Robotics and Computer–Integrated Manufacturing 101 (2026) 103289

Contents lists available at ScienceDirect

Robotics and Computer-Integrated Manufacturing

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

Full length article

A machine learning–based tool for enhancing position accuracy in industrial robots with a reduced dataset

Giuseppe Romano [a], Pietro Bilancia [a,*], Alberto Locatelli [a], Mirko Mucciarini [b], Manuel Iori [a], Marcello Pellicciari [a]

[a] Department of Sciences and Methods for Engineering, University of Modena and Reggio Emilia, Reggio Emilia, Italy
[b] Department of Engineering "Enzo Ferrari", University of Modena and Reggio Emilia, Modena, Italy

ARTICLE INFO

Keywords:
Industrial robot
Engineering tool
Robot simulation
Position accuracy
Machine learning
Offline compensation

ABSTRACT

Industry X.0 robotic manufacturing demands higher accuracy and flexibility, enabling continuously adaptive processes designed and optimized through simulations and Digital Twins. To achieve this level of flexibility and productivity in high value-added processes, where limited robot position accuracy becomes a critical constraint, advanced engineering methods and digital tools are required. These solutions must predictively compensate for inevitable robot positional accuracy errors, eliminating the need for manual pose refinement and enabling the generation of "first-time-right" robot code. This work aims to address these challenges by introducing an engineering tool capable of predictively correcting robot positioning inaccuracies across the workspace, enabling accurate point-to-point motion generation. It is intended for tasks with limited process interaction forces, where positioning errors are dominated by geometric, compliance, and joint-related effects. The tool leverages a multi-parameter Machine Learning (ML) error predictor trained on a reduced experimental dataset, minimizing data acquisition time and production downtime. Realized as a Python-based framework, it can be seamlessly integrated into commercial offline programming environments to automatically generate validated robot programs. The paper details the framework structure, focusing on the definition of the ML-based position error predictor, and its implementation on a robotic cell equipped with a high-payload KUKA robot and a FARO laser tracker. A preliminary experimental analysis identified payload, approach direction, and point location as the key operational parameters, accessible at the code level, that influence positioning accuracy. These insights guided feature selection and the design of reduced training datasets. In particular, a uniform spatial grid of only 64 points, corresponding to about one hour of measurement time, was sufficient to achieve near-optimal model accuracy. Several ML algorithms were compared, with the Tabular Prior-data Fitted Network achieving superior generalization on small datasets. Experimental validation on the KUKA robot showed up to a 98.4 % reduction in positioning error and consistent performance across all tested points, confirming the tool robustness and suitability for deployment across different industrial environments. All datasets, source code, and implementation scripts are openly released to enable reproducibility and facilitate industrial deployment.

1. Introduction

The deployment of Industrial Robots (IR) in modern manufacturing environments has rapidly expanded from traditional automation tasks like machine tending and palletizing to precision-critical tasks such as material processing and assembly [1–3], providing a drastic costs reduction and improving overall productivity and quality. This trend is mainly driven by the increasing demand for reconfigurable automated plants capable of handling variable parts and customized processes with limited setup time [4]. In this context, IR are strategic, offering high adaptability and rapid reconfigurability at low application cost with short payback times. However, their relatively low intrinsic motion and positional accuracy remains a limiting factor in many applications [5], often requiring final manual adjustments when changing processes, which significantly reduces overall productivity and manufacturing efficiency. Furthermore, even small spatial deviations with respect to the planned motion can lead to energy and material waste, rework, assembly failures, or degraded product quality [6].

In precision-driven production tasks, IR are required to perform with either pose or path accuracy, namely the ability to respectively reach a set of discrete targets or follow a continuous three-dimensional path with minimal error within their workspace [7]. Pose accuracy

* Corresponding author.
E-mail address: pietro.bilancia@unimore.it (P. Bilancia).

https://doi.org/10.1016/j.rcim.2026.103289
Received 25 August 2025; Received in revised form 16 January 2026; Accepted 27 February 2026

becomes essential when the robot must reliably reach predefined coordinates, as in precision assembly and insertion tasks [8], pick-and-place operations with tight tolerances [9], spot welding [10], and drilling [11]. In contrast, path accuracy plays a critical role in processes where the exact shape of the toolpath determines the quality of the result, such as robotic machining [12], additive manufacturing [13], laser surface treatments [14], or material dispensing [15]. Both types of accuracy degrade due to common causes, namely joint-related transmission errors, link deformation, and thermal drift, although path accuracy is further affected by the controller characteristics and the strategies adopted during motion planning.

In the literature, existing robot accuracy error compensation approaches are generally categorized as online or offline. Online strategies rely on real-time sensor feedback and advanced control loops that directly modify the executed trajectory [16]. They have been widely demonstrated in controlled laboratory environments, showing effective performance in correcting linear [17], circular [18] and contouring [19] paths, although their industrial applicability remains limited. This limitation primarily arises from the need to permanently integrate additional sensory apparatus (mainly laser trackers [16] or vision systems [20]) and to deploy hardware-dependent control architectures that are costly to maintain. More critically, in order to guarantee adequate sensor visibility during robotic operations, the layout of the cell often needs to be modified, which not only increases the overall footprint of the workstation but also necessitates adjustments to the process flow. These modifications imply further costs and reduced robustness, as the additional sensors are expensive, require delicate calibration, and are prone to malfunction when exposed to industrial dirt, chips, or coolant mists.

Offline approaches, in contrast, exploit behavioral models identified during a dedicated calibration phase [5]. Once validated, such models enable direct prediction and correction of end-effector positions without continuous sensor feedback or modification of the robot's control infrastructure [21,22]. These approaches are advantageous in industrial settings, where minimizing hardware complexity, facilitating deployment across diverse robot brands and control architectures, and improving overall plant efficiency by reducing downtime are key operational priorities.

From a methodological standpoint, the developed models can be broadly categorized as physics-based [23–25] or data-driven [26], depending on whether they provide explicit formulations of the robot's behavior or rely on empirical mappings learned directly from experimental data. The former offer high interpretability and strong extrapolation capability but require detailed parameter identification, as they combine kinematic models with geometric, stiffness, and joint-related transmission error terms evaluated through specific measurements [27]. Data-driven approaches, instead, learn empirical input–output relationships directly from experimental datasets, eliminating the need for explicit modeling but demanding a larger amount of measurements and offering less physical transparency [28]. They are generally more practical and broadly applicable, as they do not require advanced modeling expertise, which is not always available in industrial environments. However, their main limitation lies in the large volume of data needed to achieve the required prediction quality, which can become a significant constraint in production settings. Overall, despite their great potential, offline approaches have not yet reached the level of maturity required for large-scale industrial adoption, where Point-to-Point (PTP) motions are extensively used in various tasks and would greatly benefit from such advancements.

Previous research on data-driven modeling has laid a solid foundation and successfully demonstrated the integration of these models into compensation frameworks (see, e.g., the results in Refs. [21,22,28–31]), yet several key challenges remain open in the literature. In particular, one major concern is that model selection is often arbitrary: many studies adopt a specific Machine Learning (ML) algorithm from a library without justifying its suitability or providing evidence on training efficiency, sensitivity to dataset size, or other factors that would support an informed choice [32]. Indeed, the cited works mostly rely on neural networks [33], albeit in varied forms, from shallow feedforward to deep belief networks and specialized architectures like single hidden layer and radial basis functions. While this confirms the popularity and versatility of neural models, it also highlights the scarcity of comparative studies exploring fundamentally different regression paradigms [34]. Moreover, these models typically require large, diverse training datasets to achieve good generalization which, as said, can make data acquisition costly and time-consuming in industrial settings [35]. Another shortcoming is that experimental setups in related studies often lack sufficient detail on hardware and software implementation, hindering reproducibility and industrial deployment. Specifically, while conditions vary across applications and environments and replicating existing scenarios may not always be possible, reporting key experimental details remains essential. Information on the cell setup (e.g., installation of measurement devices, their settings, test procedures, and adopted control and sampling configurations) and on dataset collection parameters (number of training points, sampling directions, velocity profiles, and payloads) is crucial for defining the domain of validity and enabling meaningful comparison across studies. Furthermore, offering justifications and practical guidelines for these design choices would support informed decision-making and assist less experienced operators, particularly in industrial environments.

Beyond academic research, commercial software platforms such as RoboDK offer valuable, ready-to-use offline data-driven error-correction utilities, although these typically require paid licenses. In addition, as they connect directly to both the robot controller and the measuring system, their use is restricted to specific brands for which dedicated software interfaces have been developed [36]. These tools also provide limited transparency and control over the model training, as most assume a fixed number of reference points and offer little adaptability to application-specific requirements, such as workspace constraints or varying accuracy tolerances.

