

Article

Quantitative Evaluation of Automatic Translation Between Industrial Robot Programming Languages Using Machine Learning

Nathaniel Morales-Centla , Richard Torrealba-Meléndez * , Edna Iliana Tamariz-Flores, César Augusto Arriaga-Arriaga  and Mario López-López 

Faculty of Electronics Sciences, Autonomous University of Puebla, Puebla 72570, Mexico; nathaniel.moralesc@alumno.buap.mx (N.M.-C.); iliana.tamariz@correo.buap.mx (E.I.T.-F.); cesarau.arriaga@correo.buap.mx (C.A.A.-A.); mario.lopezlop@correo.buap.mx (M.L.-L.)

* Correspondence: richard.torrealba@correo.buap.mx; Tel.: +52-222-2-29-55-00 (ext. 7400)

Abstract

This paper presents a quantitative evaluation of an automatic translation system between industrial robot programming languages based on a sequence-to-sequence (Seq2Seq) neural architecture using Long Short-Term Memory (LSTM) networks. The study addresses the interoperability problem between proprietary robot programming languages by proposing a data-driven approach capable of learning correspondences between structured code instructions. A parallel dataset of 28,000 aligned instruction pairs was constructed and pre-processed through tokenization and normalization to enable structured sequence learning. The model was trained under four configurations (50, 100, 150 and 200 epochs) to analyze the impact of training duration on performance and generalization capability. The system was evaluated using multiple quantitative metrics, including accuracy, loss, BLEU, and Exact Match (EM), allowing assessment of both structural similarity and exact sequence correctness. Experimental results demonstrate that the 200-epoch configuration improves the performance across all metrics, achieving an accuracy of 0.9943, a BLEU score of 0.682, and an Exact Match of 0.970 on the test set. These results indicate that the model is capable of generating both structurally consistent and syntactically correct translations. The analysis shows that while BLEU captures structural similarity, EM provides a stricter evaluation of exact sequence correctness, which is critical in structured code translation tasks where minor variations may affect execution. The proposed approach demonstrates the feasibility of applying neural machine translation techniques to industrial robot programming, contributing to improved interoperability and reduced manual effort in multi-platform robotic environments.



Academic Editors: Liviu Marian Ungureanu, Iulian Sorin Munteanu and Hongying Meng

Received: 18 June 2026

Revised: 27 July 2026

Accepted: 31 July 2026

Published: 5 August 2026

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

Keywords: industrial robot programming; industrial robotics; code translation; neural machine translation (NMT); sequence-to-sequence models; LSTM networks; robot language interoperability; structured code translation; Exact Match; BLEU metric

1. Introduction

Modern industrial automation depends heavily on programmable robots capable of executing high-precision tasks in complex production environments, where accurate motion representation and control are essential [1–4]. However, one of the main technical problems in the manufacturing industry is the lack of interoperability between different

robot programming environments [1,3,5]. Each manufacturer develops proprietary languages with significant differences in syntax, control structures, motion definitions, and tool configuration [3,6,7]. As a consequence, the migration and reuse of programs between heterogeneous robotic platforms becomes complex and inefficient.

When companies expand their automated infrastructure by integrating robots from different manufacturers, program adaptation is typically performed manually by specialized personnel, leading to high engineering costs, downtime, and risks of human error [1,5,8]. This situation limits manufacturing flexibility and contrasts with the principles of Industry 4.0, which are oriented toward intelligent, adaptive, and interoperable systems [2,4,9].

To address this problem, robot programming has evolved toward offline programming (OLP) approaches, post-processing techniques, and conversion tools for proprietary systems [8,10,11]. Nevertheless, most of these solutions rely on expert-defined rules, which do not guarantee complete semantic correspondence between source and target programs [1,3,10].

The literature reports several approaches to translation and interoperability in industrial and software domains. Some works employ post-processors that convert trajectories generated in CAD/CAM environments into manufacturer-specific code [10], while others propose offline programming frameworks that generate robotic programs from abstract representations [11]. In parallel, recent studies have explored neural approaches to translating structured and formal languages, including code translation benchmarks and sequence-to-sequence models [7,12–16]. Likewise, advances in language models have shown the ability to translate natural-language instructions into executable commands and to generate robotic control policies, highlighting the convergence between machine translation and robot programming [6,14,15,17,18].

Despite these advances, evaluating translation quality in industrial programming languages remains a critical challenge. Traditional metrics such as BLEU assess structural similarity by comparing n-gram matches between generated sequences and references [19,20]. However, BLEU was designed for natural language and may not adequately reflect the correctness of structured code, where small differences can result in non-executable programs. In contrast, the Exact Match (EM) metric enables strict evaluation by verifying whether the generated sequence is identical to the reference, making it especially suitable for assessing exact sequence correctness in industrial contexts.

In recent years, artificial intelligence, particularly Neural Machine Translation (NMT), has achieved outstanding results in sequence modeling tasks [21–23]. Sequence-to-sequence (Seq2Seq) architectures based on LSTM networks have proven effective at learning correspondences between variable-length sequences [21,24–28]. Nevertheless, their application to industrial robot programming languages remains limited, especially in contexts where strict syntactic and execution constraints must be satisfied [3,12–15].

In this work, an artificial-intelligence-based approach is proposed for automatic translation between two industrial robot programming languages. The study focuses on a specific language pair and aims to conduct a detailed quantitative analysis in structured code scenarios.

To this end, a parallel dataset of 28,000 aligned samples was constructed. The programs were preprocessed with tokenization and normalization, enabling their representation as structured sequences suitable for training an LSTM-based Seq2Seq model [21,24,25].

The performance of the proposed system was evaluated using quantitative metrics commonly used in machine translation, including accuracy, BLEU, Exact Match (EM), and loss [19,20]. This approach enables the analysis of not only structural similarity but also exact sequence correctness, model convergence, and training stability across the training, validation, and test sets.

The main contribution of this work is the development and evaluation of a data-driven neural approach for translating structured robotic programs. Unlike traditional rule-based methods, the proposed model learns both structural and functional correspondences across heterogeneous languages. In addition, a comprehensive quantitative analysis is presented using multiple evaluation metrics, enabling the identification of the model's strengths and limitations in industrial code translation.

The results indicate the feasibility of using artificial intelligence to improve interoperability and reduce manual effort in industrial robotic systems, focusing on a specific case study involving translation from FANUC to KUKA (KRL) programming languages.

Automatic translation between programming languages has been extensively studied in software engineering and artificial intelligence. In the context of industrial robotics, this problem becomes more complex due to proprietary languages developed by different manufacturers, which limit interoperability among heterogeneous robotic systems [1,5,29].

Several studies have analyzed programming methods for industrial robots. Bilancia et al. [1] present an overview of control and programming approaches in modern industrial environments, while Tamizi et al. [5] highlight the evolution of motion planning techniques and their relationship to the use of advanced computational tools.

Traditional solutions for robotic program conversion typically rely on offline programming (OLP) and post-processing techniques. For example, Souza et al. [8] propose a framework that can convert offline-generated programs into native code compatible with various robotic controllers. Similarly, Nagata et al. [10] developed post-processing methods that convert trajectories generated in CAD/CAM environments into executable programs. However, these approaches rely heavily on manually defined deterministic rules, limiting their generalization capability.

Moreover, recent works have explored artificial-intelligence-based approaches for the translation of structured languages. Lee et al. [3] demonstrate the feasibility of using neural models to translate programs across industrial robot languages. Likewise, Acharjee et al. [7] propose sequence-to-sequence models to translate pseudocode into source code, highlighting the potential of neural machine translation in structured domains.

Code translation has also been widely studied beyond the robotics domain. Yan et al. [12] present CodeTransOcean, a multilingual benchmark for evaluating code translation models, and Pan et al. [13] analyze errors introduced by language models in automatic translation. In addition, Feischl and Kern [14] study the functional quality of the generated code, and Macedo et al. [15] propose using intermediate representations to improve translation accuracy.

A fundamental aspect of these systems is evaluating translation quality. The BLEU metric, proposed by Papineni et al. [19], has been widely used to measure the similarity between generated sequences and reference outputs. Subsequently, Post [20] emphasized the importance of reporting this metric accurately in experimental studies.

However, in the context of structured programming languages, evaluation poses additional challenges. Partial similarity-based metrics, such as BLEU, may not fully capture the functional correctness of generated code. In this regard, stricter metrics such as Exact Match (EM) have been used in code translation tasks because they allow verification of exact correspondence between sequences, which is critical in applications where small syntactic variations may invalidate program execution [13,14].

