TABLE 3, even though it was not explicitly designed for MWP solving. In the case

TABLE 3. MWP solving performance of the proposed approach and of state-of-the-art methods on the SVAMP dataset.

Model	Accuracy
Codegen-6B Known Result (Ours)	**0.692**
DeductReasoner [13]	0.473
Roberta-Graph2Tree [23]	0.438
Roberta-GTS [23]	0.410
Codegen-16B Unknown Result (Ours)	0.391
Graph2Tree [36]	0.365
BERT-Tree [18]	0.324
GTS [31]	0.308
Roberta-Roberta [15]	0.303
BERT-BERT [15]	0.248
GroupAttn [16]	0.215

of solving problems with unknown answer, our Codegen-16B-based approach achieved a best accuracy of 39.1%, performing better than six out of the nine state-of-the-art methods. In comparison to other available MWP solving methods, our proposed approach enables the automation of translating MWPs from natural language to the internal representation of ITS like HINTS, addressing a major limitation of these systems. Therefore, other state-of-the-art MWP solving methods cannot replace our proposed approach for the task at hand since they cannot provide the required source code representation of the MWP. As language models continue to grow in the number of parameters, we expect our proposed approach to surpass existing methods without requiring specific domain knowledge. Overall, our study shows promising results in the development of efficient methods for MWP solving, paving the way for future research in this area.

V. CONCLUSION

A. SUMMARY AND FINDINGS

This study presents a novel method for solving mathematical word problems (MWPs) and converting them to the internal representation of Intelligent Tutoring Systems (ITS). The proposed method exploits Large Language Models (LLM) to generate Python source code that can solve the problem. One of the major advantages of this approach is that it allows for the automatic naming of the quantities that appear in the MWP. This capability is particularly useful for providing a conversational interaction between the student and the system. In addition to automated problem-solving and automatic naming, the proposed approach has the potential to enable the automated translation of problem statements into the ITS's internal knowledge representation schema. This capability would allow learners to add new MWP to an ITS, enabling them to practice solving a wide variety of math problems, while also enabling tutors to add new MWP at scale. Experimental results show that the proposed approach achieved high accuracy in solving MWPs with known and unknown solutions. Specifically, Salesforce's CodeGen model with 16B parameters achieved the highest accuracy, reaching a 39.1% accuracy for the case of unknown solution and 69.1% accuracy for the known solution scenario.

This work presents a promising new approach for solving MWPs that has the potential to significantly improve math education via online learning environments. By automating the problem-solving process, providing automatic naming of quantities, and enabling the translation of problem statements into an ITS's internal knowledge representation schema, this approach can help learners to develop their math skills more efficiently and effectively, while also enabling tutors to provide a more personalised and scalable learning experience.

The effectiveness of the suggested method heavily depends on the performance of the code generation models. This is demonstrated by the substantial performance disparities between the CodeGen and InCoder models studied. Nonetheless, it is worth noting that there is a noticeable link between a model's size and its performance, indicating that the quality of the outputted solutions could potentially improve with the emergence of more sophisticated models with a greater number of parameters in the future.

It is interesting to mention the trade-off that exists between the size of the model, the number of samples generated, and the performance of the method. While larger models tend to perform better, it is possible and probably desired to use a smaller model in order to reduce the computational complexity in favour of being able to generate a larger amount of samples. However, integration of such a system into an ITS is still highly experimental, as we understand that while 69% accuracy is not enough to make it to a final product, it is however sufficient to ease the encoding of MWPs by producing a simple-to-debug intermediate representation and providing sufficient performance if human supervision is present.

B. FINAL CONSIDERATIONS

The suggested approach, however, is not without its limitations. Generating plain Python code limits the model's output to arithmetic solutions. Therefore, the approach is not able to solve problems that require the use of algebraic equation systems. Moreover, it produces just one solution graph for each problem, which although frequently the most apparent solution, may not always be the sole option. To address these drawbacks, future work will aim to develop methods for handling these issues. Furthermore, it would be interesting to explore the various different created source code snippets to extract all potential resolution paths to a particular problem. Finally, the end goal of this research is to be integrated into ITS so that new problems can be tutored without having been pre-registered on the system. Future work will propose an integration of the method and an evaluation of the students' Quality of Experience and acceptance of the tutoring system.

