Lecture Notes in Networks and Systems 253

Juan M. Corchado
Saber Trabelsi *Editors*

# Sustainable Smart Cities and Territories

Springer

# Lecture Notes in Networks and Systems

Volume 253

Juan M. Corchado · Saber Trabelsi
Editors

# Sustainable Smart Cities and Territories

Springer

*Editors*
Juan M. Corchado
Department of Computing Science
Universidad Salamanca
Salamanca, Spain

Saber Trabelsi
Texas A&M University at Qatar
Doha, Qatar

ISSN 2367-3370 ISSN 2367-3389 (electronic)
Lecture Notes in Networks and Systems
ISBN 978-3-030-78900-8 ISBN 978-3-030-78901-5 (eBook)
https://doi.org/10.1007/978-3-030-78901-5

# Modern Integrated Development Environment (IDEs)

Zakieh Alizadehsani[1](✉), Enrique Goyenechea Gomez[1], Hadi Ghaemi[2], Sara Rodríguez González[1], Jaume Jordan[3], Alberto Fernández[4], and Belén Pérez-Lancho[5]

[1] BISITE Research Group, University of Salamanca, Salamanca, Spain
{zakieh,egoyene,srg}@usal.es

[2] Computer Engineering Department, Ferdowsi University of Mashhad, Mashhad, Iran
hadi.qaemi@um.ac.ir

[3] Valencian Research Institute for Artificial Intelligence (VRAIN), Universitat Politècnica de València, Valencia, Spain
jjordan@dsic.upv.es

[4] Universidad Rey Juan Carlos, Madrid, Spain
alberto.fernandez@urjc.es

[5] Department of Computer Science and Automation, University of Salamanca, Salamanca, Spain
lancho@usal.es

**Abstract.** One of the important objectives of smart cities is to provide electronic services to citizens, however, this requires the building of related software which is a time-consuming process. In this regard, smart city infrastructures require development tools that can help accelerate and facilitate software development (mobile, IoT, and web applications). Integrated Development Environments (IDEs) are well-known tools that have brought together the features of various tools within one package. Modern IDEs include the advantages of Artificial Intelligence (AI) and Cloud Computing. These technologies can help the developer overcome the complexities associated with multi-platform software products. This paper has explored AI techniques that are applied in IDEs. To this end, the Eclipse Theia (cloud-based IDE) and its AI-based extensions are explored as a case study. The findings show that recommender system models, language modeling, deep learning models, code mining, and attention mechanisms are used frequently to facilitate programming. Furthermore, some researches have used NLP techniques and AI-based virtual assistance to promote the interaction between developers and projects.

**Keywords:** Integrated Development Environment (IDE) · Online IDEs · Software development · Artificial intelligence (AI) · Theia

## 1 Introduction

Today, cities have a strong desire to make their infrastructure smarter, which requires electronic infrastructure and developing the related software.

https://doi.org/10.1007/978-3-030-78901-5_24

Unfortunately, software development is a time-consuming process. To enable cities to create electronic services faster, several tools have emerged which accelerate and facilitate the software development life cycle. Integrated Development Environments (IDEs) are well-known tools that have brought together the features of various tools within one package. The main goal of IDEs is to increase development speed, reduce errors, and increase the accuracy of the programming process [1,2]. According to the literature, there are many functionalities (features) such as debugging, autocomplete, etc. These features can fall into two categories, 1) Features that are usually running continuously in the background most of which can verify source code state such as live syntax checker and facilitate developments, including code autocomplete during program writing time. 2) Features that users can apply arbitrarily such as control versioning, code search, etc. However, IDEs could have more automation and intelligence to help developers. These features can be obtained by using Artificial Intelligence (AI) and Machine Learning techniques. Most IDEs include several tools to cover most aspects of software development like analyzing, designing, implementing, testing, documenting, and maintaining [3]. To increase the intelligence, these IDEs have embedded training models into the modern versions. This task can fall into two methods:

- Improving current functionalities (features)
- Adding new functionalities.

In this regard, Eclipse Theia [4] which is a cloud-based IDE, includes several AI-based extensions which have been explored in the case study. The findings of the present study can help researchers use this paper to identify popular IDE functionalities and related research areas, for example, the auto-complete functionality benefits from language modeling [5] and NLP techniques. Moreover, popular desktop and popular cloud IDEs have been investigated which can give knowledge about practical aspects.

The structure of the paper is as follows. Section 2 gives an overview of the available IDE functionalities. Section 3 reviews the related works. Section 4 conducts AI-based IDE functionalities. Section 5 describes the case study conducted with Theia Cloud IDE, and finally, Sect. 6 concludes the paper.

