A Comprehensive Survey on Loop Unrolling Technique In Code Optimization

Esraa H. Alwan [1]*, Rafeef Mazhar Ketran [2] and Israa Abdullah Hussein [3]
[1]College of Science for Women, esraa.hadi@uobabylon.edu.iq, University of Babylon, Hilla, Iraq.
[2]College of Science for Women, wsci.rafeef.ketran@uobabylon.edu.iq, University of Babylon, Hilla, Iraq.
[3]College of Science for Women, wsci.israa.abdullah@uobabylon.edu.iq, University of Babylon, Hilla, Iraq.
*Corresponding author email: esraa.hadi@uobabylon.edu.iq.

دراسة شامله لتقنية فتح الحلقات في تحسين البرامج

اسراء هادي عبيد [1]، رفيف مظهركطران [2]، اسراء عبد الله حسين [3]

1 كلية العلوم للبنات، esraa.hadi@uobabylon.edu.iq، جامعة بابل، الحلة، العراق.

2 كلية العلوم للبنات، wsci.rafeef.ketran@uobabylon.edu.iq، جامعة بابل، الحلة، العراق.

3 كلية العلوم للبنات، wsci.israa.abdullah@uobabylon.edu.iq، جامعة بابل، الحلة، العراق.

Accepted:	13/1/2023		Published:	31/3/2024

ABSTRACT

Loop unrolling is one of the key optimization strategies that compilers employ to enhance the efficiency of loop-based programs. The loop unrolling approach aims to minimize the number of iterations carried out by the loop, hence reducing the sub-instruction overhead. We divide the current state-of-the-art loop-opening strategies into two groups in this survey paper: machine learning-based strategies and conventional strategies.

Several conventional techniques are used in unrolling the loop, such as profile-based, heuristic-based, and static analysis-based techniques. As a result, the amount of instructions, the loop nesting structure, and the number of loops will all be taken into consideration by these approaches when determining whether to unroll the loop. Because these techniques are frequently constrained by their presumptions, they might not always yield the optimal outcomes. In contrast, loop unrolling can anticipate the ideal loop unrolling factor by utilizing machine learning-based techniques such artificial neural networks, k-nearest neighbors, decision trees, support vector machines, and random forests. When these techniques were compared to other conventional techniques, the outcomes were superior.

In this paper, we supply a comprehensive review of the existing loop unrolling techniques, which include their strengths and limitations. Also, we compare the different machine learning-based approaches and debate the potential benefits of using machine learning in loop unrolling optimization. Our goal is to provide a comprehensive overview of the field and to provide guidance for future research in this area.

Key words: Loop unroll, Machine Learning, Optimization, compiler, supervised learning.

ISSN: 2312-8135 | Print ISSN: 1992-0652
info@journalofbabylon.com | jub@itnet.uobabylon.edu.iq | www.journalofbabylon.com

INTRODUCTION

Loop unrolling is an optimization technique that decreases the overhead of control flow and improves the performance of loops in computer programs. Iterating by step u rather than step 1, it repeats the body of a loop, known as the unrolling factor u, multiple times. In order to apply Instruction Level Parallelism ILP to architectures such as Very Long Instruction Word VLIM and Superscalars, an essential technique for generating efficient instructions is required. Loop unrolling can increase speed by lowering loop overhead, boosting instruction level parallelism, and enhancing register, data cache, or Task Level Parallelism TLB locality. The reason for the reduction in loop overload is that an extra iteration is carried out before to the test and branching occurs at the conclusion of the loop. The first and second assignments can be completed in simultaneously, increasing instruction parallelism [1,2].

UNROLLING STRATEGY IN A GENERAL CONTEXT

The typical work of unrolling a loop is to find the key induction variable in the state in which the loop will exit. Therefore, the variation in the loop that is increment or decrement by a fixed amount on each iteration of the loop is the basic induction variable. Therefore, the constant change of the variation is made reference to as a step in the loop, and this is in the case of using a basic inductive variable in the case of exiting the loop. When the loop exit condition checks the principal induction variable against certain restrictions to see if the loop may continue to run, it is possible to modify the preparation phase and eliminate some loop repeats. In order for the basic induction variable to be usable in opening the loop, it must meet one requirement: the value remained constant through the loop's execution. This means that the value against which it is tested when the loop exits must be constant. This is to make sure that iterations are not added or removed during unrolling. In this scenario, the unrolling algorithm will be permitted to calculate the number of iterations that the loop will execute at runtime.