Building on the previous considerations, this work introduces a general-purpose, Python-based engineering tool aimed at improving IR position accuracy during PTP movements. The target use cases are pick-and-place and inspection tasks, as well as emerging applications like dispensing and laser processing, in which external process forces are negligible and robot errors are primarily driven by geometric, compliance, and joint-related effects. The tool implements a brand-agnostic framework applicable to any robot equipped with standard motion instructions and programming workflows. The framework relies on a ML error predictor and operates by modifying the original robot scripts. Therefore, compatibility across different robot brands is achieved by collecting a robot-specific experimental dataset for ML training and by adapting only the Python layer responsible for parsing and rewriting PTP motion instructions in each robot native programming language. Particular attention is given to the selection of the most suitable ML model for reliable and efficient prediction and to minimizing the size of the required training dataset, thereby reducing the impact on production. The approach is validated on a research cell featuring a high-payload KUKA IR and a FARO laser tracker capable of providing position measurements only. Nonetheless, the method can be readily extended to include both position and orientation tracking using alternative equipment, such as other tracking systems or vision-based solutions, without altering the procedural workflow. The core contributions of this paper are as follows:

1. **A modular and open-source ML-based framework** for position error compensation, adaptable to multiple robot platforms and metrology systems. The related Python tool is designed for rapid industrial deployment, requiring small datasets and no real-time feedback during production, making it ideal for offline

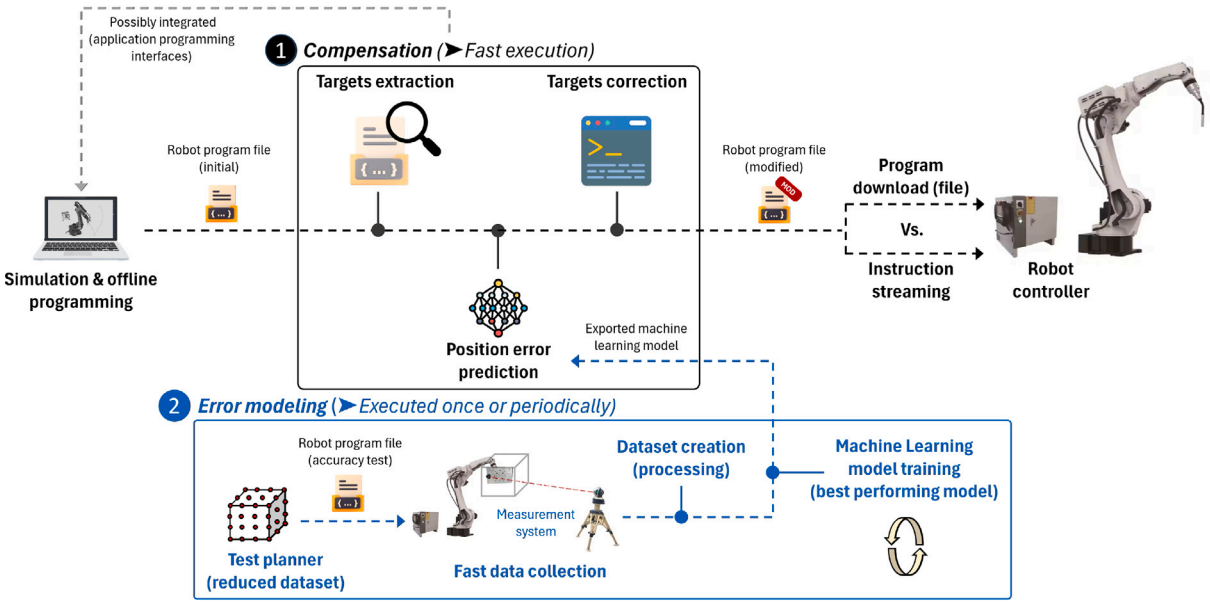

Fig. 1. Schematic of the IR position compensation tool, structured into modeling and correction modules.

integration within standard automation pipelines. All methodological details and implementation procedures are thoroughly documented to ensure full transparency and reproducibility.

2. **A parametric sensitivity analysis** of position accuracy as a function of operational parameters (speed, payload, approach direction) in line with ISO 9283 [7]. The analysis does not delve into error-source modeling at the core mechanical level, discussing effects such as backlash, compliance, or thermal drift, which have already been extensively addressed in prior academic studies (see Refs. [23,27,37,38]), but rather focuses on the influence of parameters that can be directly specified and modified within the robot codes, thereby providing a practical basis for selecting ML features prior to the subsequent training phase.
3. **A comparative benchmark study** of ML models, comprising both conventional regression algorithms and a recently released foundation model, namely the Tabular Prior-data Fitted Network (TabPFN) [39]. The latter model is specifically designed for small- to medium-sized tabular datasets and is pre-trained on millions of synthetic samples, allowing it to perform regression effectively without requiring hyperparameter tuning. As a result, TabPFN is capable of achieving high predictive accuracy even with limited training data, making it particularly suitable for industrial scenarios in which data acquisition is costly or constrained by operational limitations [40]. Performance metrics that guide dataset design and ML model selection according to application requirements and computational resources are provided and discussed.
4. **A publicly available open-source repository**, containing all the Python implementation scripts, collected datasets and practical guidelines to facilitate method reuse and further development by the research and industrial community.

The remainder of the paper is organized as follows. Section 2 presents an overview of the proposed compensation method and the associated Python tool. Section 3 describes the experimental setup along with the procedures adopted for data acquisition. Section 4 outlines the training process of the ML models and compares the performance of various algorithms to inform model selection. Section 5 reports the results of the experimental study conducted to validate the effectiveness of the proposed framework. Finally, Section 6 synthesizes the main findings and provides the concluding remarks.

2. Compensation method and tool

The proposed framework focuses on modeling and correcting systematic positioning errors in IR during the execution of PTP motions through a purely offline, non-intrusive approach. As many offline approaches, it does not require any real-time sensor feedback, access to internal signals such as motor currents, or modifications to the robot's hardware or control architecture. This ensures full compatibility with commercial robotic systems and facilitates seamless deployment in real-world industrial settings. Its general architecture, shown in Fig. 1, can potentially be applied to any IR type and brand as it operates directly on standard PTP motion instructions and target-based programming structures that are shared across all major manufacturers (e.g., KUKA PTP, ABB and Staubli `MoveJ`, FANUC J, etc.). Moreover, most IR share comparable mechanical architectures and rely on similar transmission elements (e.g., Rotating Vector reducers [41]) in their joints, leading to qualitatively consistent joint-related mechanical error sources and overall position accuracy performance maps across brands [42]. The framework, therefore, maintains a brand-agnostic structure, generating corrected robot programs prior to execution without affecting production scheduling [4,43]. At the same time, the adopted error model is specific to the individual robot unit, and deploying the framework on a different robot inevitably requires collecting new data and training a new model. The tool is organized into a modular pipeline that automates the entire process from data acquisition to deployment of corrected robot programs through two core modules: (1) a compensation module intended for end users to deliver error-corrected robot PTP instructions (specifically, target positions); (2) an error modeling module used during the initial training phase. The first module operates by processing the original robot program file typically generated during offline programming, extracting all motion targets, and producing corrected versions by adopting a pre-trained ML model that serves as an error predictor. The corrections occur offline on a standard PC and require only few seconds even for programs with hundreds of targets. The modified robot program, which preserves its original structure (i.e. vendor-specific layout and user preferences), can then be directly deployed on the real controller without delays.

For error modeling, the process relies on external measurements to capture deviations between commanded and actual robot positions. Depending on the adopted measurement system (e.g., a laser tracker [17] with or without orientation probes, or a camera-based setup [44]),

either position-only data (as in this work) or full pose data (position and orientation) can be acquired. These measurements are processed into a structured dataset of input–output pairs, where inputs include robot target coordinates (x, y, z, but also orientations if measured) and operating parameters (such as payload, motion direction, and speed), and outputs are the corresponding position (or full pose) errors. It shall be noted that to ensure practical applicability the analysis focuses specifically on quantities that are accessible and editable during robot programming and execution (target coordinates, speed settings, and tool data). To account for seasonal variations in ambient temperature, model training could be repeated at different times of the year. On the other hand, long-term degradation effects are inherently more difficult to capture due to the limited availability of long-term operational data, particularly in industrial settings, although recent studies on adaptive predictor algorithms suggest promising directions for future developments [45]. The outcome of this module is an ML model that is subsequently used by the compensator for error prediction and robot code modification. All the steps required to produce this error model, from the creation of a feasibility-checked test matrix, to the definition of robust experimental procedures and the comparison of ML methods, are rigorously automated through a set of parametric Python scripts. The following sections provide a detailed explanation of each step, with insights into their practical implementation.