## 2 Background: IDE Functionality

Although programmers can develop without the IDEs and use simple text editors, IDEs are a set of tools that help programmers significantly. Syntax highlighting, debugging and editing features, which run in background, are the most common and basic features of IDEs [6]. Also, popular IDEs have provided advanced functionalities such as version controlling, terminal console, program element suggestion. These functionalities can be significantly improved by AI algorithms. Moreover, with emerging cloud-based services and the requirement of multiple development environments for different applications, IDEs are moving from desktop-based to cloud-based which are accessible through a web browser such as AWS Cloud9 [7], Codeanywhere [8] Eclipse Che [9].

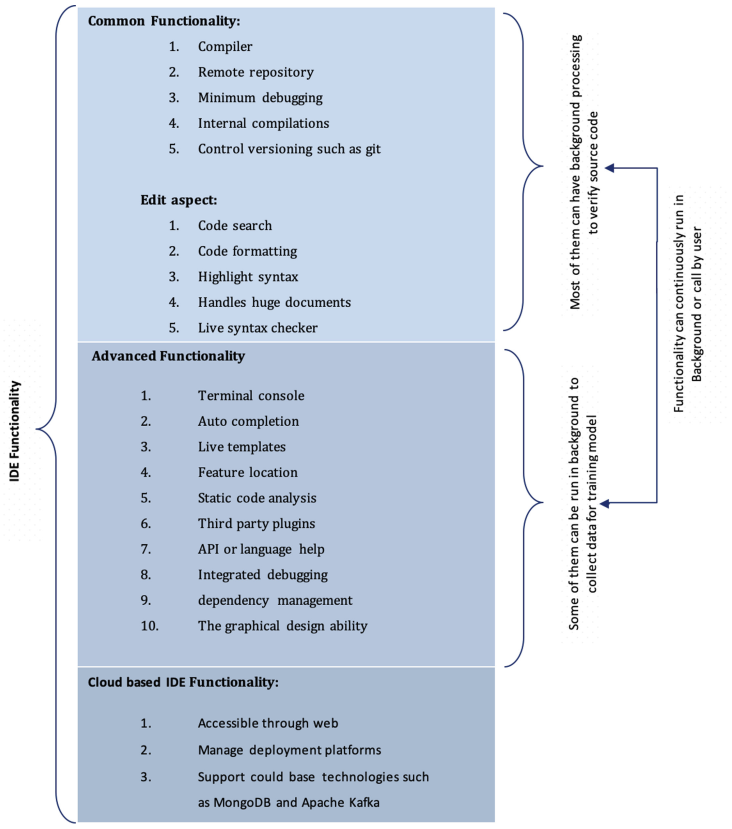
IDE Functionality
Common Functionality:
1. Compiler
2. Remote repository
3. Minimum debugging
4. Internal compilations
5. Control versioning such as git
Edit aspect:
1. Code search
2. Code formatting
3. Highlight syntax
4. Handles huge documents
5. Live syntax checker
Advanced Functionality
1. Terminal console
2. Auto completion
3. Live templates
4. Feature location
5. Static code analysis
6. Third party plugins
7. API or language help
8. Integrated debugging
9. dependency management
10. The graphical design ability
Cloud based IDE Functionality:
1. Accessible through web
2. Manage deployment platforms
3. Support could base technologies such as MongoDB and Apache Kafka
Most of them can have background processing to verify source code
Some of them can be run in background to collect data for training model
Functionality can continuously run in Background or call by user

**Fig. 1.** Classified popular IDE features

Regarding related functionality topics, in the current paper, IDE functionalities are categorized within the three classifications 1) basic and common 2) advanced 3) cloud based. The details are illustrated in Fig. 1.

## 3 Related Works

Software development is a broad research area with different application domains in mobile, web, multimedia, IoT, etc. Thus, many studies have been conducted in the different fields of software development, which are mainly related to the stages of designing (identify requirements and dependencies), developing (implement, compile, run, test, debug), and optimizing (code reviews, integration).

IDEs combine tools that facilitate the development process. Several studies have investigated IDE technologies and their future [6,10,11]. Although secondary studies on IDE tools and identifying all related topics have received little attention, [12] has presented a valuable study that has investigated the role and applications of AI in classical software engineering. Their survey can serve as an informative guideline for researchers to build intelligent IDE tools [13].

Moreover, given the notable role of cloud technologies in modern IDEs, [14] has researched cloud-based IDEs. Also, [15] is a valuable work on Cloud IDE creation, which includes the required environment and reviewing the challenges posed to building cloud IDEs. From a more general perspective, [16] have investigated the effect of AI in software development and the role of learning from available codes. These codes can be collected from online code repositories or local code projects for IDEs. The methods and learning process from code have been investigated in depth in [17], which can be of help to researchers when designing intelligent IDE tools.

Besides, recent studies in the field of IDEs have been conducted to create a more appropriate and efficient interaction between the programmer and the programming environment, making the software production process more rapid. For example, some works used virtual assistants. [18] have trained the model which identified speech action types in developer question/answer conversations while bug repair.

The current study tries to give a big picture of popular AI approaches in desktop and online IDEs which have considered more practical aspects.