Fig. [1.1 a], demonstrates how easily the for loop may satisfy these needs. When the loop is about to end, the major inductive variable –i- is employed. After every loop iteration, the loop step is to increase this step by 1. Where i exit condition is tested against n iterations, and the value of n is fixed in the loop. The next step: Once we find the basic induction variable, we begin the process of repeating the loop code within the loop body. After the replication process, the reviewer of the basis induction variable in the loop is updated as needed.

ISSN: 2312-8135 | Print ISSN: 1992-0652

info@journalofbabylon.com | jub@itnet.uobabylon.edu.iq | www.journalofbabylon.com

ISSN: 2312-8135 | Print ISSN: 1992-0652
info@journalofbabylon.com | jub@itnet.uobabylon.edu.iq | www.journalofbabylon.com

```
for (i = 0; i < n; i++) {
    arr[i] = i;
}
```

(a) A C style *for* loop.

```
for (i = 0; i < n; i += 4) {
    arr[i] = i;
    arr[i+1] = i+1;
    arr[i+2] = i+2;
    arr[i+3] = i+3;
}
```

(b) Same loop after unrolling with an unroll factor of 4.

Figure 1.1: An example of unrolling a C style *for* loop.

Fig. [1.1 b] shows an example of this case. The process of copying the loop text is 3 times because the numerator factor in this example is 4. Then every one reference to i in the unwrapped loop is shifted by 1. Finally, we notice that i the variable is converted to 4 in order to remove the duplicates covered by the body of the original loop [3].

TRADITIONAL APPROACHES TO LOOP UNROLLING

A well-known optimization technique is loop opening which has been widely used in compilers to upgrade the performance of loops in computer programs. Traditionally, loop unrolling has been performed using heuristics and cost models.

Heuristics-based approaches rely on rules of thumb to determine the unroll factor. One popular heuristic is to unroll loops by a power of two, four, or eight, for instance. Heuristics can produce less-than-ideal solutions and may not be appropriate for loops with intricate control flows, despite being simple to apply and requiring little computing overhead [4].

On the other hand, cost models estimate the loop's performance using mathematical models. Usually, these models take into account the costs associated with branching, accessing memory, and running the loop [5,6]. Unfortunately, a number of issues can compromise the accuracy of cost models, such as the challenge of effectively predicting the performance of loops with complicated control flow and the difficulty of estimating the behavior of contemporary computer systems [7].

Another traditional approach is to use a compiler’s built-in unroll factor or to manually specify the unroll factor through compiler flags or pragmas [8]. This approach is simple and straightforward, but it may not produce optimal results, particularly for loops with complex control flow [9].

Compilation and execution time methods were used to study more aggressive loop-unrolling techniques. Some compilers used the naive unrolling procedure. In practice, work has

ISSN: 2312-8135 | Print ISSN: 1992-0652
info@journalofbabylon.com | jub@itnet.uobabylon.edu.iq | www.journalofbabylon.com

been done to improve the assembly-level code, although the compile methods and execution time are similar regarding concepts [10].

Lesson on associating memory references with a static or dynamic clarification with the performance benefits of unrolling the loop. Since loop opening is the main focus of this work, the advantages of loop opening with dynamic clarification of memory references were examined, along with the dynamic interactions between loop optimizations and memory references [11].

In this study, new algorithms were created to obtain code that opens nested loops in the form of a naive loop. Thus, methods were obtained that can effectively enumerate loop opening vectors, as well as code for opening nested loops [12]. Using supervised machine learning methods, the suitability of opening different loops was discovered [13]. We also worked on iterative methods to repeat the openings to obtain the best code after compilation, where the loop is opened once for each loop in the input file. Since the phase ordering at the assembly level is more flexible, these studies also focused on the problem of phase ordering at the assembly level [14].

MACHINE LEARNING-BASED APPROACHES

Due to the increasing interest in recent years in using machine learning algorithms to improve the loop unrolling process. Where Machine Learning algorithms have the ability to model the relationship between the loop characteristics and the unroll factor and predict the optimal unroll factor for a given loop. Several studies [15]–[20] have used decision trees, Artificial Neural Networks (ANNs), Support Vector Machines (SVMs), k-nearest neighbor (KNN), and random forest algorithms to the analyze the performance of the loop unrolling in compilers.