Overall, the developed Python-based solution can be seamlessly integrated into commercial platforms (e.g., RoboDK or Tecnomatix Process Simulate), which often provide Python application programming interfaces for direct interaction with robot programs and simulation features [46]. This integration enables full automation of the simulation workflow, including virtual validation and the subsequent automatic generation of error-compensated robot motions for any newly created program. In this context, as most industrial controllers support Ethernet-based communication protocols that enable instruction streaming (see interesting openly released solutions in Refs. [47,48]), the physical robot can also be potentially incorporated as an active node, thereby supporting a Digital Twin architecture consistent with recent approaches to cyber–physical manufacturing systems [1]. Specifically, motion instructions can be streamed directly from the virtual model to the controller via Ethernet communication, while process data can be simultaneously retrieved through the same channel. This bidirectional information flow allows continuous synchronization between the physical and virtual spaces, enabling the Digital Twin not only to replicate but also to adapt to evolving operating conditions.

3. Experimental setup and approach

This section presents the experimental workflow adopted for testing on the physical robot, detailing the integration of the equipment used and the procedures implemented in accordance with ISO 9283 [7]. In the final part, a preliminary sensitivity study is conducted to identify the key parameters that most significantly influence the robot's positioning error. These findings support the creation of a well-balanced dataset for ML training, which is discussed in Section 4.

3.1. Setup description

The research setup utilized in this work comprises a high-payload IR widely adopted in industrial applications (material handling, spot welding, machining, and large-component assembly) and a laser tracker, enabling accurate and reliable characterization of positioning errors. Orientation data could eventually be retrieved using last-generation laser trackers equipped with dedicated probe accessories. Another valid alternative, already employed in the literature [20,44], is the use of modern camera-based systems, which offer relatively simple setup procedures and provide full pose data, although in this case the positional accuracy is generally lower than that of laser trackers. Other metrology devices, such as ballbars and laser interferometers, were not considered suitable for this application, as they are limited to purely circular and linear motions and also require specific installations and optic alignments that would restrict the test conditions and increase the overall setup time (see detailed discussions in Refs. [37,38]). The chosen device, namely a FARO Vantage tracker, offers a flexible yet precise platform for developing and validating position compensation strategies applicable to industrial practice.

A schematic representation of the implemented setup is provided in Fig. 2, with the main parameters of the employed devices reported in Table 1. The selected robot is a KUKA KR210 R2700 Prime, with a payload capacity of 210 kg and a maximum reach of 2700 mm. The robot, controlled by a KRC4 controller, is mounted on a linear track actuated via two servo-reducers. For the position accuracy tests, in order to effectively isolate the robot contribution and minimize the influence of the rack-and-pinion transmission backlash and compliance, the dual-motor drive system is used to actively lock and stabilize the linear track position during measurements. Specifically, one servomotor operates in position-control mode with a fixed target and a stiff gain configuration, while the other applies a constant torque of 6 N m (corresponding to 150 N m after reduction) to preload the rack, thereby reducing mechanical play and preventing residual motion along the linear axis. Both servomotors are powered with Beckhoff drives and controlled within a dedicated industrial PC (see Table 1 for details). The robot is programmed adopting the standard KUKA file structure, i.e. generating `.src` files containing the PTP instructions and corresponding `.dat` files with target data, which are then downloaded to the controller via the WorkVisual environment. Such files are automatically generated via the test planner script developed in Python (see Fig. 1 and the supplementary material). To account for the effect of payload on position accuracy, three different robot tools, with total masses of 39 kg, 79 kg, and 105 kg respectively, were used during the measurements. These payload levels were selected to reproduce realistic operating conditions of a pick-and-place task and to span the expected working range of the tested robot, consistent with typical industrial applications. As shown in Fig. 2, the tools include a holder plate, representing the minimum load condition during the production cycle (e.g., inspection or positioning tool), a Schunk two-finger gripper, and a Schunk three-finger gripper carrying a representative workpiece. In general, the number and type of payloads to be tested depend on the specific industrial context, taking into account the range of tool masses and the accuracy requirements associated with each task.

During the tests, the FARO tracker records the 3D Cartesian coordinates (x, y, z) of a Spherically Mounted Retroreflector (SMR) attached to the robot end-effector, with a nominal accuracy of $20\ \mu m + 5\ \mu m/m$. The SMR is securely mounted to each tool flange using a custom holder designed to ensure consistent alignment throughout the experiments. Measurements are conducted at the maximum sampling frequency of 1000 Hz and are initiated through the FARO tracker's utility software with a duration slightly exceeding the robot test execution. Afterwards, a dedicated post-processing script is launched on the lab PC to isolate the measurement data corresponding to each robot position in order to accurately compute the position error and organize the dataset for ML training.

3.2. Accuracy index

The performance evaluation focuses exclusively on position accuracy, which quantifies the deviation between a commanded spatial position and the average of the actual positions attained by the robot when approaching the target from the same direction. According to ISO 9283 standard [7] and Fig. 3, the position accuracy AP is defined as:

$$AP = \sqrt{{AP_x}^2 + {AP_y}^2 + {AP_z}^2} \tag{1}$$

$$AP_x = x_c - \bar{x}$$

$$AP_y = y_c - \bar{y}$$

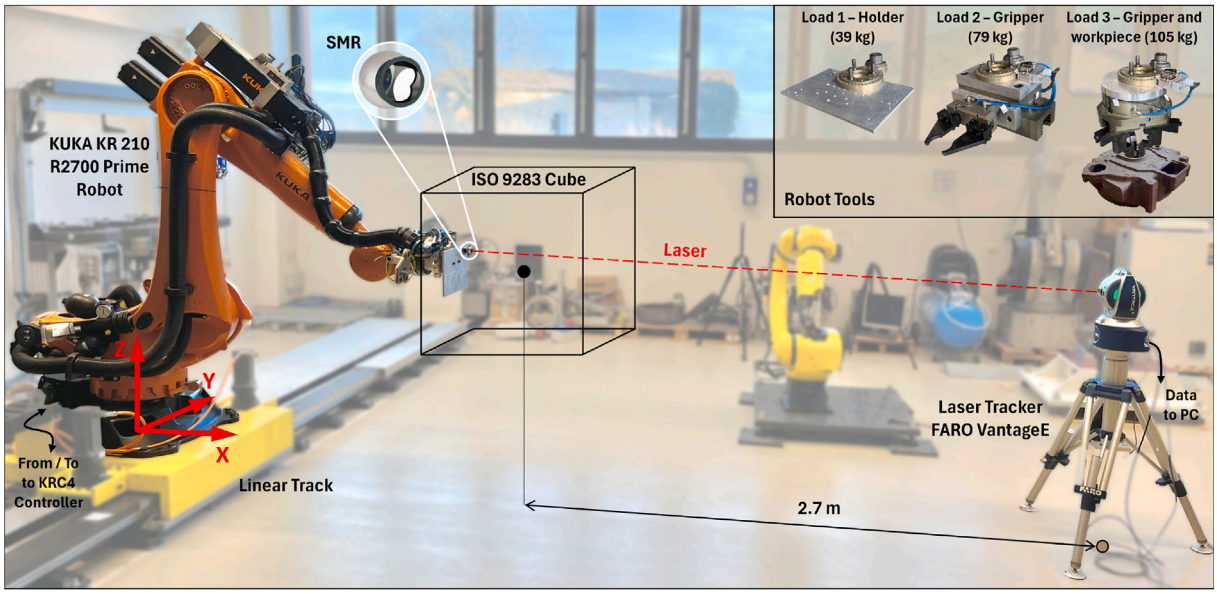

Fig. 2. Experimental setup comprising a KUKA high-payload IR, a laser tracker, and interchangeable tools.

Table 1
Relevant parameters of the installed equipment.

Device	Model	Main characteristics
Robot	KUKA KR210 R2700 prime	Reach: 2700 mm Payload: 210 kg Total mass (incl. box and cables): 1190 kg Controller: KRC4 (KSS 8.3.25) Tool changer: Stäubli MPS 260
Linear track	Custom solution	Stroke: 4300 mm Transmission: Rack-and-pinion (pitch diameter 106.1 mm) Actuation: 2 × Beckhoff AM8052 servomotors + reducer 25:1 Controller: Beckhoff PC C6040-0090 + 2 × Beckhoff AX8118 drives
Tools	1: Holder 2: Schunk PGN+380/2 & PGN+160/1 grippers 3: Schunk PZN+240/2 gripper	Tot. mass (including tool changer and flanges): 39 kg Tot. mass (including tool changer and flanges): 79 kg Tot. mass (including tool changer, flanges and workpiece): 105 kg
Laser tracker	FARO vantage	3D position measurement (no orientation) Max sampling frequency: 1000 Hz Resolution: 0.5 μm Accuracy: 20 μm + 5 μm/m Connectivity: Ethernet

$$AP_z = z_c - \bar{z}$$

where (x_c, y_c, z_c) denotes the commanded position in the robot base frame, and $(\bar{x}, \bar{y}, \bar{z})$ represents the mean of the n actual positions reached during repeated executions of the same command. These mean values are computed as:

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \quad \bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i \quad \bar{z} = \frac{1}{n}\sum_{i=1}^{n} z_i \tag{2}$$

where (x_i, y_i, z_i) are the ith measured values obtained from the laser tracker for the corresponding robot position. The AP index, which represents a position error, effectively captures the absolute positioning performance of the robot without accounting for repeatability or path-following behavior. In the experimental activity, for each target position the values of AP_x, AP_y, and AP_z are evaluated by approaching from six different directions, namely the positive and negative X, Y, and Z axes of the robot base frame, shown in Fig. 2. Position data is extracted from the saved files under static conditions, immediately after the robot reached its final target position.