## 4 AI-Based IDE Functionality

Merging artificial intelligence with existing IDE functionality can bring new opportunities in most involved area in software development tools. This improvement can fall into two methods: 1) Improving current features 2) Adding new features.

### 4.1 Improve Current Functionalities

Existing features, such as code suggestion or code search, usually resort to recommender systems so as to provide more accurate results to developers [16].

**Compiler:** Compilers turn a programming language into low-level machine languages which can hide complexity from the developer and also help execute written code on different platforms [19]. This feature is essential and must be included as a basic IDE feature. However, it can be added as a third-party extension as well. Instruction selection, translation validation, and code optimization are some implementation issues in compiler construction. Recently, some efforts applied machine learning [20] or deepening [21] for code optimization.

**Code Completion:** In traditional IDEs, code completion can suggest lists of programming language function during program writing. Improving this functionality through AI can be divided into two groups 1) API/function/Class/variable suggestion improved by recommender systems 2) Automatic programming based on language modeling [5]. The first category can improve the accuracy of suggestions and provide an efficient list of API/function/variable suggestions [22].

The second group is for the code completion approach in which AI can be used, such as NLP techniques for language modeling [23]. These suggestions can consider local code features in the current IDE file. In other words, when developers are writing the code, IDEs automatically collect the data in the background and provide the data needed to train the models. These approaches usually use Deep Neural Networks (DNN) that need to learn from large amounts of code. In practical works, some efforts applied this solution in one-line code generation. For example, TabNine [24] has sped up programming by offering APIs. TabNine is a tool for code completion suggestions, trained on millions of open-source Java applications. This plugin works on the basis of the local code on which the developer is working.

**Debugging:** Debugging software is a process for the detection, location and fixing of the bugs during software testing which takes more than 50% of a programmer's time [25]. According to software development literature, automation has been used to speed up the software development process. Fully automated debugging is still being investigated. Semi-automatic debugging, which requires human participation, has also been investigated by researchers. Existing methods can be divided into two general categories: 1) Generate-and-validate approaches, which first produce a set of candidate patches, and are then tested and validated by the test suite. 2) Synthesis-based approaches, collect information by running a test suite while using this information to create a problem solver [26].

Recently, one of the most widely used methods of debugging is the use of automatic translation methods. For example, in [27,28], the patch created for debugging is based on the use of neural machine translation. The method applied in these studies is based on sequence-to-sequence learning.

**Code Search:** In the code search area, a search query can be written in natural languages or structured text (E.g. code). Therefore, search code AI models need both search queries and related codes. Most of the studies apply NLP techniques to search code. The main problem with these methods is that they do not take into account the difference between text and code which is structured code. In this regard, [29] introduced the DNN model which has used code snippets and description. Moreover, using AI approaches in code search requires a large code dataset, which might lead to an increase in training and test time [30]. Therefore, some efforts have employed embedded neural networks to reduce the dimension of huge code snippets and to make similar codes close together [29].

**Automated Testing:** Testing is an important part of quality assurance in which to verify that software is bug-free. In software engineering, testing can be applied at different levels and with different techniques [31]. Ideally, the goal of an automated data generation system for testing is to generate experimental data in a way that enables all branches of the program with the least possible computational effort. Machine learning methods are one of the most widely used methods in this field. For example, in [32] a productive statistical learning approach for files is introduced.

Although, IDE is used as an environment which facilitates code writing, some of these IDEs can provide automatic testing as well. However, some tests are usually presented in separate tools due to the high complexity. For example, to find security vulnerabilities, different inputs are repeatedly tested and the inputs are modified, which is called the fuzzing process. There are three general methods for fuzzing. Blackbox random fuzzing [33], Whitebox constraint-based fuzzing [34], grammar-based fuzzing [35]. Also, abnormally detection have been used for this purpose, as well.

**Graphical Editor:** Interactive graphical editors allow even non-expert users to create software [36]. MDA presents levels of abstraction which enable Visual Programming Language (VPL) [37] in graphical editors. These visual items can be modules, services, etc. When the user defines the sequence of visual items, the code generator can transfer them into executable code. Recently some works have used computer vision, to improve visual programming. The main idea behind these approaches is generating text from an image. However, in the automatic code generation field, the text is code (structured text) instead of natural language text [38]. Convolutional neural network (CNN) and recurrent neural network (RNN) have been used in this type of studies which are well-known deep neural networks [38–40].

**Live Templates:** Static IDs' templates can include class, function template, expression (loop, switch cases). Predefined placeholders in templates, usually filled by developers. However, the prediction ability of AI has led to the emergence of live templates in modern IDEs. These models can predict variables based on collecting information from local code projects [41].