Finally, sequence-to-sequence models based on recurrent neural networks, particularly LSTM-based models, have proven effective for modeling sequence dependencies and learning correspondences between languages [21,22,24–26]. Nevertheless, their application to industrial robot programming languages remains an active area of research, especially in scenarios requiring high syntactic and semantic precision.

Recently, different architectures such as T5, CodeT5, CodeBERT, StarCoder, and Llama variants have opened new lines of research and application for automatic code translation and generation. Through large-scale pretraining, these architectures have improved their ability to capture complex contextual dependencies. However, their use requires greater computational resources. For this reason, LSTM-based Seq2Seq models remain a feasible option for specialized tasks involving smaller datasets and limited computing capabilities [30–33].

Although studies have already addressed translation in industrial robot programming or the integration between robots, the techniques they have used include post-processors and rule-based functions. There is still a very broad field of study in the implementation of neural models for automatic translation between industrial programming languages. Likewise, there are few works that use quantitative evaluation metrics considering exact matching and structural similarity of the generated sequences.

This work proposes a Seq2Seq model based on LSTM networks for translation between the Fanuc and Kuka languages, using a dataset of 28,000 samples taken from example programs in manuals and industrial programs that were or are currently in operation in a robotic cell. The main contribution of this work is an approach that uses machine learning implementation for interoperability between robotic platforms while simultaneously performing a quantitative analysis of the model's performance using evaluation metrics.

2. Materials and Methods

The proposed system performs automatic translation between industrial robot programming languages through a sequence-to-sequence (Seq2Seq) neural architecture. This type of model has proven highly effective for sequence translation tasks, in both natural and structured languages [21,22,24,25].

The main objective of the system is to learn correspondences between instructions written in different robot programming languages using aligned data pairs. In this work, the translation process focuses on converting programs written in the FANUC language (FANUC, Oshino, Japan) into the KUKA KRL language (KUKA Robotics, Augsburg, Germany).

The system's general architecture is divided into two main phases: training and translation. In the training phase, a parallel dataset comprising equivalent instructions in the FANUC and KUKA languages is preprocessed, tokenized, and converted into numerical representations to train the LSTM-based Seq2Seq model. In the translation phase, a new program written in the FANUC language is processed through tokenization and subsequently fed into the previously trained model, which generates the equivalent program in the KUKA language as output. This general system workflow is illustrated in Figure 1.

Unlike traditional approaches based on deterministic rules or post-processors, the proposed system adopts a data-driven approach in which the model learns the relationships between instructions directly from aligned examples. This approach has been widely explored in automatic code translation tasks, demonstrating advantages in generalization and adaptability [3,12,13].

2.1. Dataset Construction

To train the translation model, a dataset of 28,000 aligned pairs between the FANUC and KUKA languages was constructed. Each sample contains semantically equivalent commands in both languages, enabling the model to learn direct correspondences between sequences. The equivalence between instruction pairs was verified through manual inspection based on programming semantics and industrial standards.

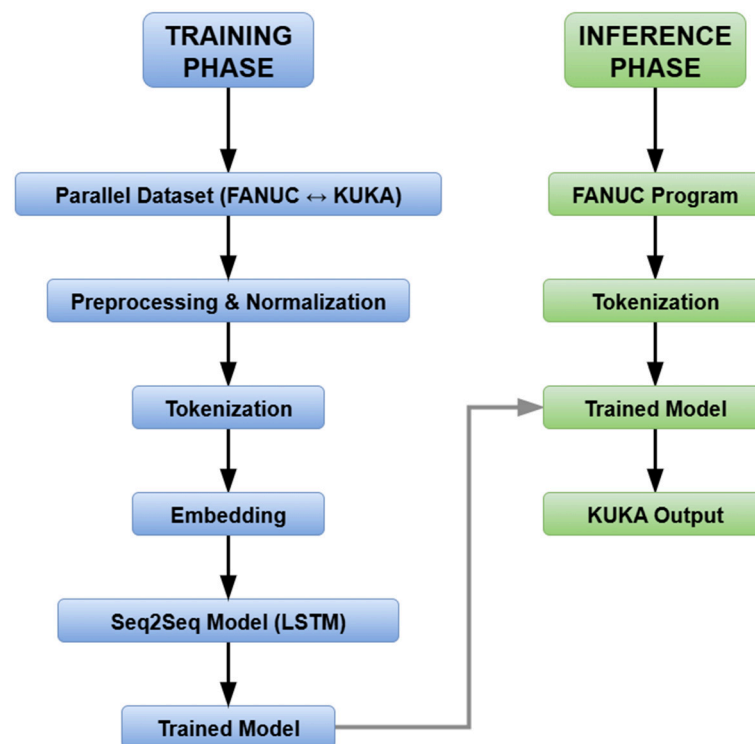


Figure 1. General workflow of the proposed learning-based translation model, including training and inference phases.

The data were obtained from real industrial robot programs and simulation-generated examples, which help ensure that the samples represent realistic operating conditions. Each dataset record includes the source-language command (FANUC), its target-language equivalent (KUKA), a unified token, and the corresponding functional category.

Table 1 shows that FANUCs can be aligned with their KUKAs while preserving the same operational intent, despite syntactic differences between the two languages. This strategy enabled the construction of a parallel dataset suitable for supervised training of the model.

Table 1. Examples of aligned robot instructions used in the training dataset (FANUC–KUKA).

Fanuc Instruction	KUKA Instruction	Unified Token	Category
J P[1] 300 mm/s FINE	PTP P1 Vel = 300 mm/s	JointMove	Motion
L P[1] 300 mm/s FINE	LIN P1 Vel = 300 mm/s	LinearMove	Motion
C P[1] P[2] 300 mm/s FINE	CIRC P1, P2 Vel = 300 mm/s	CircularMove	Motion

The instructions cover a wide variety of typical operations in industrial robot programming, including motion commands (PTP, LIN, CIRC), tool and base configurations, speed parameters, input/output control, and logical structures. To analyze the internal structure of the dataset, the distribution of instructions by unified token is presented in Table 2.

The dataset distribution shows a predominance of motion instructions, consistent with the nature of industrial robotic programs, in which trajectories constitute the main component [1,5]. Additionally, the distribution of instructions by functional category is presented in Table 3, allowing the diversity of commands used during training to be evaluated.

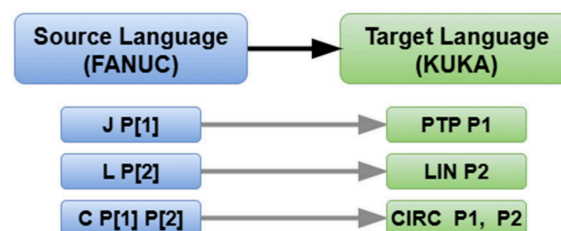
Table 2. Distribution of instructions by unified token.

Unified Token	Number of Samples
JointMove	4000
LinearMove	4000
CircularMove	4000
ToolConfig	2000
BaseConfig	2000
Velocity	2000
Wait	2000
DigitalOut	2000
ConditionalJump	2000
CallSub	1334
Timer	1334
OffsetMove	1332
Total	28,000

Table 3. Distribution of instructions by category.

Category	Number of Samples
Motion	12,000
Configuration	4000
Advanced	4000
Velocity	2000
Timer	2000
I/O	2000
Logic	2000
Total	28,000

The correspondence between equivalent instructions in the FANUC and KUKA languages is illustrated in Figure 2, which shows that different commands preserve the same functional meaning despite syntactic differences.

**Figure 2.** Example of equivalence between instructions in FANUC and KUKA languages.

The use of aligned parallel datasets has proven an effective strategy for automatic translation tasks because it allows the model to learn direct relationships between equivalent sequences [12,13].

2.2. Tokenization and Instruction Representation

Once the dataset has been constructed, the instructions are converted into a structured representation via tokenization. This process decomposes each instruction into a sequence of tokens, with each token representing a meaningful component of the command.

In the context of robot programming languages, these components include motion type, position identifiers, numerical parameters, speed units, and execution modifiers.

For example, a FANUC language instruction such as “J P[1] 300 mm/s FINE” can be decomposed into a sequence of tokens that represent its structural components. Likewise,

a KUKA language instruction such as “LIN P1 Vel = 300 mm/s” can be represented by tokens that preserve its functional meaning. The tokenization process for these instructions is illustrated in Figure 3, where each command is transformed into a structured token sequence suitable for processing by the model.