3.3. Test procedures

The measurement points were arranged within a 1 m cube aligned to the robot base frame and located in the frontal workspace of the robot (positive X), as shown in Fig. 1, to reflect a representative operating region of the robot commonly involved in industrial tasks such as part handling. To allow sufficient acceleration and deceleration phases when reaching each target at varying commanded velocities (specified as a percentage of the maximum allowable level), also accounting for the approach direction, an extended cube with a 1.6 m side length must be reserved. The additional length of 0.3 m on each side ensures that all test points belonging to the external faces of the 1 m cube can be reliably reached under varied test conditions. Then, to ensure repeatability and eliminate ambiguity arising from multiple joint configurations corresponding to the same Cartesian position, the robot kinematic configuration was explicitly defined at the beginning of each robot program. The reference point corresponding to the cube center was programmed as:

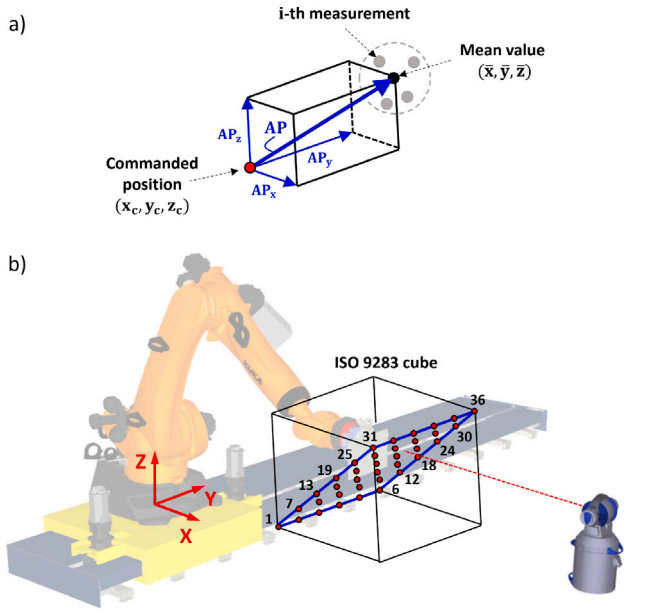

Fig. 3. Position accuracy assessment: (a) Graphical definition of the AP accuracy index; (b) ISO test domain showing the 36 target points used in the preliminary sensitivity study.

```
Center = {X 2090.0, Y 0.0, Z 1000.0, A 0.0, B 90.0, C 0.0, S 2, T 10, E1-E6 = 0.0}
```

In this definition, the parameters S (Status) and T (Turn) uniquely specify the robot's kinematic configuration. Here, $S = 2$ (binary 010) indicates that the wrist center is located in the basic area (positive X), with the arm configuration defined by a positive value of joint-3 and the wrist configuration by a positive value of joint-5. The $T = 10$ parameter further defines the signs of the joint angles that determine the initial pose. For all subsequent PTP motions, the joint solution yielding the shortest and most consistent motion path was always selected, ensuring that the same type of configuration was maintained across all program executions. In this work, as the orientation is not measured by the tracker, the end-effector is maintained at the same initial orientation $(0, 90, 0)$, expressed according to the Euler ZYX convention.

The commanded targets, expressed in Cartesian coordinates to ensure practical applicability in industrial contexts and consistency with ISO 9283, were navigated by a dedicated robot program, which sequentially moved the robot to each target point with intermediate pauses to allow for mechanical stabilization. A dwell time of 4 s was imposed at each position to ensure static conditions suitable for accurate laser tracker measurements. The test points are distributed in a grid pattern with uniform spacing, forming a full-factorial experimental domain across the cube volume. All experiments were conducted after reaching a stable thermal state of the robot, as typically done in industrial setups prior to starting production. Specifically, a preliminary warm-up phase was performed to mitigate the effects of thermal transients, which are known to influence positional accuracy due to temperature-dependent variations within the speed reducers (see the results in Refs. [23,41]). Since most IR lack embedded temperature sensors in their speed reducers (e.g., to monitor oil temperature), temperature effects are not explicitly modeled in this study. The collected dataset (see Section 4) corresponds to a thermally stabilized condition, which settles around 40–45 °C after continuous motion, as documented in Ref. [49]. This ensures that the thermal behavior is consistent and its influence minimized throughout the study.

To ensure complete visibility of all commanded targets while optimizing measurement accuracy, the laser tracker is positioned 2.7 m from the center of the measurement cube. This placement ensures a measurement accuracy of 33.5 μm, as defined by the relation presented in Section 3.1. To correlate the position measurements acquired by the laser tracker, expressed in its own coordinate frame, to the robot base frame, a geometric transformation is applied. The transformation matrix is obtained using the Circle Point Analysis method, as described in detail in Ref. [50] and adapted by the Authors in Refs. [17,51]. Following this approach, the robot performs single rotational motions by individually moving joint-1 and then joint-2, while the SMR is respectively fixed to link-1 and link-2. The resulting 3D circular paths allow for the geometric extraction of the joints axes. By combining this information with the known vertical offset between joint 1 and the robot base plane, the position and orientation of the robot base frame can be determined, thus enabling the complete frame transformation. A similar procedure, involving joint-5 and joint-6 is then performed for each installed tool, namely the holder, the 2-fingers gripper and the 3-fingers gripper holding the workpiece (see Fig. 2), to accurately identify the location of the SMR relative to the robot flange. This information is then incorporated into the `TOOL_DATA` of the KUKA controller to ensure accurate motion computation throughout the tests.

At last, the Absolute Accuracy mode was deactivated during all tests. This feature, offered by KUKA to enhance the robot's positioning performance, is typically provided as an optional add-on and may not be available in standard industrial setups due to additional costs. To disable it, the system variable `DEACTIVATE_ABS_ACCUR` was set to `TRUE` within the `custom.dat` configuration file.

3.4. Preliminary parametric study

This section presents the preliminary parametric study aimed at identifying how key operating parameters influence the quasi-static accuracy index AP. Given the nature of the proposed compensation tool (see Section 2 and Fig. 1), which operates through code-level target corrections within the robot programs, the present study focuses on parameters directly accessible in such programs. Detailed investigations into the mechanical and control related sources of position errors, such as the primary effects of joint transmissions, are not included here, as these topics have been comprehensively addressed in previous studies. The findings of this investigation serve to inform the data collection strategy for the ML error model described in Section 4.3. Following the guidelines of ISO 9283, the position accuracy was evaluated on targets located on the inclined plane of the test cube. However, while the norm specifies only five reference points (the four corners and the center of the inclined face), in this study the number of targets was increased to 36, as shown in Fig. 3, to achieve a more detailed mapping of position accuracy across the test domain. Each target was sequentially approached from 6 distinct directions, namely along the negative and positive axes of the robot base frame ($-X$, $+X$, $-Y$, $+Y$, $-Z$, and $+Z$) to assess direction-dependent effects. The complete list of targets and their corresponding coordinates is provided in the supplementary material, although it can also be easily derived from the geometry, given the location of the center point and the cube side length.