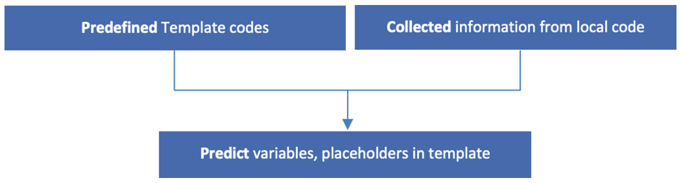
Predefined Template codes
Collected information from local code
Predict variables, placeholders in template

**Fig. 2.** Live template data

## 4.2 Adding New Functionalities

With the advent of online code repositories and improved data collection, it has become possible to add more new intelligent functionalities.

**IDE Functionality Recommendations:** [42] have found that users would rarely use all the functionalities of IDEs (on average they use just 42 commands out of the 1100 available commands). Therefore, some of the efforts [43,44] have presented new feature for offering IDE functionalities. These employ recommendation system models to make users aware of the existing functionalities. Some of theses functionalities have been outlined in this section.

**Code Summarizing:** The process of summarizing source code is to generate short descriptions in the natural language of written code. Code summarization can be considered as the basis of software documentation. The main idea of giving a short explanation to a programmer is to help them understand the code quickly and easily without having to read the code [45]. In general, source code memorization methods can be divided into two categories. 1) Pattern-based methods, 2) Artificial intelligence-based methods. For example, [46] has used pattern-based methods to create code explanations. The use of deep learning techniques, such as language modeling, has attracted the researchers' attention to this area as well [45,47].

**AI Virtual Assistant:** The AI-based virtual assistant is a program that can perform duties on the basis of commands or requests [48]. It identifies speech action types in developer question/answer conversations during bug repair. Google Assistant and Apple's Siri are popular assistants in the real-world. However, some works such as [18,49,50] have used virtual assistants in IDEs as well.

**Comment Generation:** Although comments are very useful, developers are not utilizing them enough in their coding, even if they add comments, they are not in the same style. Therefore, some research efforts have focused on generating descriptive comments for source code blocks [51]. Most of the academic works have focused on generating text description based on the source code functionality [52,53].

## 5 Case Study: Theia Cloud IDE

IDEs include several features, each of which is related to different disciplines, such as software development, user interaction, etc. Therefore, creating IDEs from scratch is troublesome. Under these circumstances, reusing other high-level frameworks can save time. These frameworks already provided the basic features, such as editing text (highlighting syntax, search, undo, find and replace), debugging, autocomplete for functions, block comment/uncomment [54]. As a result, using ready-made frameworks will facilitate the process for the developers, making it possible for them to invest more time in their custom IDEs. Table 1 shows the comparison between Eclipse Theia and Other IDEs.

There are different terms involved in Theia platform architectures. For example, the term "Theia" means Theia platform or Theia editor. Therefore, it is necessary to first define the terms used to refer to this technology.

**Table 1.** Comparison between Eclipse Theia and other IDEs

| Title | Open source | Desktop/online | Building IDE product |
|---|---|---|---|
| VS Code | Only desktop version | +/+ | − |
| Intelligent IDE | Freemium | +/+ | − |
| Codeanywhere | Freemium | +/+ | − |
| AWS Cloud9 | Freemium | +/+ | − |
| JSFiddle | Free | −/+ | − |
| Eclipse Theia | Free | +/+ | + |

- **Workspace** is a specific machine, acting as a container that holds the project files, package managers, and an IDE-Interface. The interesting thing about workplaces is that developers can deploy a project and its dependencies in a workplace, then provide an image of it. This new workplace can be used as a basis for other projects [15]. However, it should be regarded that as the number of workplaces is increasing, using an orchestration tool for managing workplaces is essential. This is where that workplace server plays an important role.
- **Workspace server** supports the management of developer workplaces. It can give a dashboard for creating, ending, starting and sharing workspaces. This dashboard is similar to the management of a virtual machine on cloud technologies [55].
- **Theia platform**, Theia is a platform and some developers utilize products based on Theia (Che Theia). However, Che Theia is an editor for Eclipse Che based on the Theia platform and it is called Theia as well [56].
- **Eclipse Che**, is a developer workspace server for the Eclipse Theia platform. [57].

After understanding the basic concepts, it is helpful to learn about the structure of the Theia platform. Theia is available as a desktop and web-based application. To handle both architectures with a single source, Theia uses the client-server architecture. Moreover, Theia can also reuse other high-level frameworks, technologies, protocols, etc. Theia benefits from several technologies, including:

- **Keycloak**: In fact, Che Theia is an editor for Eclipse Che. Therefore, it is not responsible for multi-developer management on the system level, via containers or OS user rights. These services are related to Eclipse Che, which manages developer workspaces. Eclipse Che utilizes Keycloak to authenticate developers [57].
- **Monaco**: Not only Eclipse Theia has followed most of VS code objects but it uses VS code abilities such as Monaco code editor (the editor of VS Code), as well. This reuse allows developers to use VS Code extensions in Theia [58].
- **TypeScript**: Considering that JavaScript cannot support complex applications; Typescript is used to enhance maintainability. In this regard, Theia UI is fully implemented through Typescript [59].