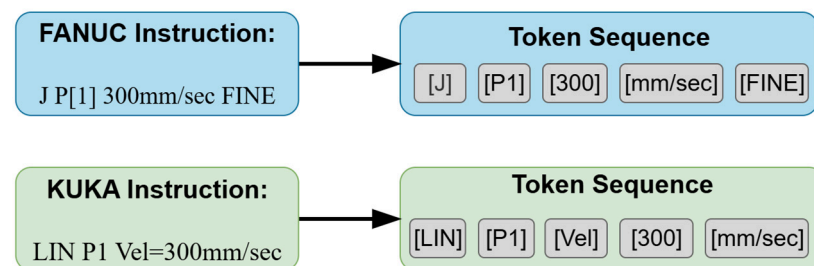


Figure 3. Example of tokenization of robot programming instructions, where each instruction is transformed into a structured token sequence for model processing.

Although both languages exhibit syntactic differences, tokenization enables the representation of instructions as structured sequences that preserve their functional meaning. This type of representation is widely used in automatic code translation tasks because it facilitates learning patterns across different languages [7,13]. Since robotic instructions are usually relatively short, a maximum sequence length of 30 tokens is defined. To ensure consistency during training, all sequences are normalized using padding, enabling efficient batch processing.

2.3. Numerical Representation and Seq2Seq Model

After tokenization, the sequences are transformed into numerical representations by assigning integer indices to each token. This process makes it possible to convert the instructions into a format suitable for deep learning models.

The translation system is implemented through a sequence-to-sequence (Seq2Seq) architecture based on Long Short-Term Memory (LSTM) networks, which have proven effective for modeling dependencies in variable-length sequences [21,24]. As shown in Figure 4, the model comprises an encoder that processes the input sequence and a decoder that generates the output sequence from a latent representation.

The model learns the conditional probability of an output sequence given an input sequence, which can be expressed as [21,22]:

$$P(y_1, y_2, \dots, y_T \mid x_1, x_2, \dots, x_T), \quad (1)$$

This probability can be factorized as [21,22]:

$$P(y|x) = \prod_{t=1}^T P(y_t | y_1, \dots, y_{t-1}, x), \quad (2)$$

This decomposition makes it possible to model the generation of each token as a function of the previous tokens, capturing sequential dependencies within the instruction [21].

LSTM networks are particularly suitable for this type of task due to their ability to capture long-term dependencies through gating mechanisms, which are essential for translating structured sequences [24].

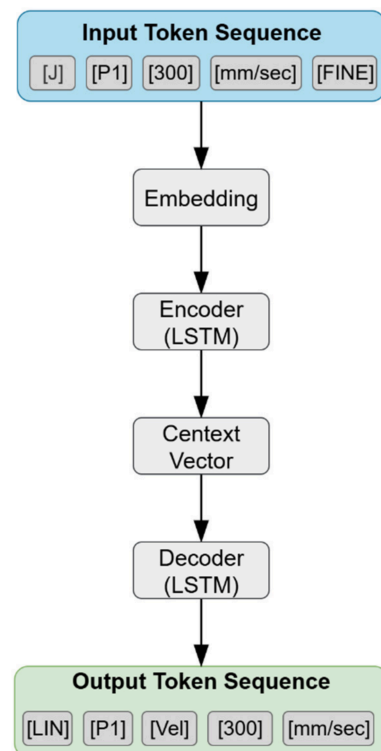


Figure 4. Sequence-to-sequence (Seq2Seq) architecture based on LSTM networks, illustrating the transformation of input token sequences into output token sequences through encoder–decoder processing.

2.4. General System Configuration

The model training process uses the previously described parallel dataset, which is divided into training, validation, and test subsets to evaluate the system’s generalization capability.

During training, the model learns to map input sequences in the source language to equivalent sequences in the target language by optimizing its internal parameters through an iterative process.

To achieve proper convergence, standard deep learning techniques are used to stabilize training and improve the model’s generalization. These configurations, including hyperparameters and optimization strategies, are described in detail in the following subsections.

2.5. Model Architecture

The proposed translation system is based on a Seq2Seq (Sequence-to-Sequence) model implemented using two LSTM layers, one for the encoder and one for the decoder. This architecture was selected for the translator because it has been shown in other reported works to be highly effective in long-term sequence modeling tasks and in translating between natural language texts with different grammatical structures [21,22,24,28].

The two layers used in the architecture are:

Encoder: processes the input sequence in the source language (FANUC) and transforms it into a latent representation that captures the relevant information in the instruction.

Decoder: generates the output sequence in the target language (KUKA) sequentially from this latent representation.

To improve data representation, an embedding layer is used to transform discrete tokens into dense vectors, thereby capturing semantic relationships among vocabulary elements [25]. The architecture uses a latent dimension of 128 units, providing a suitable balance between modeling capacity and computational complexity. In addition, regularization techniques such as dropout and recurrent dropout with a value of 0.2 are incorporated to reduce overfitting and improve the model’s generalization capability [34].

During training, the teacher-forcing technique is employed, in which the true sequence is used as input to the decoder at each time step. This strategy accelerates model convergence and improves training stability. To delimit sequences, the special tokens <START> and <END> are used, indicating the beginning and end of each instruction, respectively.

Each LSTM layer employs 128 hidden units, and the architecture design focuses on minimizing the risk of overfitting while maintaining high performance in the evaluation metrics employed. The entire architecture was configured to achieve a balance between learning capacity and computational cost.

For model optimization, training tests were performed with different numbers of epochs, from 50 to 200 with intervals of 50 between each one, seeking optimal convergence and performance. It was determined that with 200 epochs there was the best convergence in the model. Furthermore, the Adam optimizer was used for parameter updates during training since it has good computational efficiency, fast convergence and use in deep learning applications [21,22,24,28]. Finally, a batch size of 16 samples was used for training.

Within the dataset, different percentages were used for training, validation, and testing of the total dataset, 22,400 samples corresponding to 80% were used for training, and the remaining 20% was divided equally between testing and validation, resulting in 2800 samples per subset. The first subset was used for model learning, the validation subset to monitor the model's generalization capability during training, and the testing subset to evaluate its final performance on previously unseen data.

2.6. Data Representation and Preprocessing

The input and output sequences correspond to tokenized instructions from robotic programs, where source sequences (FANUC) are translated into target sequences (KUKA).

The tokenization process separates instructions into structured components, allowing each command to be represented as a sequence of meaningful tokens. This process is complemented by text normalization techniques that reduce unnecessary variations in instruction representation and improve dataset consistency.

The vocabulary is generated dynamically from the dataset, with an approximate size of 80–150 tokens. Each token is represented by an integer index, allowing efficient processing by the neural model.

To ensure batch processing, all sequences are padded to a maximum length of 30 tokens (post-padding). This approach preserves the sequential structure of the data without affecting token order.

Finally, special tokens are incorporated to indicate the beginning and end of each sequence, facilitating model learning during training.

2.7. Training Configuration

The dataset is divided into three subsets: training (80%), validation (10%), and test (10%), following standard machine learning practices to evaluate the generalization capability of the proposed model.

The model was initially trained for 50 epochs using a batch size of 16 samples. Based on the analysis of the training and validation metrics, increasing the number of epochs was found to improve convergence without significant evidence of overfitting, leading to the selection of 200 epochs as the optimal training configuration [35].

The Adam optimizer is employed with an approximate learning rate of 0.001, which has proven effective in optimization problems involving deep neural networks [25,36].

The loss function used is sparse categorical cross-entropy, suitable for sequence prediction problems where tokens are represented as integer indices. This function is defined as [23,25]:

$$L = -\sum_y^t \log(y_t), \quad (3)$$

where (y_t) is the probability that the model assigns to the correct token at time t .

During training, validation metrics are monitored to evaluate the model's generalization capability and detect potential overfitting.

To improve training stability and reduce overfitting, regularization techniques such as dropout and recurrent dropout are employed with a value of 0.2. Likewise, the teacher forcing technique is used, in which the real output is provided as decoder input at each time step, accelerating convergence and improving prediction quality.

The combination of these strategies enables training a model capable of learning complex translation patterns between industrial robot programming languages while maintaining a balance among accuracy, stability, and generalization.

2.8. Evaluation Metrics

Model performance is evaluated using four main metrics: Loss, Accuracy, BLEU (Bilingual Evaluation Understudy), and Exact Match (EM). These metrics enable analysis of various aspects of model behavior, including convergence during training, token-level precision, and the quality of generated sequences.