The first campaign assessed the influence of travel velocity. Using a fixed payload of 39 kg (i.e., mounting the first tool shown in Fig. 2), the complete 36-point, 6-direction sequence was executed at 10%, 25%, and 50% of the robot's maximum velocity, resulting in a total of 648 individual samples (36 × 6 ×3). Within this velocity range, no significant variations in AP were observed, as it can be noted in Fig. 4a. The plot reports the mean AP value for each of the 36 points, computed by averaging the 6 directional measurements, across all three velocity levels (square Vs. circle Vs. triangle markers). To quantitatively verify this outcome, a non-parametric Friedman test was performed in Python using the `friedmanchisquare()` function from the `SciPy.stats` module. This test compares subsets of data to determine whether variations in a single factor (in this case speed) have a statistically significant impact on the performance index AP. In

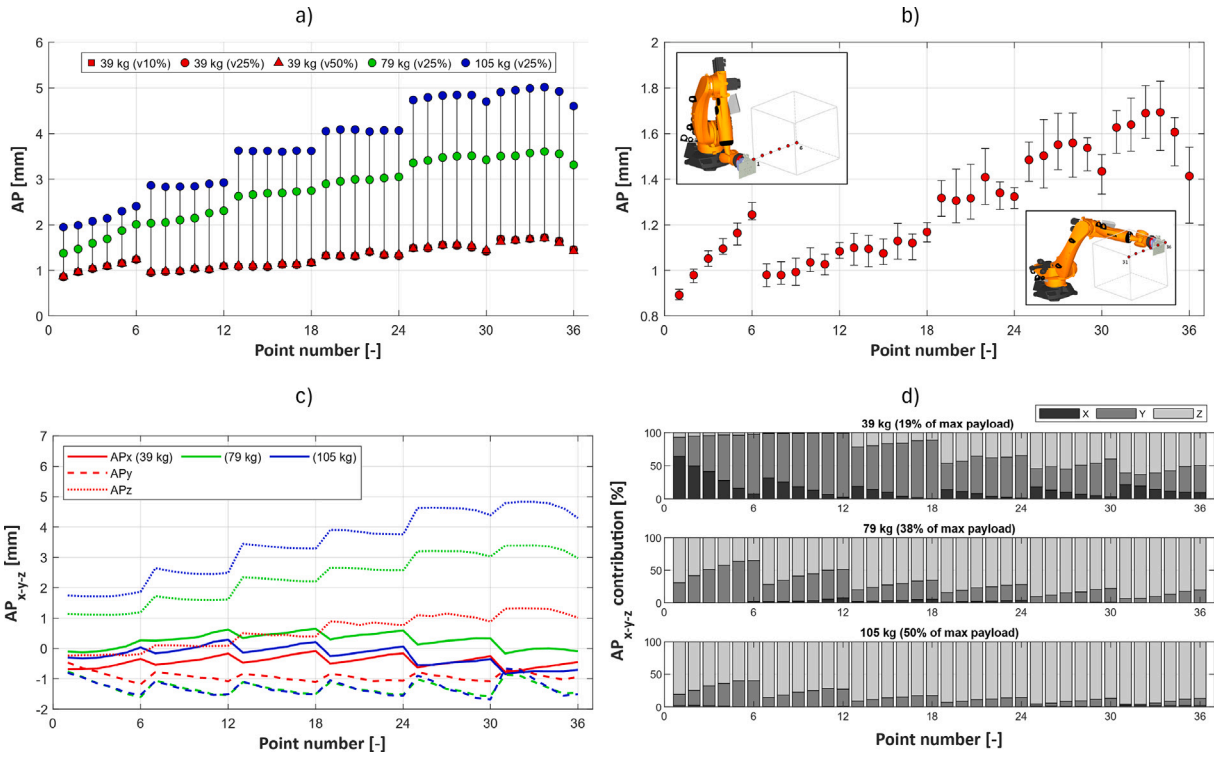

Fig. 4. Results of the sensitivity study: (a) influence of payload and speed on AP, (b) range of AP variation due to approach direction, (c) directional contributions AP_x, AP_y, AP_z and (d) percentage weight of each directional error for the tested payloads.

Table 2
Statistical results showing the effect of parameter variations on AP. Reported values include mean absolute changes (mm), along with the corresponding percentage and p-values. Cell colors indicate impact on accuracy: green for improvement and red for deterioration.

Parameter	Mean AP change [mm]	p-value
Travel speed		
10% to 25%	–	0.45
10% to 50%	–	0.25
25% to 50%	–	0.25
Payload		
39 kg to 79 kg	1.47 (114.3%)	1.8e−48
39 kg to 105 kg	2.46 (190.8%)	1.8e−48
79 kg to 105 kg	0.99 (35.4%)	1.8e−48
Approach direction		
X+ to X-	−0.10 (−6.3%)	2.8e−27
Y+ to Y-	0.03 (1.9%)	3.1e−4
Z+ to Z-	0.09 (6.4%)	3.1e−38
X to Y	–	0.96
X to Z	–	0.31
Y to Z	−0.02 (−1.2%)	2.3e−11

all test runs, the Friedman test yielded negative results, with p-values greater than 0.05, confirming the absence of any statistically significant influence of speed on positioning accuracy. The results of the statistical analysis are summarized in Table 2.

In light of these findings, which align well with previous experimental studies (see, e.g., Ref. [17]), a fixed speed of 25% was selected for the subsequent experiments. Here, the previous 36 × 6 sequence of movements was performed utilizing three different payloads, namely the holder (39 kg), 2-fingers gripper (79 kg), and 3-fingers gripper with workpiece (105 kg). The outputs of the tests are plotted in Fig. 4a (red Vs. green Vs. blue markers). It is observed that the accuracy index AP degraded monotonically with increasing payload, reaching a maximum error of approximately 5 mm at 105 kg (see points 31–35 in the figure). These curves, together with the p-values reported in Table 2, confirmed that payload has a significant influence on accuracy and must be explicitly modeled in the compensation strategy. As expected, the point's location was also found to impact accuracy. As illustrated in Fig. 3, targets located nearer the robot base (e.g., points 1–6), requiring more folded robot configurations, tended to exhibit lower errors, whereas targets in more stretched configurations (e.g., points 31–36) showed increased deviations.

Furthermore, in both the experimental datasets a clear dependency on motion direction was observed. This trend is evident from Fig. 4b, which shows the range of AP variation for each point tested at a payload of 39 kg and a velocity of 25%. This is observed to be around 0.05–0.1 mm in the first points (1–6), where the robot kinematic configuration ensures higher stiffness characteristics, and be up to 0.35 mm in the last points (31–36). Statistical analysis (Table 2) reveals that this dependency is not uniformly significant across all motion directions. Certain transitions, such as from $+X$ to $-X$, or even from Y to Z, exhibit pronounced and recurring variations, whereas others have only marginal or inconsistent effects. This result clearly highlights that the influence of the approach direction on AP cannot be reliably deduced from purely theoretical considerations and instead suggest the use of ML approaches.

Finally, the individual effects of AP_x, AP_y, and AP_z are shown in Fig. 4c, where the mean values across all 36 points are presented. As clearly visible in Fig. 4d, the effect of AP_z becomes predominant either when the robot's stiffness decreases, due to reaching farther points and thus adopting more extended kinematic configurations, or when the payload increases. Overall, these findings led to three practical guidelines for building the comprehensive dataset used in ML model training:

- The travel speed was fixed at 25% to streamline data collection without compromising result quality. Since the robot is mounted on a linear track, it is important to exclude any potential sources of positioning error not directly attributable to the robot itself. In this regard, the imposed speed level helps to reduce dynamic

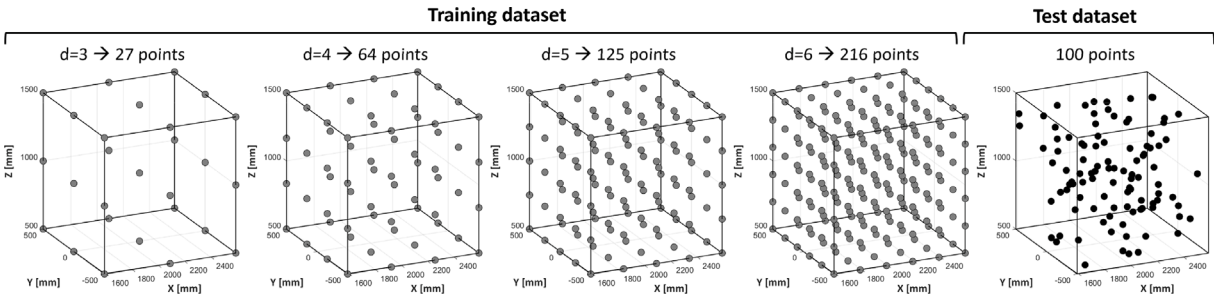

Fig. 5. Spatial distribution of training points using full-factorial grid with granularity $d \in \{3,4,5,6\}$ (resulting in 27, 64 125, 216 points), within the ISO cube centered at (2090, 0, 1000) mm. The final plot shows 100 test points generated via Latin Hypercube sampling for model evaluation.

forces along the linear track, which, as discussed in Section 3.1, is locked in position.
- Include at least three discrete payload levels to span the robot's expected operating range. The dependency on this parameter follows a generally monotonic trend, though it is not strictly linear.
- Retain all six approach directions while ensuring coverage of both central and peripheral workspace regions to investigate different kinematic configurations.

4. Data collection and model training

This section introduces a general data-driven approach for accurately predicting positioning errors in IR using supervised ML models. These models are trained and tested on empirical datasets collected through laser tracker measurements, as described in Section 3.