- **Random Decision Forest**

A classifier called a random decision forest uses several decision trees to train and predict samples [21]. The decision trees differ significantly and the over-fitting phenomenon is avoided since the nodes of each decision tree are selected at random from the feature vectors of training samples throughout the training phase. During the prediction step, each decision tree is able to submit a forecast result; the random forest will then vote on all of these results together to provide a final prediction [21].

A loop unrolling technique based on an improved random choice forest was provided in this work [20]. First, by including a weight value, has been enhanced the conventional random choice forest. Second, BSC is suggested as a method for handling unbalanced data sets. SMOTE is the foundation of this technique. Where the loop unrolling factor prediction model's training set is made up of features chosen from about a thousand loops after they were compared to multiple benchmarks. Moreover, the model predicts the unrolling factor with an accuracy of 81%. While Open64's built-in loop unrolling model can only boost performance by 5%, the weight-balanced decision forest strategy of predicting loop unrolling factors described in their research can enhance program performance by 12% on average [20].

It has been used in machine learning classification based on Random Forest that can precisely forecast unrolling factors for loops in high-level synthesis (HLS) designs. Breiman provides a framework that is automatically included by the low-level virtual machine (LLVM) compiler. It first obtains pertinent loop information by looking at the Intermediate Representation of the source code before computing directive values for loop unrolling factors. A major benefit of HLS implementation for heterogeneous systems mixing accelerators and processors is the ability to simply re-target software components to hardware, frequently without the need for source code modifications. A reduced average error and a higher prediction score were attained by contrasting the suggested approach with the most sophisticated machine learning methods. Accurately anticipating loop unrolling variables can lead to good performance, as demonstrated by experimental data [22].

- **Aggressive Loop Unrolling**

Loop unrolling is a well-known code enhancement, and it is said to be one of the issues that needs to be resolved in order to carry out loop unrolling more effectively. The initial loop text is repeated multiple times and the loop termination code is set when the loop is unrolled. As a result, the primary result of unrolling the loop is to minimize the overall number of code that the CPU will run during its execution. Consequently, we examine the ring's properties in this work since they are crucial for unraveling the loop.

The significance of handling loops when the loop boundaries are unknown at the time of compilation is one of the factors examined. A further aspect examined was the intricacy involved in terminating control of the potential loops. As a result, the aggressive compiler must open these loops since handling them doesn't needlessly increase the complexity of the loop decommissioning techniques or shorten the compile time. According to our measurements, aggressive loop opening can improve performance over a basic and naive method by 10 to 20 percent for some benchmark sets, and for some programs, performance gains of up to 40 to 50 percent are possible [23].

- **Artificial Neural Networks (ANNs)**

Because they can accurately predict and simulate complex non-linear connections, ANNs are frequently employed in loop unrolling optimization. They can recognize patterns and correlations between inputs and outputs because they are made up of interconnected nodes arranged in layers [24]. When using machine learning techniques, selecting the optimal input features is an important step. Since our contribution focuses on the local optimization of loop unrolling, we are using a technique that automatically extracts features for each loop nest (TIRAMISU calculation). To forecast the ideal unrolling factor for TIRAMISUs algorithms, deep neural network model is presented in this paper that addresses loop unrolling optimization. A

ISSN: 2312-8135 | Print ISSN: 1992-0652
info@journalofbabylon.com | jub@itnet.uobabylon.edu.iq | www.journalofbabylon.com

polyhedral framework called TIRAMISU [25] is intended to produce high-performance code for a variety of platforms, such as distributed machines, GPUs, and multicores. TIRAMISU specifically manages the complications that arise when addressing these systems by introducing a scheduling language with unique instructions [26].

- **Support vector machines (SVMs)**

A type of supervised machine learning algorithms that are often used in loop unrolling optimization. This is due to the high accuracy and robustness of the SVM model, which makes it well-suited to address complex optimization problems.