The metrics were selected because they provide complementary information about the model's performance. Loss and Accuracy, used in the training learning process, are combined with BLEU and Exact Match, which allow for the evaluation of two important aspects: structural similarity and exact match, respectively. This enables a more comprehensive evaluation of the proposed model and the translation of industrial robot languages.

The Loss metric measures model error during training, while accuracy indicates the proportion of correctly predicted tokens in the output sequence.

On the other hand, the BLEU metric is widely used in machine translation tasks because it evaluates the structural similarity between the generated and reference sequences using n-gram matches [19,20,37,38]. The BLEU score is defined as:

$$\text{BLEU} = \text{BP} \cdot \exp\left(\sum_{n=1}^N \omega_n \log p_n\right), \quad (4)$$

where (p_n) represents n-gram precision and (ω_n) are the weights associated with each n-gram level.

The brevity penalty factor is defined as [19,20]:

$$\text{BP} = \begin{cases} 1, & \text{if } c > r, \\ \exp^{(1-\frac{r}{c})}, & \text{if } c \leq r, \end{cases} \quad (5)$$

where (c) is the length of the generated sequence and (r) is the length of the reference sequence.

Additionally, the Exact Match (EM) metric measures the exact match between the generated and reference sequences [13,14].

$$\text{EM} = \frac{1}{N} \sum_{i=1}^N I(y_i^{\text{pred}} = y_i^{\text{true}}) \quad (6)$$

where $I(\cdot)$ is an indicator function that takes the value of 1 if the generated sequence exactly matches the reference, and 0 otherwise.

Unlike BLEU, which allows partial matches, EM is a strict metric that requires complete correspondence between sequences. In the context of industrial robot programming languages, this metric is particularly relevant because small syntactic variations can produce incorrect or non-executable programs.

Recent studies have emphasized that EM provides a more reliable indicator of exact sequence correctness in code translation tasks, as partial similarity metrics such as BLEU may overestimate model performance in the presence of minor syntactic deviations [18,39].

However, due to its strict nature, EM is highly sensitive to minor variations such as differences in spacing, capitalization, or parameter representation. For this reason, low EM values may be obtained even when the translation is structurally and syntactically correct.

Consequently, the combined use of BLEU and EM provides a more complete evaluation of the system, with BLEU capturing structural similarity and EM evaluating the exact correctness of generated sequences [40].

To mitigate this effect, a normalized version of EM can be considered, in which cleaning and standardization processes are applied to the sequences before comparison. This reduces sensitivity to superficial variations, providing an evaluation more closely aligned with the functional behavior of the generated code.

The combined use of BLEU, strict EM, and normalized EM enables evaluation of model performance from multiple perspectives, considering both structural similarity and the exact correctness of the generated sequences.

3. Results

This section presents the quantitative evaluation of the proposed model for the automatic translation of industrial robot language instructions. The analysis is carried out using standard metrics in sequence modeling tasks, including loss, accuracy, BLEU, and Exact Match (EM).

The evolution of model training is examined, along with its final performance on the training, validation, and test sets. In addition, a detailed analysis of model stability and the differences among the metrics used is presented to provide a comprehensive evaluation of the proposed system.

3.1. Training Evolution

The evolution of the model training process is analyzed using loss and accuracy curves across epochs, considering both the training and validation sets. These metrics enable evaluation of model convergence and generalization capability.

As shown in Figure 5, the loss function decreases progressively during training, reaching values close to zero on the training set. In the validation set, a similar trend is observed, albeit with slightly higher values, consistent with the expected behavior in deep learning models.

On the other hand, the accuracy metric shows a sustained increase throughout the epochs, reaching values close to 1.0 on the training set. In the validation set, accuracy stabilizes above 0.97, indicating the model's strong ability to generalize to unseen data. This behavior suggests that the model can effectively learn correspondences between the input and output sequences without significant overfitting. The difference observed between training and validation metrics remains within an acceptable range, indicating an adequate balance between fitting and generalization. Additionally, the stability of the curves during the final epochs suggests that the model has converged, with further training producing only marginal performance improvements.

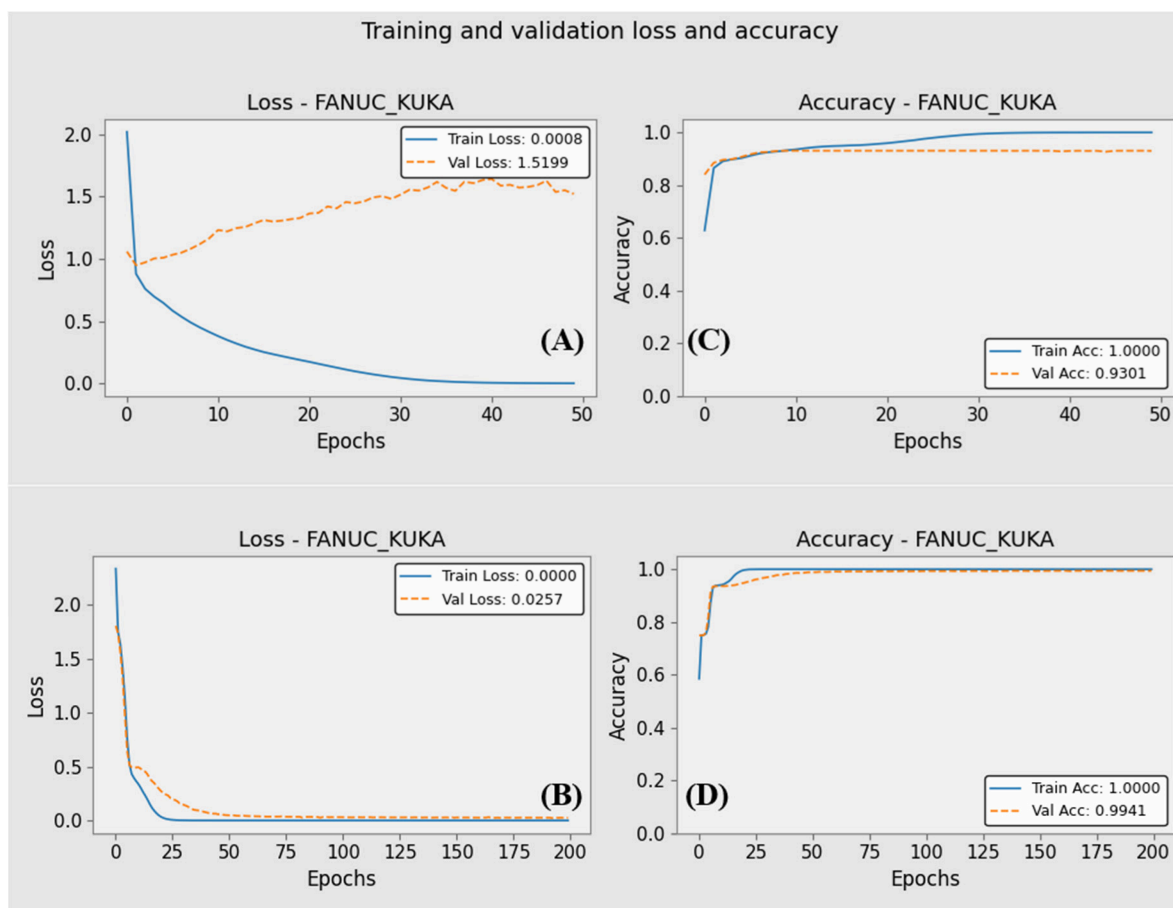


Figure 5. Training and validation metrics across epochs for two training configurations: (A) training and validation loss for 50 epochs, (B) training and validation loss for 200 epochs, (C) training and validation accuracy for 50 epochs, and (D) training and validation accuracy for 200 epochs.

The obtained results confirm that the training configuration enables stable convergence and effective generalization. As shown in Figure 5, the model trained for 200 epochs exhibits improved convergence and lower validation loss than the 50-epoch configuration.

3.2. Final Results

The model's final performance is evaluated on the test set to assess its generalization to unseen data during training. The results obtained for the proposed model under four training configurations (50, 100, 150 and 200 epochs) are presented in Table 4, which includes the loss, accuracy, BLEU, and Exact Match metrics for the test set. The 50-epoch configuration is used as a baseline for comparison.

Table 4. Comparison of model performance on the test dataset for four training configurations (50, 100, 150 and 200 epochs).