The problem of estimating the positioning error is formally defined in Section 4.1 as three distinct regression problems, one for each Cartesian coordinate. Each regression problem aims to predict the corresponding error component based on a set of input features. Section 4.2 describes the training sets of varying sizes used to train the models, as well as the test set employed for their validation. Section 4.3 provides an overview of several state-of-the-art ML models selected to address the proposed regression tasks, highlighting their main characteristics and suitability for the compensation strategy. Alongside well-established families of ML models commonly used in the literature for predicting positioning errors in IR, this work introduces the TabPFN [39], a recent ML model that, to the best of the authors' knowledge, has not previously been applied in this context. Section 4.4 starts with a sensitivity analysis assessing the impact of training set characteristics on the ML models accuracy, offering practical guidelines for optimal dataset design in similar industrial applications. Once the training set size that achieves the best trade-off between model accuracy and data acquisition effort is identified, the ML models trained on this dataset are extensively evaluated to select the most accurate error predictors for integration into the compensation module described in Section 2 and validated in Section 5.

4.1. Problem formulation and methodology

The proposed methodology begins with a preliminary parametric study aimed at identifying the input features that most significantly influence the positioning error $AP = (AP_x, AP_y, AP_z)$, where each component of AP represents the difference between the commanded and the actual position reached by the robot along the corresponding Cartesian axis. This analysis, detailed in Section 3.4, revealed that the commanded position, approach direction, and payload level are the most influential features. Based on these findings, the prediction of AP is formulated as three independent regression problems, one for each Cartesian component. The three components are treated separately because the relationship between the positioning error and the input features differs across the X, Y, and Z axes, and the corresponding errors are only weakly correlated due to axis-specific mechanical characteristics and physical phenomena. For instance, gravity makes the Z-axis positioning error more sensitive to variations in payload level than the X and Y components. Therefore, treating each component independently allows the learning process to more accurately capture axis-specific dependencies. To validate this choice, the prediction of AP was also attempted using a single predictor for all three axes. This approach produced slightly less accurate results (see Section 4.4), confirming the benefit of treating each Cartesian component independently.

Focusing on the X component, the first regression problem consists in estimating AP_x by learning a function $\widetilde{AP}_x : \mathcal{F} \to \mathbb{R}$, defined over the 7-dimensional input space $\mathcal{F} = \mathcal{C} \times \{0, \pm 1\}^3 \times \mathbb{R}_+$. Here, $\mathcal{C} = \{(x, y, z) \in \mathbb{R}^3 \mid 1590 \le x \le 2590;\ -500 \le y \le 500;\ 500 \le z \le 1500\}$ defines the workspace region of the commanded target positions (see Section 3), while the set $\{0, \pm 1\}^3$ encodes the approach direction. For instance, $(1, 0, 0) \in \{0, \pm 1\}^3$ denotes an approach direction to the target point along the positive X-axis of the robot base frame, whereas $(0, -1, 1) \in \{0, \pm 1\}^3$ represents an approach direction along the negative Y-axis and positive Z-axis. Finally, the payload is represented by a non-negative continuous variable in $\mathbb{R}_+$. A supervised learning approach is employed to train different ML models for $\widetilde{AP}_x$, using a dataset $\mathcal{D}^x_{\text{train}} = \{(f^i, AP^i_x)\}_{i=1}^{1152}$, where each sampled $f^i \in \mathcal{F}$ is a 7-dimensional input vector, and $AP^i_x \in \mathbb{R}$ is the corresponding measured positioning error. The learning task is to find a function $\widetilde{AP}_x$, selected from a given family of ML models, such that for each input $f^i \in \mathcal{F}$, the predicted value $\widetilde{AP}_x(f^i)$ is as close as possible to the target value AP^i_x. The functions $\widetilde{AP}_y$ and $\widetilde{AP}_z$ are defined analogously to $\widetilde{AP}_x$, each mapping the input space $\mathcal{F}$ to $\mathbb{R}$ and learned by independently training supervised ML models on the datasets $\mathcal{D}^y_{\text{train}} = \{(f^i, AP^i_y)\}_{i=1}^{1152}$ and $\mathcal{D}^z_{\text{train}} = \{(f^i, AP^i_z)\}_{i=1}^{1152}$, respectively.

In Section 4.4, for each coordinate, the final unbiased effectiveness of the trained models is computed on the provided test sets $\mathcal{D}^x_{\text{test}}$, $\mathcal{D}^y_{\text{test}}$, and $\mathcal{D}^z_{\text{test}}$, respectively, to select the most accurate error predictors for integration into the compensation tool described in Section 5.

4.2. Data description

To train accurate and generalizable ML models, the training sets must provide balanced coverage of the input space $\mathcal{F}$. In this work, the training samples were generated using uniform design, a method originally introduced in the early 1980s, which aims to uniformly distribute experimental points across $\mathcal{F}$. Uniform design has been widely adopted in industrial experiments, particularly in cases where the underlying model and the relationships between inputs and outputs are unknown (see, e.g., Ref. [52]). Let $d \in \{3,4,5,6\}$, and let $\mathcal{C}^d_{\text{grid}}$ denote a uniform grid over the commanded position space $\mathcal{C}$, obtained by selecting d equally spaced values along each Cartesian axis, resulting in a total of d^3 spatial points, as shown in Fig. 5. The sets $D = \{(\pm 1, 0, 0),\ (0, \pm 1, 0),\ (0, 0, \pm 1)\}$ and $P = \{39,\ 79,\ 105\}$ represent the six

Table 3
Example of a data row from $\mathcal{D}^x_{\text{train}}$ illustrating the adopted CSV structure. The same format applies to $\mathcal{D}^y_{\text{train}}$ and $\mathcal{D}^z_{\text{train}}$.

Commanded position [mm]			Approach direction			Payload [kg]	AP_x [mm]
x	y	z	X	Y	Z		
1590	−500	500	−1	0	0	105	0.31

principal approach directions and the three payload levels, respectively, used in the experimental setup described in Section 3.

For the datasets used in the preliminary sensitivity analysis, only the payload of 39 kg was considered. As highlighted in Section 3.4, the effect of varying the payload on the positioning error follows an approximately linear trend, which is effectively captured by all the ML models considered in this paper. Thus, the choice to consider only 39 kg as the payload in this phase allowed reducing the time required for data acquisition without compromising the validity and generality of the results for scenarios involving different payload values. For each $d \in \{3,4,5,6\}$, the input design grid is defined as $\mathcal{F}^d_{\text{grid}} = C^d_{\text{grid}} \times D \times \{39\}$. For each $f^i \in \mathcal{F}^d_{\text{grid}}$, the corresponding positioning error $AP^i = (AP^i_x, AP^i_y, AP^i_z)$ was measured using the laser tracker. Assuming that each point of the datasets is approached at a fixed speed of 25%, the total experimental time required to collect these datasets is approximately 0.3, 1, 2, and 3.5 h for $d = 3,4,5,6$, respectively. The significant difference in data acquisition times underscores the importance of the sensitivity study, as time efficiency is critical in many industrial applications. Section 4.4 presents this analysis, which aims to identify the smallest acceptable granularity $d \in \{3,4,5,6\}$ of the design grid $\mathcal{F}^d_{\text{grid}}$ that still ensures sufficient prediction accuracy for the ML models. Following data acquisition, for each $d \in \{3,4,5,6\}$, the three resulting training datasets, each containing $m = |\mathcal{F}^d_{\text{grid}}|$ sample points, were defined as $\mathcal{D}^{d,x}_{\text{train}} = \{(f^i, AP^i_x)\}^m_{i=1}$, $\mathcal{D}^{d,y}_{\text{train}} = \{(f^i, AP^i_y)\}^m_{i=1}$, and $\mathcal{D}^{d,z}_{\text{train}} = \{(f^i, AP^i_z)\}^m_{i=1}$. To construct the test sets $\mathcal{D}^x_{\text{test}}$, $\mathcal{D}^y_{\text{test}}$, and $\mathcal{D}^z_{\text{test}}$, 100 input configurations $\{t^i\}^{100}_{i=1}$ were sampled from the input space $C \times \{0, \pm 1\}^3 \times \{39\}$ using Latin hypercube sampling (see Fig. 5), a space-filling technique that distributes points evenly across the domain while avoiding clustering and redundant samples (see, e.g., Ref. [53]). For each generated input configuration t^i, the corresponding positioning error $AP^i = (AP^i_x, AP^i_y, AP^i_z)$ was measured using the laser tracker. The resulting test sets were defined as $\mathcal{D}^x_{\text{test}} = (t^i, AP^i_x)^{100}_{i=1}$, $\mathcal{D}^y_{\text{test}} = (t^i, AP^i_y)^{100}_{i=1}$, and $\mathcal{D}^z_{\text{test}} = (t^i, AP^i_z)^{100}_{i=1}$.