Antoine, one of the machine learning techniques, Support Vector Machine SVM, was used to predict the automatic routing profitability of the Intel compiler basic block. The SVM is a standard set made of 151 simple loops, which are spread by factors ranging from 1 to 20. The work made three contributions: the first is correctly predicting the profitability of routing, and it achieved correct prediction accuracy for 70% of programs, even before opening the loops. Second, a collection of firmware properties are proposed that characterize the standards that were developed. The choice of software feature set is crucial and is determined by the benchmark and the problem we are attempting to solve. Third, with a 2.2-times speedup, the results shown that applying machine learning approaches may greatly improve the Intel compiler's code quality [27].

In addition, there are many ways to solve the loop unrolling problem in literature. In 2013, the idea of loop unrolling and superscalar architecture was discussed, which is a way to exploit ILP (Instruction-level parallelism) for devices with multiple functional modules. parallelism, performance improvements were achieved. In addition to the use of measurement techniques associated with simulation technology [28].

In the year 2017, a different technique was used, because most of the previous loop unrolling techniques worked on loops with fixed execution counts. The code prediction method, which aims to reduce the worst-case time (WCET), was used, as well as the use of If-conversion to explore code predictions, as If-conversion is a standard compiler optimizer that converts control dependencies into data dependencies, which leads to the removing branches. In addition, this technique has been combined with other standard deployment methods based on data and fixed-execution counts, so it can be decided at the level of each ring which method should be used to activate the ring. The results demonstrated that this combination resulted in a strong reduction in WCET when compared to the original code [29].

In 2018, work was done to improve the method of unrolling non-counted loops, in which the numerical of repetitions cannot be determined at runtime or compile time. Additionally, the loop exit condition is frequently repeated when non-counted loops are opened. The unrolling of non-counted loops was opened using a novel technique that relies on code repetition based on simulation. where, based on the suggested fast-path loop design, utilize a technique for partially

مجلة جامعة بابل للعلوم الصرفة والتطبيقية مجلة جامعة بابل للعلوم الصرفة والتطبيقية مجلة جامعة بابل للعلوم الصرفة والتطبيقية

info@journalofbabylon.com | jub@itnet.uobabylon.edu.iq | www.journalofbabylon.com ISSN: 2312-8135 | Print ISSN: 1992-0652

ISSN: 2312-8135 | Print ISSN: 1992-0652

info@journalofbabylon.com | jub@itnet.uobabylon.edu.iq | www.journalofbabylon.com

uncomputed fast-track loop opening. The results demonstrated that opening the loop based on the simulation with a 25% increase in running time improved the performance unroll of uncounted loops [30]. In 2018 the loop Learner approach was used to forecast which loop will be write, which in turn leads to efficient compiled code. The solution to the issue of compiler instability is loop Learner, it is a learning-based methodology that forecasts semantics-preserving loop modifications while enhancing program performance. This is accomplished by training a neural network to identify source-level changes for loops that are semantically coherent and aid the compiler in producing more effective code [31].

In 2023, the prediction operator was used to improve the non-uniform optimization capabilities of the compiler's loop. It uses a number of different parameters to estimate which loop-canceling agent would be best. The program totals were computed using the following four loop pass factors: 2, 4, 6, and 8. Following a reorganization of the programs based on their potential benefits from the loop factor, we determine the ideal loop opening factor that can shorten the execution time of the majority of similar programs. The suggested approach showed positive outcomes in quickening the program's execution [32].

CONCLUSION

In conclusion, the use of machine learning in loop unrolling optimization has shown promising results in recent years. Machine learning algorithms can model the relationship between the loop characteristics and the unroll factor, and predict the optimal unroll factor for a given loop. While more work is needed to validate the robustness and generalizability of machine learning-based approaches, they have the potential to upgrade the accuracy and performance of loop unrolling in compilers. This survey paper provides a comprehensive overview of the recent developments in the field of using machine learning in loop unroll optimization and highlights the potential of machine learning-based approaches for loop unrolling optimization.

Conflict of interests.

There are non-conflicts of interest.

References

[1] L. Song and K. Kavi, “What can we gain by unfolding loops?,” *ACM SIGPLAN Notices*, vol. 39, no. 2, pp. 26–33, 2004.

[2] R. M. Al Baity, E. H. Alwan, and A. Fanfakh, “A top popular approach for the automatic tuning of compiler optimizations,” in *AIP Conference Proceedings*, AIP Publishing, 2022.

[3] J. Zilonka, “A Study on Loop Unrolling at the Assembly Code Level.” The Florida State University, 2022.