Metric	50 Epochs	100 Epochs	150 Epochs	200 Epochs
Accuracy	0.9301	0.9321	0.9901	0.9943
Loss	1.5199	1.4794	0.0369	0.0227
BLEU	0.388	0.387	0.669	0.682
Exact Match	0.752	0.759	0.944	0.970

On the test set, the 200-epoch configuration achieves an accuracy of 0.9943 and a loss value of 0.0227, indicating high precision in token prediction. Likewise, the model obtains a BLEU score of 0.682, reflecting high structural similarity between the generated

sequences and the references. Additionally, the model achieves an Exact Match (EM) of 0.970, indicating that 97% of the generated sequences fully match the reference translations. This metric is widely used in code translation tasks as a strict evaluation criterion to measure exact correctness of generated sequences, complementing similarity-based metrics such as BLEU [18,27,39,41–43].

These results demonstrate that the model effectively learns correspondences between industrial robot programming languages, achieving both high token-level precision and adequate structural similarity in the generated sequences. Overall, the results show that the proposed architecture is suitable for translating robotic instructions and achieves competitive performance on standard evaluation metrics. The structured and repetitive nature of robot programming instructions partially explains the high performance. This behavior is consistent with structured code translation tasks, where patterns are more regular compared to natural language.

As observed, increasing the number of training epochs leads to significant improvements across all evaluation metrics, particularly in Exact Match (EM), highlighting the importance of sufficient training for structured code translation tasks.

3.3. Comparison Across Datasets

The comparative analysis across the training, validation, and test sets enables evaluation of the model's generalization capability and detection of potential overfitting.

On the training set, the model achieves values close to 1.0 for both accuracy and loss reduction, indicating that it has effectively learned the correspondences in the dataset. However, this behavior alone is not sufficient to guarantee good performance on unseen data.

On the validation set, a slight decrease in accuracy is observed, with values close to 0.9941, and a moderate increase in the loss function. This behavior is consistent with expectations for deep learning models and suggests that the model maintains adequate generalization capability.

On the test set, the results are consistent with those obtained on the validation set, achieving an accuracy of 0.9943 and a BLEU score of 0.682. This similarity indicates that the model does not exhibit significant overfitting and can generalize well to new data.

It is important to note that although the accuracy metric approaches near-perfect values, the BLEU score is lower, as expected in code translation tasks. This difference arises because accuracy evaluates predictions at the level of individual tokens, whereas BLEU measures the structural similarity of the entire sequence.

In this sense, small variations in the generated sequence may not significantly affect accuracy but may affect the BLEU score. This highlights the importance of using multiple metrics to evaluate model performance from different perspectives. In particular, Exact Match (EM) has been widely adopted in structured code translation tasks as a strict metric for evaluating full-sequence correctness, while BLEU captures partial structural similarity [27,39,41].

The consistency between validation and test results, as well as the relationships among the metrics used, confirms that the model exhibits stable behavior and adequate generalization capability for the task of translating robotic instructions.

3.4. Performance Analysis

The combined analysis of BLEU and Exact Match (EM) metrics provides a more complete view of model performance in the task of translating robotic instructions [40].

The BLEU score on the test set is 0.682, indicating high structural similarity between the generated and reference sequences. This result suggests that the model can adequately capture the syntactic patterns found in industrial robot programming languages.

This difference between metrics is particularly relevant in the context of structured programming languages. For example, a generated instruction that differs only in one parameter or in token order may still be semantically valid, but it will be penalized by the EM metric.

In this sense, scenarios may be observed in which the model achieves high BLEU scores, indicating a correct sequence structure, while EM scores are lower due to minor differences in the exact representation of the instruction. As shown in Figure 6, the discrepancy between BLEU and Exact Match reveals the limitations of partial similarity-based metrics for evaluating structured code translation tasks. In addition, the 200-epoch configuration consistently achieves superior performance across all evaluation metrics compared to the 50-epoch model.

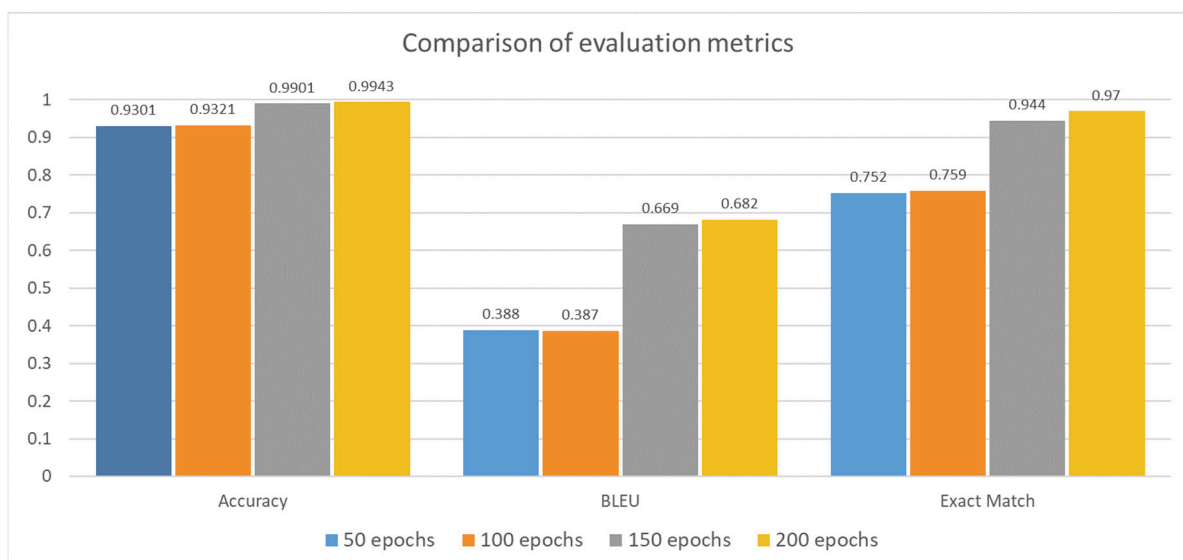


Figure 6. Comparison of test-set evaluation metrics obtained using different training epoch configurations, including Accuracy, BLEU, and Exact Match (EM).

This analysis demonstrates that the model is not only capable of generating syntactically correct sequences, but also of effectively approximating the expected behavior of industrial robot programs, which is real-world industrial applications.

Despite the strong performance, some limitations remain. Errors are mainly observed in parameter representation and token ordering, particularly in less frequent instruction patterns. These cases highlight the need for more diverse datasets and of the proposed model.

3.5. Statistical Analysis and Reproducibility

To ensure the model's reliability, stability and reproducibility were evaluated. Ten independent runs were performed using the same dataset and parameters, focusing on all 200 epochs. Accuracy, Loss, BLEU, Exact Match (EM), and training time were recorded for each run. The results of each run are shown in Table 5.

Subsequently, the mean and standard deviation were calculated for each of the metrics mentioned in the previous table. This was performed to analyze the variability of the results, which showed low variability across all evaluated metrics, indicating consistent and stable model behavior during the training and evaluation process. The mean and standard deviation results are shown in Table 6.

Table 5. Performance metrics obtained from ten independent training runs of the proposed model.

Value Test	Accuracy	Loss	BLEU	Exact Match	Training Time
1	0.99687475	0.011463702	0.694548833	0.983333333	02:36:49
2	0.993820429	0.028454605	0.684496156	0.967	02:23:48
3	0.996752799	0.012550828	0.696195813	0.982666667	02:20:08
4	0.993559718	0.026703293	0.679328715	0.965666667	02:04:58
5	0.996183336	0.016181936	0.689237431	0.979666667	02:07:46
6	0.997189879	0.010193966	0.698236296	0.985	02:21:17
7	0.997064173	0.011037312	0.696965551	0.984333333	02:13:31
8	0.998128414	0.009208709	0.704484438	0.99	02:22:14
9	0.994811416	0.02249917	0.685179506	0.972333333	02:25:01
10	0.995631278	0.015375367	0.692491084	0.976666667	02:17:23

Table 6. Mean and sample standard deviation of the performance metrics obtained from ten independent training runs.

Metric	Mean	Standard Deviation
Accuracy	0.996001619	±0.001516505
Loss	0.016366889	±0.007054776
BLEU	0.692116382	±0.007561867
Exact Match	0.978666667	±0.008087485
Training Time	02:19:18	±0:09:07

3.6. Qualitative Translation Example

To illustrate the practical behavior of the proposed model, a qualitative translation example is presented using a FANUC program generated in RoboGuide and its corresponding output in KUKA KRL format. This example demonstrates how the model translates structured robotic instructions while preserving their operational semantics.