- **InversifyJS**: Theia uses the dependency injection framework Inversify.js to compose and configure the frontend and backend applications. InversifyJS is a lightweight inversion of control (IoC) container for TypeScript and JavaScript apps. This feature contributes to extensibility and the future growth of products [60].
- **Language server protocol (LSP)**: Theia has a distributed architecture that needs to communicate between the client and the server. LSP is designed for communicating between a tool (the client) and a language smartness provider (the server). This protocol allows to implement the autocomplete feature of IDEs in client-server structure [61,62].

All of the mentioned technologies convert Theia into a platform for building IDEs and developing tools. Theia not only has most of the basic code editor features - highlight syntax, debugging, etc., but also, has significant expendability. Theia has core and other extensions that help to build extensions. This is where deep learning models can be extended as extensions for desktop or cloud IDE products. Also, researchers can use the benefits of VS code extensions in the VS code marketplace which contains several AI-based extensions [63]. This study has conducted some traditional and AI-based extensions that can be used as developer assistants (Table 2). In general, practical tools has focused on providing features. Therefore, finding theoretical background which is used in their development is difficult. However, there are some works on IDE extensions that have provided valuable information:

**kite** [64,65] is an auto-complete tool that supports many back-end and front-end languages such as Python, Java, JavaScript, etc. The unique feature of this plugin is the completion of multi-line codes. Kite uses GPT-2 (utilizing deep learning) which is a prepared general-purpose model called strong AI. This kind of learner can be used for different tasks, such as text translation, answering questions, summarizing passages, etc. This model used 25 million open-source code files that can be run locally.

**Table 2.** Traditional and AI-based extensions example

| Title | Extension title | AI techniques | Ref |
|---|---|---|---|
| Kite | Auto-complete | DNN/GPT-2 | [65] |
| Flutter | Debugger | – | [66] |
| DeepL | Auto-complete | DNN | [67] |
| VSearch | Code search | – | [68] |
| Voice-enabled programming | Virtual assistant | NLP techniques | [49] |
| Virtual assistant and skill templates | Virtual Assistant | – | [69] |

**DeepL**, [67]**.** This API is a JSON-RPC API for direct translation of text from Visual Studio code. This plugin is a language translation service based on neural networks and deep learning algorithms, and is currently supported in German, English, French, Spanish, Italian, and Polish.

**Voice-Enabled Programming Extension.** [49] have introduced a VS Code extension that provides Voice-Enabled Programming. This model used the Language Understanding (LUIS) concept in the Microsoft Azure LUIS app. In general, extensions send query text to Azure and receive the results as JSON responses. Typical client extensions for LUIS are virtual assistants offering online chat communication via text or text-to-speech (chatbots).

In [15], the authors addressed the Eclipse Che structure. Figure 2 demonstrates the structure of AI-based extensions which can add to cloud based IDEs and bring the new structure (Fig. 3).