Once the sensitivity analysis, conducted in Section 4.4, identified $d = 4$ as the optimal granularity, the input design grid $\mathcal{F}^4_{\text{grid}}$ was extended to $C^4_{\text{grid}} \times D \times P$ to include multiple payload levels. Similarly, the points generated via Latin hypercube sampling were measured for all payloads in P. The resulting training sets $\mathcal{D}^x_{\text{train}} = (f^i, AP^i_x)^{1152}_{i=1}$, $\mathcal{D}^y_{\text{train}} = (f^i, AP^i_y)^{1152}_{i=1}$, and $\mathcal{D}^z_{\text{train}} = (f^i, AP^i_z)^{1152}_{i=1}$ were used to train the final ML models. On the other hand, the corresponding test sets, denoted (by slight abuse of notation) as $\mathcal{D}^x_{\text{test}} = \{(t^i, AP^i_x)\}^{300}_{i=1}$, $\mathcal{D}^y_{\text{test}} = \{(t^i, AP^i_y)\}^{300}_{i=1}$, and $\mathcal{D}^z_{\text{test}} = \{(t^i, AP^i_z)\}^{300}_{i=1}$, were used to compare the ML models and evaluate their predictive performance, with the objective of selecting the most accurate predictors for integration into the compensation tool described in Section 5.

All training and test datasets were stored in Comma-Separated Values (CSV) files, each containing eight columns. The first seven columns represent the input features: the commanded position coordinates, the encoded approach direction, and the payload level. The last column contains the corresponding measured positioning error component. Table 3 shows an example data row of $\mathcal{D}^x_{\text{train}}$ to illustrate the CSV file structure. The datasets $\mathcal{D}^y_{\text{train}}$ and $\mathcal{D}^z_{\text{train}}$ have the same structure as $\mathcal{D}^x_{\text{train}}$, with the only difference that the last column reports AP_y and AP_z, respectively, instead of AP_x.

During the experiments, outlier observations were systematically detected, and the corresponding measurements were repeated to ensure high data quality. Consequently, no additional data-cleaning procedures were required.

4.3. ML models

In problems related to predicting positioning errors in IR, neural networks and deep learning models have been widely applied due to their flexibility and ability to model complex nonlinear relationships (see, e.g., Refs. [54,55]). However, their performance typically relies on the availability of large volumes of training data. This assumption does not always hold in real-world industrial scenarios, where data collection can be expensive, time-consuming, or constrained by operational limitations. As a result, limited data availability can significantly affect the prediction accuracy of such models [35]. Alternative approaches, such as support vector machines and tree-based models, which perform well with smaller datasets, have been successfully employed (see, e.g., Refs. [41,56]).

To further enhance predictive performance, including in the context of small to medium-sized training sets, this study proposes the use of the TabPFN for the prediction of positioning errors. TabPFN is a generative, transformer-based foundation model recently introduced in Ref. [39]. Unlike traditional supervised ML models, TabPFN employs the transformer architecture, which is the same underlying architecture used in large language models [57]. TabPFN not only outperforms state-of-the-art models on datasets with up to 10 000 samples and 500 features, but is also capable of modeling both smooth and non-smooth function types out of the box [39]. This capability represents a valuable property when the characteristics of the underlying function to be learned are unknown or not fully explored.

To ensure a comprehensive evaluation, this work benchmarks TabPFN against several well-established ML models commonly used in the literature. These models were chosen to represent the main families of supervised ML approaches for regression, neamly, linear models, kernel-based methods, neural networks, and ensemble tree-based learners. The specific ML models implemented in this study are:

- Elastic Net Regression (EN) is a linear regression model that combines the penalties of Lasso and Ridge regression [58]. While it cannot model complex nonlinear patterns, it is used in this study as a baseline for comparison.
- Support Vector Regression (SVR) is a kernel-based method that aims to find a function that approximates the target values with deviations no greater than a specified margin of tolerance. SVR can effectively capture nonlinear relationships through kernel functions, which implicitly map the input data into a higher-dimensional feature space [59].
- Multi-Layer Perceptron (MLP) is a feedforward artificial neural network consisting of multiple layers of interconnected neurons. It can capture nonlinear mappings between input features and outputs, making it suitable for regression problems with complex dependencies [33].
- Three different ensemble tree-based learners:
  - Gradient Boosting Machine (GBM), built iteratively into a single strong learner. At each step, a new weak learner is fitted to improve the estimation of the response variable. GBM is effective at capturing nonlinear relationships and feature interactions [60].
  - XGBoost (XGB) is an efficient implementation of gradient boosting that builds additive tree models in a sequential manner. It incorporates regularization techniques to prevent overfitting [61].
  - CatBoost (CB) is a gradient boosting framework that handles categorical variables natively and uses ordered boosting to reduce overfitting. It is particularly effective on heterogeneous datasets common in industrial applications [62].
- TabPFN is an ML model in the family of Prior-Data Fitted Networks (PFNs) that extends the standard transformer architecture to two dimensions, enabling attention to be computed across both features and samples. This two-way attention treats the dataset as

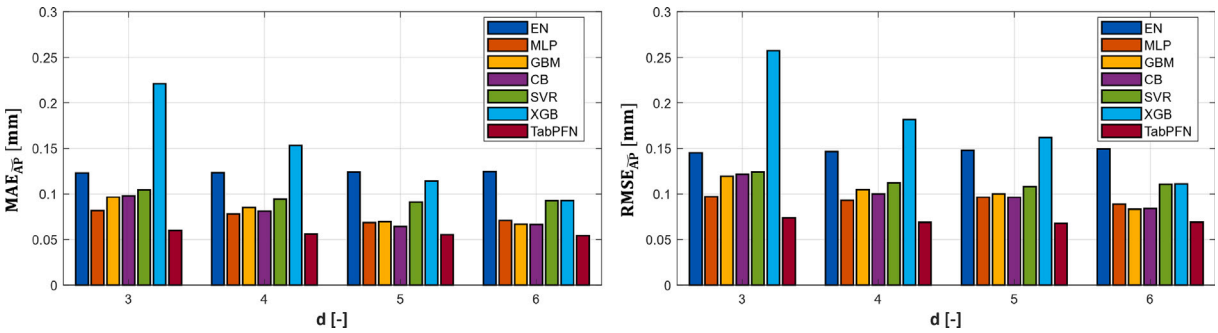

Fig. 6. Average MAE and RMSE values computed for all tested ML models, evaluated at different values of the design grid granularity d.

a grid rather than a sequence, making the model invariant to the ordering of samples and features [63]. Unlike traditional supervised ML models, TabPFN is pretrained on millions of synthetic datasets, with the training dataset provided only as context. As a result, TabPFN can produce accurate predictions without any additional training or parameter tuning, making it particularly suitable for practical use in industrial settings [39].

While these models have demonstrated efficacy in various contexts, each presents limitations that may affect prediction accuracy. SVR is computationally inefficient on large datasets and requires careful, often complex, parameter tuning. MLPs demand substantial training data and long training times and tend to perform poorly on datasets featuring highly irregular or non-smooth patterns, such as step functions [34]. On the other hand, the remaining tree-based models, namely RF, XGB, and CB, often struggle to accurately capture smooth and continuous functional relationships [39]. This property is crucial when smooth predictions are required for effective error compensation in robotic systems, as abrupt fluctuations in the compensated values may compromise system stability [64].

4.4. Performance evaluation of ML models

Computational experiments were conducted to evaluate the performance of the ML models introduced in Section 4.3. These models were developed in Python using the open-source `scikit-learn` library [65,66], while TabPFN was implemented directly using code from its official repository https://github.com/PriorLabs/TabPFN. All training and testing procedures were carried out on a computer equipped with a 2.1 GHz Intel Xeon Gold 6430 processor and 16 GB of memory.

The first part of this section is devoted to a sensitivity analysis aimed at identifying the smallest acceptable granularity $d \in \{3,4,5,6\}$ of the grid $\mathcal{F}^d_{\text{grid}}$ that achieves optimal dataset design by reducing data collection while maintaining accurate model predictions. Indeed, as d increases, the number of points in $\mathcal{F}^d_{\text{grid}}$ grows exponentially, resulting in substantially larger training datasets. As detailed in Section 4.2, this increase significantly extends data acquisition time but, conversely, may improve model accuracy by offering more comprehensive coverage of the input space. For each $d \in \{3,4,5,6\}$, each ML model was trained on the training sets $\mathcal{D}^{d,x}_{\text{train}}$, $\mathcal{D}^{d,y}_{\text{train}}$, and $\mathcal{D}^{d,z}_{\text{train}}$, and evaluated on the corresponding test sets $\mathcal{D}^{x}_{\text{test}}$, $\mathcal{D}^{y}_{\text{test}}$, and $\mathcal{D}^{z}_{\text{test}}$, respectively. A 10-fold cross-validation procedure [67] was used to ensure a robust and unbiased assessment of model performance. The accuracy of the trained models has been evaluated using two widely adopted statistical metrics: the Mean Absolute Error (MAE) and the Root Mean Square Error (RMSE). These indicators are computed separately for each component of the positioning error. The MAE, commonly used in the forecasting literature [68], measures the average absolute difference between the predicted and actual values. For the X axis, the MAE is computed as:

$$\text{MAE}_{\widetilde{AP}_x} = \frac{1}{\left|\mathcal{D}^{x}_{\text{test}}\right|} \sum_{(f^i, AP^i_x) \in \mathcal{D}^{x}_{\text{test}}} \left| AP^i_x - \widetilde{AP}_x(f^i) \right|.$$