The input program consists of motion instructions, including joint and linear movements, with associated parameters such as speed and termination conditions. The proposed translation system generates the corresponding KUKA code by adapting the structure and syntax of the original program to the target language.

In particular, the generated output follows the standard KUKA architecture, which separates the execution logic and data definition into two files: a main program (.SRC) and a data (.DAT) file. This separation contrasts with the FANUC format, where both instruction logic and positional data are typically defined within a single file.

Figure 7 shows the input FANUC program and the corresponding translated KUKA files generated by the proposed system.

The translation preserves key motion semantics, including the correspondence between joint and linear movements ($J \rightarrow PTP$, $L \rightarrow LIN$), demonstrating correct mapping of core instruction types.

For example, differences can be observed in the representation of motion termination and velocity parameters between the source and target programs.

Some discrepancies are observed in parameter representation, motion blending conventions, and controller-specific formatting, reflecting differences between FANUC and KUKA programming standards. For instance, termination conditions such as FINE in FANUC do not have a direct one-to-one equivalent in KUKA and are approximated through blending parameters. These differences do not alter the program's operational intent but highlight the challenges of achieving exact correspondence across heterogeneous industrial languages.

FANUC (.LS)		(A)
<pre> /PROG PRUEBA_FANUC_1 /ATTR OWNER = MNEDITOR; COMMENT = ""; PROG_SIZE = 697; CREATE = DATE 24-11-25 TIME 16:54:14; MODIFIED = DATE 24-11-25 TIME 18:01:52; FILE_NAME = ; VERSION = 1; LINE_COUNT = 5; MEMORY_SIZE = 1049; PROTECT = READ_WRITE; TCD: STACK_SIZE = 0, TASK_PRIORITY = 50, TIME_SLICE = 0, BUSY_LAMP_OFF = 0, ABORT_REQUEST = 0, PAUSE_REQUEST = 0; DEFAULT_GROUP = 1,*,*,*,*; CONTROL_CODE = 00000000 00000000; /MN 1 :J P[1] 100% FINE ; 2 :L P[2] 100mm/sec FINE ; 3 :L P[3] 100mm/sec FINE ; 4 :L P[4] 100mm/sec FINE ; 5 :L P[5] 100mm/sec FINE ; /POS P[1]{ </pre>		<pre> /POS P[1]{ GP1: UF : 0, UT : 1, CONFIG : 'N U T, 0, 0, 0', X = 1500.000 mm, Y = 0.000 mm, Z = 1140.000 mm, W = 0.000 deg, P = -90.000 deg, R = 0.000 deg }; P[2]{ GP1: UF : 0, UT : 1, CONFIG : 'N U T, 0, 0, 0', X = 1500.000 mm, Y = -640.000 mm, Z = 1140.000 mm, W = 0.000 deg, P = -90.000 deg, R = 0.000 deg }; P[3]{ GP1: UF : 0, UT : 1, CONFIG : 'N U T, 0, 0, 0', X = 1500.000 mm, Y = 0.000 mm, Z = 560.044 mm, W = 0.000 deg, P = -90.000 deg, R = 0.000 deg }; P[4]{ GP1: UF : 0, UT : 1, CONFIG : 'N U T, 0, 0, 0', X = 1500.000 mm, Y = 103.000 mm, Z = 560.044 mm, W = 0.000 deg, P = -90.000 deg, R = 0.000 deg }; P[5]{ GP1: UF : 0, UT : 1, CONFIG : 'N U T, 0, 0, 0', X = 1500.000 mm, Y = 0.000 mm, Z = 1140.000 mm, W = 178.409 deg, P = -80.709 deg, R = -178.507 deg }; /END </pre>
KUKA (.SRC)	KUKA (.DAT)	(C)
<pre> DEF kuka_translate() BAS (#INITMOV,0) PTP XP1 C_DIS LIN XP2 C_DIS LIN XP3 C_DIS LIN XP4 C_DIS LIN XP5 C_DIS END </pre>	<pre> DECL FDAT FDAT1={TOOL_NO 1, BASE_NO 0} DECL PDAT PDAT1={VEL 100, ACC 0} DECL POS XP1={X 1500.0, Y 0.0, Z 1140.0, A 0.0, B -90.0, C 0.0} DECL FDAT FDAT2={TOOL_NO 1, BASE_NO 0} DECL PDAT PDAT2={VEL 100, ACC 0} DECL POS XP2={X 1500.0, Y -640.0, Z 1140.0, A 0.0, B -90.0, C 0.0} DECL FDAT FDAT3={TOOL_NO 1, BASE_NO 0} DECL PDAT PDAT3={VEL 100, ACC 0} DECL POS XP3={X 1500.0, Y -640.0, Z 560.044, A 0.0, B -90.0, C 0.0} DECL FDAT FDAT4={TOOL_NO 1, BASE_NO 0} DECL PDAT PDAT4={VEL 100, ACC 0} DECL POS XP4={X 1500.0, Y 103.0, Z 560.044, A -0.0, B -90.0, C 0.0} DECL FDAT FDAT5={TOOL_NO 1, BASE_NO 0} DECL PDAT PDAT5={VEL 100, ACC 0} DECL POS XP5={X 1500.0, Y 0.0, Z 1140.0, A 178.409, B -80.709, C -178.507} </pre>	
(B)		

Figure 7. Qualitative example of translation from FANUC to KUKA programming languages using the proposed model: (A) input FANUC program (.LS) generated in RoboGuide, (B) translated KUKA execution file (.SRC), and (C) corresponding KUKA data file (.DAT), highlighting both structural correspondence and minor translation discrepancies.

This example is intended for qualitative illustration and complements the quantitative evaluation presented in previous sections. As shown in Figure 7, small variations in the translated instructions help explain why evaluation metrics such as BLEU and Exact Match do not reach perfect scores, even when the generated sequences preserve the overall structure and intent of the original program. The example demonstrates that the model is capable of generating structurally consistent and semantically aligned translations, while also highlighting areas where further refinement is required for full compliance with industrial controller specifications.

4. Discussion

4.1. Model Analysis

Recent advances in machine translation and code generation have seen significant progress thanks to architectures such as Transformers and Large Language Models like T5, CodeT5, CodeBERT, StarCoder, and some based on Llama. These architectures have yielded promising results in natural language processing, code generation, and generating correct structures in language translations [30–33].

For capturing large and complex amounts of data, Transformer-based models are particularly useful. CodeT5 and CodeBERT are primarily used for understanding and generating source code [30,33]. On the other hand, models like StarCoder and those based on Llama focus on multilingual coding and the tracing of instructions from natural language to machine language [31,33]. Recent studies show the application of these models to solve human–robot interaction problems and the interpretation of natural language instructions into robotic instructions [3,31–33,44,45].

Although these are significant advantages, the use of these architectures requires larger datasets compared to traditional Seq2Seq models. Furthermore, the computational resources and fine-tuning strategies are more complex in Transformer-based architectures [30–33,44]. In comparison, the LSTM-based Seq2Seq architecture proposed in this work achieved a good response using a moderate dataset and efficient computational resources, representing an alternative for translating industrial robot languages.

4.2. Limitations

The present work has limitations that should be considered. First, the architecture was trained and evaluated exclusively for the translation between the FANUC and KUKA industrial robot programming languages; therefore, other robot manufacturers were not considered in this initial study. Additionally, the dataset consisted of 28,000 instruction pairs obtained from example exercises, routines executed in RoboGuide, and robotic cell programs that were previously or are currently in use. Although an effort was made to include a broad range of programming scenarios, the dataset may still be limited in terms of the syntactic structures and programming situations represented during training.

The evaluation conducted in this work focused primarily on translation metrics, structural correspondence, and syntactic quality. Furthermore, the proposed system is capable of generating code files compatible with manufacturer-specific simulation environments; however, experimental validation using these simulators has not yet been performed. Finally, this work is limited to the analysis of an LSTM-based Seq2Seq architecture, and no experimental comparison was carried out with more recent Transformer-based architectures or Large Language Models.

4.3. Future Work

As future work, it is proposed to extend the model to support multiple industrial robot programming languages, including ABB (RAPID) and other proprietary environments, to improve translation across a broader range of programming languages. Additionally, the evaluation of the generated code in simulation and real execution environments, such as RoboGuide, WorkVisual, and RobotStudio, is proposed. This will make it possible to analyze not only syntactic quality, but also the execution correctness of the generated programs.