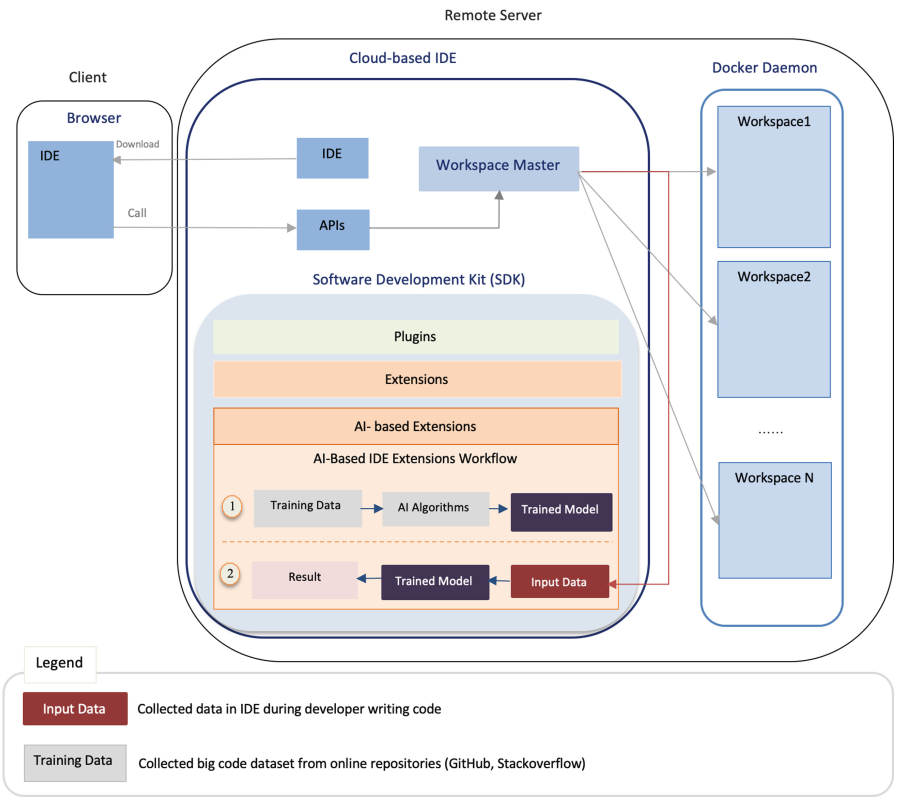

**Fig. 3.** Structure of AI-based extension in a Cloud IDE

## 6 Conclusion

There are some processes in software development such as implementing, debugging, bug detection, testing, etc. which can take the developers time. In this regard, IDEs include several tools that can be used to speed up these processes and provide automation. Recently, AI models have been utilized in IDEs to increase the automation. This growth can fall into two methods: 1) Improving current functionalities (features) 2) Adding new functionalities.

The first category includes improving current functionalities such as code autocomplete, code search, live templates, etc. These functionalities have been improved extensively by AI algorithms, especially through recommendation system models. The second category involves new IDE features such as AI virtual assistance, code summarization, etc. Some fields within AI have been more attended to so as to provide IDE features. Language modeling based on the sequence to sequence models, recommender systems, learning from existing code, and source code analyzing are some instances to name a few.

According to the literature, generative models and language modeling (RNN) have presented a powerful role in IDE learning models (autocomplete, test generation, code summarizing, etc.). However, learning long-term text is still a challenge. In this regard, attention-based neural networks have been investigated in recent academic works.

Moreover, there are some efforts that have used NLP techniques for promoting conversational experiences (Bots) and the interaction between developers and projects.

In the present study, Theia has been exclusively taken into account. Not only does Theia have most of the basic code editor features, but it also has significant extendibility which includes using VS code extensions.

Modern IDEs can bring many benefits, however, there can also be challenges in using online IDEs which must be addressed by researchers and engineers:

– One of the main features of online IDEs is that they can provide desktop and online IDE version. However, the findings show that some extensions, which work well in the desktop version, may encounter problems in the online version.
– Web technologies are changing fast, making maintenance difficult. Therefore, using high-level designs, such as templates, can be useful when developing extensions.
– The embedded learning models should bring a real-time application which needs to have a lightweight training model and low-latency prediction. Therefore, this feature should be considered when engineers choose an extension or develop it.
– When developing a Cloud IDE, it is important that it provide basic IDE features and support different languages, versioning tools, databases (SQL, NoSQL), cross-platform and multimedia development, online debugging, etc. Although most of the theses features can be found in extension market places, the serious projects need more guaranty. This is because maintenance, versioning, and extending these extensions may be challenging. Therefore, if a new IDE product is designed for the use of several extensions, engineers must carefully choose the extensions to minimize the risk that other extensions from other IDE functionalities have other requirements. Moreover, extensions should be investigated in terms of maintenance.

**Acknowledgements.** Supported by the project "Intelligent and sustainable mobility supported by multi-agent systems and edge computing (InEDGEMobility): Towards Sustainable Intelligent Mobility: Blockchain-based framework for IoT Security", Reference: RTI2018–095390-B-C32, financed by the Spanish Ministry of Science, Innovation and Universities (MCIU), the State Research Agency (AEI) and the European Regional Development Fund (FEDER).

## References

1. Yigitcanlar, T., Butler, L., Windle, E., Desouza, K.C., Mehmood, R., Corchado, J.M.: Can building "artificially intelligent cities" safeguard humanity from natural disasters, pandemics, and other catastrophes? An urban scholar's perspective. Sensors **20**(10), 2988 (2020)
2. Chamoso, P., González-Briones, A., Prieta, F.D.L., Venyagamoorthy, G.K., Corchado, J.M.: Smart city as a distributed platform: toward a system for citizen-oriented management. Comput. Commun. **152**, 323–332 (2020)
3. Gasparic, M., Murphy, G.C., Ricci, F.: A context model for IDE-based recommendation systems. J. Syst. Softw. **128**, 200–219 (2017)
4. Theia, E.: Platform to develop Cloud & Desktop (2019). https://theia-ide.org/. Accessed 2020
5. Bengio, Y., Ducharme, R., Vincent, P., Jauvin, C.: A neural probabilistic language model. J. Mach. Learn. Res. **3**, 1137–1155 (2003)
6. Rustan, K., Leino, M., Wüstholz, V.: The Dafny integrated development environment. arxiv Preprint arxiv:1404.6602 (2014)
7. Cloud9, Cloud IDE. https://aws.amazon.com/cloud9/. Accessed 2021
8. Codeanywhere, Cloud IDE. https://codeanywhere.com/. Accessed 2021
9. Eclipse Che, Eclipse next-generation IDE. https://www.eclipse.org/che/. Accessed 2021
10. Omori, T., Hayashi, S., Maruyama, K.: A survey on methods of recording fine-grained operations on integrated development environments and their applications. Comput. Softw. **32**(1), 60–80 (2015)
11. Aho, T., et al.: Designing ide as a service. Commun. Cloud Softw. **1**(1) (2011)
12. Barenkamp, M., Rebstadt, J., Thomas, O.: Applications of AI in classical software engineering. AI Perspect. **2**(1), 1–15 (2020)
13. Corchado, J.M., et al.: Deepint.net: a rapid deployment platform for smart territories. Sensors **21**(1), 236 (2021)
14. Arora, P., Dixit, A.: Analysis of cloud IDEs for software development. Int. J. Eng. Res. General Sci. **4**(4) (2016)
15. Applis, L.: Theoretical evaluation of the potential advantages of cloud ides for research and didactics. In: SKILL 2019-Studierendenkonferenz Informatik (2019)
16. Lin, Z.-Q., et al.: Intelligent development environment and software knowledge graph. J. Comput. Sci. Technol. **32**(2), 242–249 (2017)
17. Allamanis, M., Barr, E.T., Devanbu, P., Sutton, C.: A survey of machine learning for big code and naturalness. ACM Comput. Surv. (CSUR) **51**(4), 1–37 (2018)
18. Wood, A., Rodeghero, P., Armaly, A., McMillan, C.: Detecting speech act types in developer question/answer conversations during bug repair. In: Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp. 491–502 (2018)
19. Cooper, K., Torczon, L.: Engineering a Compiler. Elsevier, Amsterdam (2011)