ACKNOWLEDGMENT

Artificial Intelligent agents, such as ChatGPT[3] or Grammarly[4] have been used for improving the quality of written English language and grammatical error corrections.

REFERENCES

[1] D. Arnau, M. Arevalillo-Herráez, and J. A. González-Calero, "Emulating human supervision in an intelligent tutoring system for arithmetical problem solving," *IEEE Trans. Learn. Technol.*, vol. 7, no. 2, pp. 155–164, Apr. 2014, doi: 10.1109/TLT.2014.2307306.

[2] D. Arnau, M. Arevalillo-Herráez, L. Puig, and J. A. González-Calero, "Fundamentals of the design and the operation of an intelligent tutoring system for the learning of the arithmetical and algebraic way of solving word problems," *Comput. Educ.*, vol. 63, pp. 119–130, Apr. 2013.

[3] D. G. Bobrow, "Natural language input for a computer problem solving system," Massachusetts Inst. Technol., Cambridge, MA, USA, Tech. Rep., AIM-066, 1964.

[4] T. B. Brown et al., "Language models are few-shot learners," in *Advances in Neural Information Processing Systems*, vol. 33. Red Hook, NY, USA: Curran Associates, 2020, pp. 1877–1901.

[5] M. Chen et al., "Evaluating large language models trained on code," 2021, *arXiv:2107.03374*.

[6] T.-R. Chiang and Y.-N. Chen, "Semantically-aligned equation generation for solving and reasoning math word problems," in *Proc. Conf. North*, 2019, pp. 2656–2668.

[7] A. Chowdhery et al., "PaLM: Scaling language modeling with pathways," 2022, *arXiv:2204.02311*.

[8] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proc. Conf. North*, 2019, pp. 4171–4186, doi: 10.18653/v1/N19-1423.

[9] D. Fried, A. Aghajanyan, J. Lin, S. Wang, E. Wallace, F. Shi, R. Zhong, W.-T. Yih, L. Zettlemoyer, and M. Lewis, "InCoder: A generative model for code infilling and synthesis," 2022, *arXiv:2204.05999*.

[3]https://chat.openai.com
[4]https://grammarly.com

[10] L. Gao, S. Biderman, S. Black, L. Golding, T. Hoppe, C. Foster, J. Phang, H. He, A. Thite, N. Nabeshima, S. Presser, and C. Leahy, ''The pile: An 800GB dataset of diverse text for language modeling,'' 2021, *arXiv:2101.00027*.

[11] D. Goldwasser and D. Roth, ''Learning from natural instructions,'' *Mach. Learn.*, vol. 94, pp. 205–232, Sep. 2013. [Online]. Available: https://www.ijcai.org/Proceedings/11/Papers/301.pdf

[12] D. Huang, J. Liu, C. Y. Lin, and J. Yin, ''Neural math word problem solver with reinforcement learning,'' in *Proc. 27th Int. Conf. Comput. Linguistics*, Santa Fe, NM, USA, Aug. 2018, pp. 213–223.

[13] Z. Jie, J. Li, and W. Lu, ''Learning to reason deductively: Math word problem solving as complex relation extraction,'' in *Proc. 60th Annu. Meeting Assoc. Comput. Linguistics*, 2022, pp. 5944–5955.

[14] W. Kintsch and J. G. Greeno, ''Understanding and solving word arithmetic problems.,'' *Psychol. Rev.*, vol. 92, no. 1, pp. 109–129, 1985.

[15] Y. Lan, L. Wang, Q. Zhang, Y. Lan, B. T. Dai, Y. Wang, D. Zhang, and E. P. Lim, ''MWPToolkit: An open-source framework for deep learning-based math word problem solvers,'' in *Proc. AAAI Conf. Artif. Intell.*, vol. 36, Jun. 2022, pp. 13188–13190, doi: 10.1609/aaai.v36i11.21723.

[16] J. Li, L. Wang, J. Zhang, Y. Wang, B. T. Dai, and D. Zhang, ''Modeling intra-relation in math word problems with different functional multi-head attentions,'' in *Proc. 57th Annu. Meeting Assoc. Comput. Linguistics*, 2019, pp. 6162–6167, doi: 10.18653/v1/P19-1619.