Furthermore, extend the approach to the translation of complete programs, which represents a significant challenge in the context of industrial robotic systems.

5. Conclusions

In this work, an artificial-intelligence-based approach was presented for automatically translating instructions in industrial robot programming languages. Using a sequence-to-sequence (Seq2Seq) architecture based on LSTM networks, it was possible to model correspondences between FANUC and KUKA language instructions with an aligned parallel dataset.

The results indicate that the model effectively learns relationships among structured sequences, achieving high accuracy on both the validation and test sets and competitive BLEU scores. Specifically, the model achieves 0.9943 accuracy, 0.682 BLEU, and 0.970 Exact Match on the test set. These results indicate that the model has an adequate capability to capture the syntactic structure of the analyzed programming languages.

Additionally, the use of the Exact Match (EM) metric enabled a stricter evaluation of the model, highlighting the difference between structural similarity and exact correspondence in the generated sequences. This analysis emphasizes the importance of using multiple metrics for code translation tasks, particularly in contexts where small variations can affect program execution.

Finally, this work and the proposed approach demonstrate that deep-learning-based models can be used effectively to improve interoperability among industrial robotic systems, reducing reliance on traditional rule-based methods and facilitating adaptation across different platforms.

6. Patents

A patent application resulting from the work reported in this manuscript has been submitted to the Mexican Institute of Industrial Property (IMPI) with application number [MX/a/2026/007546].

The source code developed for the implementation of the work reported in this manuscript is registered as a computer program with the National Institute of Copyright (INDAUTOR) under registration number 03-2026-012812371100-01.

Author Contributions: Conceptualization, N.M.-C., R.T.-M., E.I.T.-F., C.A.A.-A. and M.L.-L.; methodology, N.M.-C. and R.T.-M.; software, N.M.-C.; validation, N.M.-C., R.T.-M., E.I.T.-F., C.A.A.-A. and M.L.-L.; formal analysis, N.M.-C. and R.T.-M.; investigation N.M.-C., R.T.-M., E.I.T.-F., C.A.A.-A. and M.L.-L.; resources, N.M.-C., R.T.-M., E.I.T.-F., C.A.A.-A. and M.L.-L.; data curation, N.M.-C. and R.T.-M.; writing—original draft preparation, N.M.-C. and R.T.-M.; writing—review and editing, N.M.-C., R.T.-M., E.I.T.-F., C.A.A.-A. and M.L.-L.; visualization, N.M.-C., R.T.-M. and E.I.T.-F.; supervision, N.M.-C. and R.T.-M.; project administration, N.M.-C. and R.T.-M. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The original contributions presented in the study are included in the article, further inquiries can be directed to the corresponding author. A sample of the dataset used can be found in https://github.com/CENTLA/Dataset_Sample.git (accessed 19 July 2026).

Acknowledgments: SECIHTI (Secretaría de Ciencias Humanidades Tecnología e Innovación, Ciudad de México, México) for the scholarship, Nathaniel Morales-Centla CVU 1075434. During the preparation of this manuscript/study, the authors used Google Translate (Google LLC, Mountain View, CA, USA, cloud-based service), Grammarly (Grammarly Inc., San Francisco, CA, USA, 2026 version), and ChatGPT (OpenAI, version 4o) for the purposes of grammatical editing and text refinement during

the preparation of this manuscript after the translation from Spanish to English. The authors have reviewed and edited the output and take full responsibility for the content of this publication.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

AI	Artificial Intelligence
LSTM	Long Short-Term Memory
Seq2Seq	Sequence-to-Sequence
BLEU	Bilingual Evaluation Understudy
EM	Exact Match
LLM	Large Language Model
NMT	Neural Machine Translation
NLP	Natural Language Processing
OLP	Offline Programming

References

1. Bilancia, P.; Schmidt, J.; Raffaeli, R.; Peruzzini, M.; Pellicciari, M. An Overview of Industrial Robots Control and Programming Approaches. *Appl. Sci.* **2023**, *13*, 2582. [\[CrossRef\]](#)
2. Soori, M.; Arezoo, B.; Dastres, R. Artificial Intelligence, Machine Learning and Deep Learning in Advanced Robotics: A Review. *Cogn. Robot.* **2023**, *3*, 54–70. [\[CrossRef\]](#)
3. Syniawa, D.; Droste, L.; Kuhlenkötter, B. Semi-Automated Programming of Industrial Robotic Systems Using Large Language Models and Standardized Data Model. *Robotics* **2026**, *15*, 79. [\[CrossRef\]](#)
4. Delledonne, M.; Villagrossi, E.; Beschi, M. Optimizing Parameters of Robotic Task-Oriented Programming via a Multiphysics Simulation. In Proceedings of the 2023 IEEE 28th International Conference on Emerging Technologies and Factory Automation (ETFA), Sinaia, Romania, 12–15 September 2023; pp. 1–4. [\[CrossRef\]](#)
5. Tamizi, M.G.; Yaghoubi, M.; Najjaran, H. A Review of Recent Trends in Motion Planning of Industrial Robots. *Int. J. Intell. Robot. Appl.* **2023**, *7*, 253–274. [\[CrossRef\]](#)
6. Hu, H.; Chen, J.; Liu, H. Natural Language-Based Automatic Programming for Industrial Robots. *J. Grid Comput.* **2022**, *20*, 26. [\[CrossRef\]](#)
7. Acharjee, U.K.; Arefin, M.; Hossen, K.M. Sequence-to-Sequence Learning-Based Conversion of Pseudo-Code to Source Code Using Neural Translation Approach. *IEEE Access* **2022**, *10*, 26730–26742. [\[CrossRef\]](#)
8. Souza, J.P.; Castro, A.; Rocha, L.; Relvas, P.; Silva, M.F. Converting Robot Offline Programs to Native Code Using the AdaptPack Studio Translators. In Proceedings of the IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC), Porto, Portugal, 24–26 April 2019; pp. 1–6. [\[CrossRef\]](#)
9. Matsumoto, T.; Hayashi, E.; Chumkamon, S.; Tominaga, A.; Mowshowitz, A. Research on AR System for Industrial Robot Introduction. In Proceedings of the International Conference on Artificial Life and Robotics (ICAROB 2023), Online, 9–12 February 2023; Volume 28, pp. 323–326. [\[CrossRef\]](#)
10. Nagata, F.; Okada, Y.; Otsuka, A.; Ikeda, T.; Kusano, T.; Watanabe, K. Post Processor for Industrial Robots—Circular Arc Interpolation of CLS Data to Generate FANUC Robotic Program. In Proceedings of the 2017 IEEE International Conference on Mechatronics and Automation (ICMA), Takamatsu, Japan, 6–9 August 2017; pp. 519–524. [\[CrossRef\]](#)
11. Bottazzi, V.S.; Fonseca, J.F.C. Off-Line Robot Programming Framework. In Proceedings of the Joint International Conference on Autonomic and Autonomous Systems and International Conference on Networking and Services (ICAS-ICNS'05), Papeete, France, 23–28 October 2005; p. 71. [\[CrossRef\]](#)
12. Yan, W.; Tian, Y.; Li, Y.; Chen, Q.; Wang, W. CodeTransOcean: A Comprehensive Multilingual Benchmark for Code Translation. In Proceedings of the Association for Computational Linguistics: EMNLP 2023, Singapore, 6–10 December 2023; pp. 5067–5089. [\[CrossRef\]](#)
13. Pan, R.; Ibrahimzada, A.R.; Krishna, R.; Sankar, D.; Wassi, L.P.; Merler, M.; Sobolev, B.; Pavuluri, R.; Sinha, S.; Jabbarvand, R. Lost in Translation: A Study of Bugs Introduced by Large Language Models while Translating Code. In Proceedings of the IEEE/ACM 46th International Conference on Software Engineering (ICSE '24), Lisbon, Portugal, 14–20 April 2024; pp. 1–13. [\[CrossRef\]](#)
14. Feischl, C.; Kern, R. Large Language Models for Code Translation: An In-Depth Analysis of Code Smells and Functional Correctness. *ACM Trans. Softw. Eng. Methodol.* **2025**, *35*, 1–40. [\[CrossRef\]](#)