20. Wang, Z., O'Boyle, M.: Machine learning in compiler optimization. Proc. IEEE **106**(11), 1879–1901 (2018)
21. Chen, T., et al.: {TVM}: an automated end-to-end optimizing compiler for deep learning. In: 13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 2018), pp. 578–594 (2018)
22. Nguyen, A.T., et al.: API code recommendation using statistical learning from fine-grained changes. In: Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, pp. 511–522 (2016)
23. Loaiza, F.L., Wheeler, D.A., Birdwell, J.D.: A partial survey on AI technologies applicable to automated source code generation. Technical report, Institute for Defense Analyses Alexandria United States (2019)
24. TabNine, Autocompletion with deep learning 2019. https://www.kite.com/. Accessed 2020
25. Gazzola, L., Micucci, D., Mariani, L.: Automatic software repair: a survey. IEEE Trans. Software Eng. **45**(1), 34–67 (2017)
26. Martinez, M., Monperrus, M.: Astor: exploring the design space of generate-and-validate program repair beyond GenProg. J. Syst. Softw. **151**, 65–80 (2019)
27. Hata, H., Shihab, E., Neubig, G.: Learning to generate corrective patches using neural machine translation. arXiv preprint arXiv:1812.07170 (2018)
28. Chen, Z., Kommrusch, S.J., Tufano, M., Pouchet, L.-N., Poshyvanyk, D., Monperrus, M.: SEQUENCER: sequence-to-sequence learning for end-to-end program repair. IEEE Trans. Softw. Eng. (2019)
29. Gu, X., Zhang, H., Kim, S.: Deep code search. In: 2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE), pp. 933–944. IEEE (2018)
30. Cambronero, J., Li, H., Kim, S., Sen, K., Chandra, S.: When deep learning met code search. In: Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp. 964–974 (2019)
31. Portolan, M.: Automated testing flow: the present and the future. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **39**(10), 2952–2963 (2019)
32. Godefroid, P., Singh, R., Peleg, H.: Machine learning for input fuzzing. US Patent App. 15/638,938, 4 October 2018
33. Sutton, M., Greene, A., Amini, P.: Fuzzing: Brute Force Vulnerability Discovery. Pearson Education (2007)
34. Godefroid, P., Levin, M.Y., Molnar, D.: Automated whitebox fuzz testing. In: Proceedings of NDSS (2008)
35. Gupta, R., Pal, S., Kanade, A., Shevade, S.: DeepFix: fixing common C language errors by deep learning. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 31 (2017)
36. Casado-Vara, R., Rey, A.M.-d., Affes, S., Prieto, J., Corchado, J.M. : IoT network slicing on virtual layers of homogeneous data for improved algorithm operation in smart buildings. Future Gener. Comput. Syst. **102**, 965–977 (2020)
37. Coronado, E., Mastrogiovanni, F., Indurkhya, B., Venture, G.: Visual programming environments for end-user development of intelligent and social robots, a systematic review. J. Comput. Lang. **58**, 100970 (2020)
38. Vinyals, O., Toshev, A., Bengio, S., Erhan, D.: Show and tell: a neural image caption generator. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 3156–3164 (2015)
39. Beltramelli, T.: pix2code: generating code from a graphical user interface screenshot. In: Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems, pp. 1–6 (2018)