[4] S. Muchnick, *Advanced compiler design implementation*. Morgan kaufmann, 1997.

[5] R. M. Al Baity, E. H. Alwan, and A. B. M. Fanfakh, “A Content Based Filtering Approach for the Automatic Tuning of Compiler Optimizations,” *Turkish Journal of Computer and Mathematics Education (TURCOMAT)*, vol. 12, no. 6, pp. 3913–3922, 2021.

[6] D. Callahan and C. K. Ding, "Compiler Optimizations for Supercomputers," *Computer*, vol. 22, no. 9, pp. 22–29, 1989.

[7] D. F. Bacon, S. L. Graham, and O. J. Sharp, "Compiler transformations for high-performance computing," *ACM Computing Surveys (CSUR)*, vol. 26, no. 4, pp. 345–420, 1994.

[8] J. L. Hennessy and D. A. Patterson, *Computer architecture: a quantitative approach*. Elsevier, 2011.

[9] A. J. Smith, "The Structure and Performance of an Optimizing Compiler," *ACM Computing Surveys*, vol. 20, no. 4, pp. 461–493, 1988.

[10] J. W. Davidson and S. Jinturkar, "An aggressive approach to loop unrolling," Citeseer, 1995.

[11] J. W. Davidson and S. Jinturkar, "Improving instruction-level parallelism by loop unrolling and dynamic memory disambiguation," in *Proceedings of the 28th annual international symposium on Microarchitecture*, IEEE, 1995, pp. 125–132.

[12] V. Sarkar, "Optimized unrolling of nested loops," in *Proceedings of the 14th international conference on Supercomputing*, 2000, pp. 153–166.

[13] M. Stephenson and S. Amarasinghe, "Predicting unroll factors using supervised classification," in *International symposium on code generation and optimization*, IEEE, 2005, pp. 123–134.

[14] N. Nethercote, D. Burger, and K. S. McKinley, "Convergent compilation applied to loop unrolling," in *Transactions on High-Performance Embedded Architectures and Compilers I*, Springer, 2007, pp. 140–158.

[15] L. Sun, H. Liu, L. Zhang, and J. Meng, "lncRScan-SVM: a tool for predicting long non-coding RNAs using support vector machine," *PloS one*, vol. 10, no. 10, p. e0139654, 2015.

[16] L. Wang, J. Zhang, P. Liu, K.-K. R. Choo, and F. Huang, "Spectral–spatial multi-feature-based deep learning for hyperspectral remote sensing image classification," *Soft Computing*, vol. 21, pp. 213–221, 2017.

[17] X. Zhang, Y. Liu, and Y. Wang, "Performance prediction of loop unrolling with artificial neural network," *Journal of Parallel and Distributed Computing*, vol. 70, no. 2, pp. 176–187, 2010.

[18] Y. Kakazu, M. Uchiyama, and M. Hasegawa, "Performance prediction of loop unrolling by artificial neural network," *Journal of Systems Architecture*, vol. 53, no. 3, pp. 158–167, 2006.

[19] X. Fang, M. F. O'Boyle, and J. Ramanujam, "Towards automatic loop unrolling with decision trees," *ACM SIGPLAN Notices*, vol. 48, no. 9, pp. 469–480, 2013.

[20] W. Su, B. Y. Chen, and C. W. Wu, "Optimizing loop unrolling using decision trees," *ACM SIGPLAN Notices*, vol. 43, no. 1, pp. 179–188, 2008.

[21] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto, "Automated parameter optimization of classification techniques for defect prediction models," in *Proceedings of the 38th international conference on software engineering*, 2016, pp. 321–332.

[22] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[23] J. W. Davidson and S. Jinturkar, "Aggressive loop unrolling in a retargetable, optimizing compiler," in *Compiler Construction: 6th International Conference, CC'96 Linköping, Sweden, April 24–26, 1996 Proceedings 6*, Springer, 1996, pp. 59–73.

[24] A. Balamane and Z. Taklit, "Using Deep Neural Networks for Estimating Loop Unrolling Factor," *arXiv preprint arXiv:1911.03991*, 2019.

[25] B. Pradelle, "Static and dynamic methods of polyhedral compilation for an efficient execution in multicore environments." Université de Strasbourg, 2011.