15. Macedo, M.; Tian, Y.; Nie, P.; Cogo, F.R.; Adams, B. INTERTRANS: Leveraging Transitive Intermediate Translations to Enhance LLM-Based Code Translation. In Proceedings of the IEEE/ACM 47th International Conference on Software Engineering (ICSE), Ottawa, ON, Canada, 27 April–3 May 2025; pp. 1153–1164. [\[CrossRef\]](#)
16. Deepak Naik, M.V.; Swaminathan, J. Advancing Code Translation with Context-Aware Pre-Training in Data-Scarce Environments. *IEEE Access* **2026**, *14*, 16717–16730. [\[CrossRef\]](#)
17. Suárez Bonilla, F.; Ruiz Ugalde, F. Automatic Translation of Spanish Natural Language Commands to Control Robot Commands Based on LSTM Neural Network. In Proceedings of the 2019 Third IEEE International Conference on Robotic Computing (IRC), Naples, Italy, 25–27 February 2019; pp. 125–131. [\[CrossRef\]](#)
18. Shi, F.; Wang, Y.; Chen, H. Execution-Based Evaluation for Natural Language to Code Generation. In Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing (EMNLP), Abu Dhabi, United Arab Emirates, 7–11 December 2022; pp. 231–245. Available online: <https://aclanthology.org/2022.emnlp-main.231> (accessed on 12 December 2025).
19. Papineni, K.; Roukos, S.; Ward, T.; Zhu, W. BLEU: A Method for Automatic Evaluation of Machine Translation. In Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL), Philadelphia, PA, USA, 7–12 July 2002; pp. 311–318. [\[CrossRef\]](#)
20. Post, M. A Call for Clarity in Reporting BLEU Scores. In Proceedings of the Third Conference on Machine Translation (WMT), Brussels, Belgium, 31 October–1 November 2018; pp. 186–191. [\[CrossRef\]](#)
21. Sutskever, I.; Vinyals, O.; Le, Q.V. Sequence to Sequence Learning with Neural Networks. In Proceedings of the Advances in Neural Information Processing Systems (NeurIPS), Montreal, QC, Canada, 8–13 December 2014; Volume 27, pp. 3104–3112. [\[CrossRef\]](#)
22. Maru, G.G.; Lee, S.; Ji, S.; Ko, S.-K.; Im, H. Neural Methods for Programming: A Comprehensive Survey and Future Directions. *Appl. Sci.* **2025**, *15*, 12150. [\[CrossRef\]](#)
23. Goodfellow, I.; Bengio, Y.; Courville, A. *Deep Learning*; MIT Press: Cambridge, MA, USA, 2016.
24. Hochreiter, S.; Schmidhuber, J. Long Short-Term Memory. *Neural Comput.* **1997**, *9*, 1735–1780. [\[CrossRef\]](#) [\[PubMed\]](#)
25. Géron, A. *Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow*, 2nd ed.; O'Reilly Media: Sebastopol, CA, USA, 2019.
26. Bishop, C.M. *Pattern Recognition and Machine Learning*; Springer: New York, NY, USA, 2006.
27. Evtikhiev, M.; Bogomolov, E.; Sokolov, Y.; Bryksin, T. Out of the BLEU: How Should We Assess Quality of the Code Generation Models? *J. Syst. Softw.* **2023**, *203*, 111741. [\[CrossRef\]](#)
28. Liu, F.; Li, G.; Zhao, Q. Learning to Represent Code Semantics. *Sci. China Inf. Sci.* **2025**, *68*, 172101. [\[CrossRef\]](#)
29. Sooda, G.; Hariharan, A.; Patil, A.; Vinod, A.; Chandana, S.; Manvith, L.B.; Mandahas, N.; Siri, H.; Shobha, G.; Deepamala, N.; et al. Bridging Language Barriers in Coding: An NLP-Powered Tool for Programming Education in Regional Languages. *IEEE Access* **2025**, *13*, 119577–119589. [\[CrossRef\]](#)
30. Hamed, A.U.; Hasoon, S.O. Generative Artificial Intelligence for Code Generation Based on CodeT5 Algorithm. In Proceedings of the 2025 3rd International Conference on Cyber Resilience (ICCR), Baghdad, Iraq, 26–27 February 2025; pp. 1–5. [\[CrossRef\]](#)
31. Arefin, M.; Tran, D.; He, H. Complex Instruction Translation Using Fine-Tuned Large Language Models. In *Social Robotics; Lecture Notes in Computer Science*; Springer Nature: Singapore, 2025; Volume 15372, pp. 312–322. [\[CrossRef\]](#)
32. Fan, H.; Liu, X.; Fuh, J.Y.H.; Lu, W.F.; Li, B. Embodied Intelligence in Manufacturing: Leveraging Large Language Models for Autonomous Industrial Robotics. *J. Intell. Manuf.* **2025**, *36*, 1141–1157. [\[CrossRef\]](#)
33. Hussain, S.; Biswas, S.; Dutta, A.; Saad, M.; Baimagambetov, A.; Saeed, K.; Polatidis, N. A Review of Advances in Large Language and Vision Models for Robotic Manipulation: Techniques, Integrations, and Challenges. *SN Comput. Sci.* **2025**, *6*, 588. [\[CrossRef\]](#)
34. Zhao, S.; Tian, J.; Fu, J.; Chen, J.; Wen, J. FeaMix: Feature Mix with Memory Batch Based on Self-Consistency Learning for Code Generation and Code Translation. *IEEE Trans. Emerg. Top. Comput. Intell.* **2025**, *9*, 192–201. [\[CrossRef\]](#)
35. Zhang, H.; Zhang, K.; Li, Z. Deep Learning for Code Generation: A Survey. *Sci. China Inf. Sci.* **2024**, *67*, 191101. [\[CrossRef\]](#)
36. Gülmez, B. Code Generation With Large Language Models: A Survey from Neural Program Synthesis to Autonomous Software Development. *Appl. Intell.* **2026**, *56*, 200. [\[CrossRef\]](#)
37. Ren, S.; Guo, D.; Lu, S. CodeBLEU: A Method for Automatic Evaluation of Code Synthesis. *arXiv* **2022**, arXiv:2009.10297. [\[CrossRef\]](#)
38. Shin, J.; Hemmati, H.; Wei, M.; Wang, S. Assessing Evaluation Metrics for Neural Test Oracle Generation. *IEEE Trans. Softw. Eng.* **2024**, *50*, 2337–2349. [\[CrossRef\]](#)
39. He, J.; Liu, Q.; Wang, S. ExeCoder: Execution-Based Code Generation and Translation Evaluation. In Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing (EMNLP), Suzhou, China, 5–9 November 2025; pp. 362–375. Available online: <https://aclanthology.org/2025.emnlp-main.362> (accessed on 3 February 2026).
40. Esmaeili, A.; Saberi, I.; Fard, F. Empirical Studies of Parameter Efficient Methods for Large Language Models of Code and Knowledge Transfer to R. *Empir. Softw. Eng.* **2026**, *31*, 30. [\[CrossRef\]](#)
41. Liguori, P.; Improta, C.; Natella, R.; Cukic, B.; Cotroneo, D. Who Evaluates the Evaluators? On Automatic Metrics for Assessing AI-Based Offensive Code Generators. *Expert Syst. Appl.* **2023**, *225*, 120073. [\[CrossRef\]](#)

42. Liang, Y.; Zhang, X.; Li, Z. Addressing the Entity Translation Problem in Neural Machine Translation. In Proceedings of the Association for Computational Linguistics: ACL 2024, Bangkok, Thailand, 11–16 August 2024; pp. 691–705. Available online: <https://aclanthology.org/2024.findings-acl.691> (accessed on 30 November 2025).
43. McNabb, L. LLM-Augmented Machine Translation for E-Commerce Search Queries. In Proceedings of the CEUR Workshop Proceedings, Aachen, Germany, 1 June 2025; Volume 4123, pp. 18–27. Available online: https://ceur-ws.org/Vol-4123/paper_18.pdf (accessed on 3 March 2026).
44. Henzel, M.; Böggemann, M.; Witte, E.; Daut, F. Enabling Robot-Assisted Production Through Natural Language Input. In Proceedings of the 2025 7th Experiment@ International Conference (Exp.at'25), Funchal, Portugal, 2–4 June 2025; pp. 121–125. [CrossRef]
45. Lykov, A.; Cabrera, M.A.; Konenkov, M.; Serpiva, V.; Gbagbe, K.F.; Alabbas, A.; Fedoseev, A.; Moreno, L.; Khan, M.H.; Guo, Z.; et al. Industry 6.0: New Generation of Industry Driven by Generative AI and Swarm of Heterogeneous Robots. In Proceedings of the 2025 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Hangzhou, China, 19–23 October 2025; pp. 19992–19997. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.