40. Pang, X., Zhou, Y., Li, P., Lin, W., Wu, W., Wang, J.Z.: A novel syntax-aware automatic graphics code generation with attention-based deep neural network. J. Netw. Comput. Appl. **161**, 102636 (2020)
41. JetBrains, High-speed coding with Custom Live Templates. https://www.jetbrains.com/help/idea/using-live-templates.html. Accessed 2020
42. Murphy-Hill, E.: Continuous social screencasting to facilitate software tool discovery. In: 2012 34th International Conference on Software Engineering (ICSE), pp. 1317–1320. IEEE (2012)
43. Gasparic, M., Janes, A., Ricci, F., Murphy, G.C., Gurbanov, T.: A graphical user interface for presenting integrated development environment command recommendations: design, evaluation, and implementation. Inf. Softw. Technol. **92**, 236–255 (2017)
44. Gasparic, M., Gurbanov, T., Ricci, F.: Improving integrated development environment commands knowledge with recommender systems. In: Proceedings of the 40th International Conference on Software Engineering: Software Engineering Education and Training, pp. 88–97 (2018)
45. LeClair, A., Haque, S., Wu, L., McMillan, C.: Improved code summarization via a graph neural network. In: Proceedings of the 28th International Conference on Program Comprehension, ICPC 2020, pp. 184–195. Association for Computing Machinery, New York (2020)
46. Oda, Y., et al.: Learning to generate pseudo-code from source code using statistical machine translation (t). In: 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 574–584. IEEE (2015)
47. Iyer, S., Konstas, I., Cheung, A., Zettlemoyer, L.: Summarizing source code using a neural attention model. In: Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pp. 2073–2083 (2016)
48. Bedia, M.G., Rodríguez, J.M.C., et al.: A planning strategy based on variational calculus for deliberative agents (2002)
49. Joshi, P., Bein, D.: Audible code, a voice-enabled programming extension of visual studio code. In: Latifi, S. (eds.) 17th International Conference on Information Technology-New Generations (ITNG 2020). AISC, vol. 1134, pp. 335–341. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-43020-7_44
50. Virtual Assistant and Skill Templates. https://marketplace.visualstudio.com/items?itemName=BotBuilder.VirtualAssistantTemplate. Accessed 2020
51. Xu, F.F., Vasilescu, B., Neubig, G.: In-ide code generation from natural language: promise and challenges. arXiv preprint arXiv:2101.11149 (2021)
52. Wong, E., Yang, J., Tan, L.: Autocomment: mining question and answer sites for automatic comment generation. In: 2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 562–567. IEEE (2013)
53. Xing, H., Li, G., Xia, X., Lo, D., Jin, Z.: Deep code comment generation with hybrid lexical and syntactical information. Empir. Softw. Eng. **25**(3), 2179–2217 (2020)
54. Sidhanth, N., Sanjeev, S., Swettha, S., Srividya, R.: A next generation ide through multi tenant approach. Int. J. Inf. Electron. Eng. **4**(1), 27 (2014)
55. Shi, S., Li, Q., Le, W., Xue, W., Zhang, Y., Cai, Y.: Intelligent workspace. US Patent 9,026,921, 5 May 2015
56. Eclipse Foundation (2020). https://ecdtools.eclipse.org/. Accessed 2021

57. Saini, R., Bali, S., Mussbacher, G.: Towards web collaborative modelling for the user requirements notation using Eclipse Che and Theia IDE. In: 2019 IEEE/ACM 11th International Workshop on Modelling in Software Engineering (MiSE), pp. 15–18. IEEE (2019)
58. Kahlert, T., Giza, K.: Visual studio code tips & tricks, vol. 1. Microsoft Deutschland GmbH (2016)
59. Bierman, G., Abadi, M., Torgersen, M.: Understanding typescript. In: Jones, R. (eds.) ECOOP 2014. LNCS, vol. 8586, pp. 257–281. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44202-9_11
60. Inversify, lightweight inversion of control (IoC) container for TypeScript and JavaScript apps (2018). https://github.com/inversify/InversifyJS. Accessed 2021
61. langserver, Language Server protocol. https://langserver.org/. Accessed 2020
62. Bünder, H.: Decoupling language and editor-the impact of the language server protocol on textual domain-specific languages. In: MODELSWARD, pp. 129–140 (2019)
63. Microsoft. VS Marketplace, Extensions for the Visual Studio products. https://marketplace.visualstudio.com/. Accessed 2021
64. Kite, AI powered code completions (2019). https://www.kite.com/. Accessed 2020
65. Kite visualstudio. https://marketplace.visualstudio.com. Accessed 2021
66. Flutter, Dart-Code. https://marketplace.visualstudio.com/items?itemName=Dart-Code.flutter. Accessed 2021
67. deepl, AI powered code completions (2019). https://www.deepl.com/en/docs-api/. Accessed 2020
68. VSearch code. https://marketplace.visualstudio.com/items?itemName=mario-0.VSearch102. Accessed 2021
69. Virtual Assistant and Skill Templates. https://marketplace.visualstudio.com/items?itemName=BotBuilder.VirtualAssistantTemplate. Accessed 2021