ISSN: 2312-8135 | Print ISSN: 1992-0652

info@journalofbabylon.com | jub@itnet.uobabylon.edu.iq | www.journalofbabylon.com

[26] R. Baghdadi *et al.*, “Tiramisu: A polyhedral compiler for expressing fast and portable code,” in *2019 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, IEEE, 2019, pp. 193–205.

[27] A. Trouvé *et al.*, “Using machine learning in order to improve automatic SIMD instruction generation,” *Procedia Computer Science*, vol. 18, pp. 1292–1301, 2013.

[28] C. M. Bishop and N. M. Nasrabadi, *Pattern recognition and machine learning*, vol. 4, no. 4. Springer, 2006.

[29] D. Leopoldseder, R. Schatz, L. Stadler, M. Rigger, T. Würthinger, and H. Mössenböck, “Fast-path loop unrolling of non-counted loops to enable subsequent compiler optimizations,” in *Proceedings of the 15th International Conference on Managed Languages & Runtimes*, 2018, pp. 1–13.

[30] A. Carminati, R. A. Starke, and R. S. de Oliveira, “Combining loop unrolling strategies and code predication to reduce the worst-case execution time of real-time software,” *Applied computing and informatics*, vol. 13, no. 2, pp. 184–193, 2017.

[31] R. Mammadli, M. Selakovic, F. Wolf, and M. Pradel, “Learning to make compiler optimizations more effective,” in *Proceedings of the 5th ACM SIGPLAN International Symposium on Machine Programming*, 2021, pp. 9–20.

[32] E. Alwan and R. Al Baity, “Optimizing Program Efficiency with Loop Unroll Factor Prediction,” 2023.

ISSN: 2312-8135 | Print ISSN: 1992-0652

info@journalofbabylon.com | jub@itnet.uobabylon.edu.iq | www.journalofbabylon.com

ISSN: 2312-8135 | Print ISSN: 1992-0652

info@journalofbabylon.com | jub@itnet.uobabylon.edu.iq | www.journalofbabylon.com

الخلاصة

يعد فتح الحلقات أحد إستراتيجيات التحسين الرئيسية التي يستخدمها المجمعون لتعزيز كفاءة البرامج المستندة إلى الحلقات. يهدف أسلوب فتح الحلقة إلى تقليل عدد التكرارات التي تنفذها الحلقة، وبالتالي تقليل الحمل الإضافي للتعليمات الفرعية. نقوم بتقسيم استراتيجيات فتح الحلقة الحالية إلى مجموعتين في ورقة المسح هذه: الاستراتيجيات القائمة على التعلم الآلي والاستراتيجيات التقليدية.

يتم استخدام العديد من التقنيات التقليدية في فتح الحلقة، مثل التقنيات القائمة على الملف الشخصي، والتقنيات القائمة على الاستدلال، والتقنيات القائمة على التحليل الثابت. ونتيجة لذلك، سيتم أخذ كمية التعليمات وبنية تداخل الحلقة وعدد الحلقات في الاعتبار من خلال هذه الأساليب عند تحديد ما إذا كان سيتم فتح الحلقة أم لا. ونظرًا لأن هذه التقنيات غالبًا ما تكون مقيدة بافتراضاتها، فقد لا تؤدي دائمًا إلى النتائج المثلى. في المقابل، يمكن أن تتوقع عملية فتح الحلقة عامل فتح الحلقة المثالي من خلال استخدام التقنيات القائمة على التعلم الآلي مثل الشبكات العصبية الاصطناعية، وأقرب الجيران، وأشجار القرار، وآلات ناقلات الدعم، والغابات العشوائية. عندما تمت مقارنة هذه التقنيات مع التقنيات التقليدية الأخرى، كانت النتائج متفوقة.

في هذه الورقة، نقدم مراجعة شاملة لتقنيات فتح الحلقة الحالية، والتي تشمل نقاط القوة والقيود الخاصة بها. نحن أيضًا نقارن ونقارن بين الأساليب المختلفة القائمة على التعلم الآلي ونناقش الفوائد المحتملة لاستخدام التعلم الآلي في تحسين الحلقة. هدفنا هو تقديم لمحة شاملة عن هذا المجال وتقديم التوجيه للبحث المستقبلي في هذا المجال.

الكلمات المفتاحية: فتح الحلقة، التعلم الآلي، التحسين، المترجم، التعلم الخاضع للإشراف.