[17] S. Li, L. Wu, S. Feng, F. Xu, F. Xu, and S. Zhong, ''Graph-to-tree neural networks for learning structured input-output translation with applications to semantic parsing and math word problem,'' in *Proc. Findings Assoc. Comput. Linguistics*, 2020, pp. 2841–2852, doi: 10.18653/v1/2020.findings-emnlp.255.

[18] Z. Li, W. Zhang, C. Yan, Q. Zhou, C. Li, H. Liu, and Y. Cao, ''Seeking patterns, not just memorizing procedures: Contrastive learning for solving math word problems,'' 2021, *arXiv:2110.08464*.

[19] Z. Liang, J. Zhang, L. Wang, W. Qin, Y. Lan, J. Shao, and X. Zhang, ''MWP-BERT: Numeracy-augmented pre-training for math word problem solving,'' in *Proc. Findings Assoc. Comput. Linguistics, NAACL*, 2022, pp. 997–1009.

[20] C. Liguda and T. Pfeiffer, ''Modeling math word problems with augmented semantic networks,'' in *Natural Language Processing and Information Systems*, G. Bouma, A. Ittoo, A. Métais, E. H. Wortmann, Eds. Berlin, Germany: Springer, 2012, pp. 247–252.

[21] E. Nijkamp, B. Pang, H. Hayashi, L. Tu, H. Wang, Y. Zhou, S. Savarese, and C. Xiong, ''CodeGen: An open large language model for code with multi-turn program synthesis,'' 2022, *arXiv:2203.13474*.

[22] F. Ortin, J. Quiroga, O. Rodriguez-Prieto, and M. Garcia, ''Evaluation of the use of different parser generators in a compiler construction course,'' in *Information Systems and Technologies*, vol. 3. Cham, Switzerland: Springer, 2022, pp. 338–346.

[23] A. Patel, S. Bhattamishra, and N. Goyal, ''Are NLP models really able to solve simple math word problems?'' in *Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics, Hum. Lang. Technol.*, 2021, pp. 2080–2094.

[24] S. Roy and D. Roth, ''Mapping to declarative knowledge for word problem solving,'' *Trans. Assoc. Comput. Linguistics*, vol. 6, pp. 159–172, Dec. 2018.

[25] Y. Shen and C. Jin, ''Solving math word problems with multi-encoders and multi-decoders,'' in *Proc. 28th Int. Conf. Comput. Linguistics*, 2020, pp. 2924–2934, doi: 10.18653/v1/2020.coling-main.262.

[26] J. R. Slagle, ''Experiments with a deductive question-answering program,'' *Commun. ACM*, vol. 8, no. 12, pp. 792–798, Dec. 1965, doi: 10.1145/365691.365960.

[27] L. Wang, Y. Wang, D. Cai, D. Zhang, and X. Liu, ''Translating a math word problem to a expression tree,'' in *Proc. Conf. Empirical Methods Natural Lang. Process.*, 2018, pp. 1064–1069.

[28] L. Wang, D. Zhang, L. Gao, J. Song, L. Guo, and H. T. Shen, ''MathDQN: Solving arithmetic word problems via deep reinforcement learning,'' in *Proc. AAAI Conf. Artif. Intell.*, vol. 32, 2018, pp. 1–12.

[29] L. Wang, D. Zhang, J. Zhang, X. Xu, L. Gao, B. T. Dai, and H. T. Shen, ''Template-based math word problem solvers with recursive neural networks,'' in *Proc. AAAI Conf. Artif. Intell.*, vol. 33, 2019, pp. 7144–7151, doi: 10.1609/aaai.v33i01.33017144.

[30] Q. Wu, Q. Zhang, and Z. Wei, ''An edge-enhanced hierarchical graph-to-tree network for math word problem solving,'' in *Proc. Findings Assoc. Comput. Linguistics, EMNLP*, 2021, pp. 1473–1482, doi: 10.18653/v1/2021.findings-emnlp.127.

[31] Z. Xie and S. Sun, ''A goal-driven tree-structured neural model for math word problems,'' in *Proc. 28th Int. Joint Conf. Artif. Intell.*, Aug. 2019, pp. 5299–5305.

[32] F. F. Xu, U. Alon, G. Neubig, and V. J. Hellendoorn, ''A systematic evaluation of large language models of code,'' in *Proc. 6th ACM SIGPLAN Int. Symp. Mach. Program.*, Jun. 2022, pp. 1–10.

[33] W. Yu, Y. Wen, F. Zheng, and N. Xiao, ''Improving math word problems with pre-trained knowledge and hierarchical reasoning,'' in *Proc. Conf. Empirical Methods Natural Lang. Process.*, 2021, pp. 3384–3394, doi: 10.18653/v1/2021.emnlp-main.272.

[34] D. Zhang, L. Wang, L. Zhang, B. T. Dai, and H. T. Shen, ''The gap of semantic parsing: A survey on automatic math word problem solvers,'' *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 42, no. 9, pp. 2287–2305, Sep. 2020, doi: 10.1109/TPAMI.2019.2914054.

[35] J. Zhang, R. K.-W. Lee, E.-P. Lim, W. Qin, L. Wang, J. Shao, and Q. Sun, ''Teacher-student networks with multiple decoders for solving math word problem,'' in *Proc. 29th Int. Joint Conf. Artif. Intell.*, Jul. 2020, pp. 4011–4017.

[36] J. Zhang, L. Wang, R. K.-W. Lee, Y. Bin, Y. Wang, J. Shao, and E.-P. Lim, ''Graph-to-tree learning for solving math word problems,'' in *Proc. 58th Annu. Meeting Assoc. Comput. Linguistics*, 2020, pp. 3928–3937.

PABLO ARNAU-GONZÁLEZ received the degree in computer engineering from Universitat de Valéncia (UV), in 2015, and the Ph.D. degree from the University of the West of Scotland (UWS) under the supervision of Prof. Naeem Ramzan and Miguel Arevalillo-HerrÁez. He is currently a Postdoctoral Research Fellow with UV. He has participated in three national research projects. He has authored and coauthored over 15 research publications, including peer-reviewed journals, book chapters, and conference proceedings. His research interests include intelligent tutoring systems, natural language processing, and applied machine learning.

ANA SERRANO-MAMOLAR received the B.Sc. degree (Hons.) in telecommunications from the University of Valladolid, Spain, in 2006, the M.Sc. degree in artificial intelligence and pattern recognition from the Polytechnic University of Valencia (UPV), Spain, in 2012, and the Ph.D. degree from the University of the West of Scotland (UWS), in 2019. She is currently an Assistant Professor and a Postdoctoral Research Fellow with Universidad de Burgos, Spain. She has alternated academic career with private industry over years and is back to academia. She has participated international and international research projects. She has authored or coauthored over 14 research publications, including peer-reviewed journals and conference proceedings. Her research interests include affective computing, applied machine learning, and adaptive systems.

STAMOS KATSIGIANNIS (Member, IEEE) received the B.Sc. degree (Hons.) in informatics and telecommunications from the National and Kapodistrian University of Athens, Greece, in 2009, the M.Sc. degree in computer science from the Athens University of Economics and Business, Greece, in 2011, and the Ph.D. degree in computer science (biomedical image and general-purpose video processing) from the National and Kapodistrian University of Athens, in 2016. He is currently an Assistant Professor with the Department of Computer Science, Durham University, U.K. He has participated in seven national and international research projects. He has authored or coauthored over 60 research publications, including peer-reviewed journals, book chapters, and conference proceedings. His research interests include machine learning, natural language processing, affective computing, image analysis, and image and video quality.

TURKE ALTHOBAITI received the B.Sc. degree in computer science from Taif University, Saudi Arabia, in 2009, the M.Sc. degree in computer science from Ball State University, USA, in 2014, and the Ph.D. degree in computer science from the University of the West of Scotland, U.K., in 2019. He is currently an Assistant Professor with Northern Border University, Saudi Arabia. His research interests include affective computing and machine learning, wearable and flexible sensors, and the Internet of Things.

MIGUEL AREVALILLO-HERRÁEZ received the degree in computing from the Technical University of Valencia, Spain, in 1993, and the B.Sc. degree (Hons.) in computing, the P.G.Cert. degree in teaching and learning in higher education, and the Ph.D. degree from Liverpool John Moores University, U.K., in 1997 and 1997, respectively. He was a Postdoctoral Research Fellow and a Senior Lecturer with Liverpool John Moores University, until 1999. Then, he left to work in private industry for one year and came back to academia, in 2000. He was a Program Leader of the computing and business degrees with the Mediterranean University of Science and Technology, until 2006. Since 2006, he has been a Full Professor of computer science and artificial intelligence with Universitat de Valéncia.