$\text{MAE}_{\widetilde{AP}_y}$ and $\text{MAE}_{\widetilde{AP}_z}$ are calculated in a similar manner. The RMSE, on the other hand, gives more weight to larger errors and is particularly useful when large deviations are especially undesirable. For the X axis, the RMSE is defined as:

$$\text{RMSE}_{\widetilde{AP}_x} = \sqrt{\frac{1}{\left|\mathcal{D}^{x}_{\text{test}}\right|} \sum_{(f^i, AP^i_x) \in \mathcal{D}^{x}_{\text{test}}} \left(AP^i_x - \widetilde{AP}_x(f^i) \right)^2}.$$

$\text{RMSE}_{\widetilde{AP}_y}$ and $\text{RMSE}_{\widetilde{AP}_z}$ are computed similarly. Fig. 6 shows the average MAE and RMSE values for all tested ML models at different levels of design grid granularity $d \in \{3,4,5,6\}$. As expected, model accuracy generally improves as d increases. Fig. 6 highlights that, across all granularity levels, TabPFN is the best predictor, consistently achieving the lowest MAE and RMSE values. Among the remaining models, MLP, CB, and GBM demonstrate good overall accuracy. In contrast, the linear model EN achieved the highest overall error values, highlighting its limitations in capturing the complex and nonlinear relationships inherent in the positioning error. Considering the trade-off between predictive accuracy and data acquisition effort, a granularity of $d = 4$ was selected as optimal. Indeed, the accuracy improvement of TabPFN when moving from $d = 3$ to $d = 4$ is particularly evident along the X axis: $\text{MAE}_{\widetilde{AP}_x}$ decreases from 0.049 mm to 0.039 mm, while the $\text{RMSE}_{\widetilde{AP}_x}$ decreases from 0.057 mm to 0.045 mm. On the other hand, further increases in granularity result in only negligible improvements in the accuracy of TabPFN, while significantly increasing the time required to generate the training data. Specifically, the sampling time varies from approximately 1 h for $d = 4$ to 3.5 h for $d = 6$. Moreover, for granularities beyond $d = 4$, all other models remain less accurate than TabPFN (for $d = 4$). Therefore, $d = 4$ provides a good balance between predictive performance and data acquisition cost across all evaluated ML models and granularities.

The second part of this section provides a detailed performance comparison of the final ML models trained on the comprehensive datasets with granularity $d = 4$, considering all payload levels in P. The resulting training set contains $4^3 \times 6 \times 3 = 1152$ samples, which is smaller than the dataset sizes typically used by small-sample ML models in the literature, whose training sets generally contain around 2000 samples collected in a workspace smaller than the region C considered in this study (see, e.g., Refs. [22,69]). This choice aligns with the goal of developing a methodology that maintains accuracy while reducing the data acquisition effort compared with existing approaches. Each model was trained independently for the three Cartesian axes, using the complete training datasets $\mathcal{D}^{x}_{\text{train}} = \{(f^i, AP^i_x)\}_{i=1}^{1152}$, $\mathcal{D}^{y}_{\text{train}} = \{(f^i, AP^i_y)\}_{i=1}^{1152}$, and $\mathcal{D}^{z}_{\text{train}} = \{(f^i, AP^i_z)\}_{i=1}^{1152}$, respectively. To ensure a fair comparison, hyperparameter optimization was performed using `Optuna` [70]. Specifically, 100 trials were executed with the Tree-structured Parzen Estimator (TPESampler) optimization algorithm. For each ML model, the search space (reported in Table 4) spans the full range of the relevant hyperparameters, i.e., those with a relative parameter importance score of at least 5%, as estimated by `Optuna`.

Table 4
Results of the hyperparameter optimization conducted using 100 Optuna trials. For each model, the table reports the explored search space and the hyperparameter configuration yielding the best accuracy along the X, Y, and Z axes.

Model	Hyperparameter	Search space	X	Y	Z
EN	fit_intercept	{True, False}	False	True	True
SVR	kernel	{'rbf', 'linear', 'poly'}	rbf	rbf	rbf
	C	[0.001, 100]	11	25	12
	epsilon	[0.0001, 1]	0.008	0.014	0.005
	degree	[2,6]	2	2	5
MLP	hidden_layer_sizes	{(50, 50), (100, 100), (1000, 1000), (1000, 500, 100), (2000, 1000, 200)}	(100,100)	(1000,1000)	(50,50)
	activation	{'relu', 'tanh'}	tanh	relu	tanh
	solver	{'adam', 'lbfgs'}	lbfgs	lbfgs	lbfgs
	alpha	[1e−5, 1]	0.016	0.008	0.007
GBM	n_estimators	[50, 1000]	550	709	533
	learning_rate	[0.001, 1]	0.29	0.24	0.18
	min_samples_split	[2, 20]	6	19	8
	subsample	[0.5, 1]	0.94	0.86	0.88
XGB	n_estimators	[100, 1000]	445	456	817
	learning_rate	[0.001, 1]	0.01	0.01	0.03
	gamma	[0,100]	0.013	0.013	0.0013
CB	n_estimators	[100, 1000]	651	524	759
	random_strength	[0,1]	0.16	0.53	0.85
	learning_rate	[0.001, 1]	0.78	0.30	0.22
	depth	[1,10]	3	4	3
	l2_leaf_reg	[1,10]	3.16	9.21	6.04

Table 5
Comparison of MAE and RMSE values for the positioning error components along the X, Y, and Z axes for each ML model, evaluated at $d = 4$. average values are reported at the bottom.

	EN	SVR	MLP	GBM	XGB	CB	TabPFN
$MAE_{\widetilde{AP}_x}$	0.318	0.402	0.041	0.065	0.117	0.065	**0.039**
$RMSE_{\widetilde{AP}_x}$	0.360	0.437	0.052	0.081	0.154	0.082	**0.049**
$MAE_{\widetilde{AP}_y}$	0.208	0.256	0.145	0.115	0.126	0.100	**0.096**
$RMSE_{\widetilde{AP}_y}$	0.248	0.287	0.174	0.145	0.150	0.126	**0.121**
$MAE_{\widetilde{AP}_z}$	0.739	0.533	0.156	0.155	0.344	0.155	**0.070**
$RMSE_{\widetilde{AP}_z}$	0.869	0.612	0.209	0.189	0.416	0.190	**0.093**
$MAE_{\widetilde{AP}}$	0.422 (+517%)	0.397 (+481%)	0.114 (+67%)	0.111 (+63%)	0.196 (+186%)	0.106 (+56%)	**0.068 (Ref.)**
$RMSE_{\widetilde{AP}}$	0.492 (+462%)	0.445 (+408%)	0.145 (+66%)	0.138 (+58%)	0.240 (+174%)	0.133 (+52%)	**0.088 (Ref.)**

All remaining parameters were kept at their default settings. The best-performing configurations for each ML model are reported in Table 4. For the traditional ML models, the training phase ranged from 20 s for the EN to 4724 s for the MLP, whereas the fitting phase for TabPFN required only 0.25 s. The MAE and RMSE values obtained by each ML model for the positioning error components are reported in Table 5. The results align with the preliminary sensitivity analysis, confirming TabPFN as the best predictor across all three axes. Indeed, TabPFN achieves the lowest errors for both metrics without requiring any hyperparameter tuning. In comparison, all other models show substantially higher errors, with increases ranging from +56% to +517% for MAE and from +52% to +462% for RMSE compared to TabPFN. This substantial margin establishes it as the most reliable and effective model among those tested for integration into the compensation tool. Finally, to validate the choice of training separate models for each axis, the same methodology was applied using a single predictor for all three axes. This approach yielded slightly but consistently lower predictive accuracy, in terms of both MAE and RMSE (with average increases of 4.6% in MAE and 2.7% in RMSE across the models), further confirming that training separate models for each axis is more effective.

5. Tool validation

This section evaluates the practical effectiveness of the proposed compensation tool. Based on the results in Section 4.3 (illustrated in Fig. 6 and Table 5), the best-performing ML model, TabPFN, trained on the 64-point dataset with a granularity of $d = 4$ is selected for error prediction during the tests.

5.1. Error compensation

The workflow introduced in Section 2 (see Fig. 1, module 1) has been implemented as in the Python script shown below. This script is executed with four command-line inputs: the original robot code file (`.src` or `.dat` in the case of KUKA IR), the pre-trained ML model file, the output file path for saving the corrected code, and the payload parameter representing the effective load during the planned robotic task.

```
#!/usr/bin/env python3
import re, pickle, numpy as np, pandas as pd, argparse
from pathlib import Path

# Block 1: User's input
# paths of initial robot code, ML model and payload
ap = argparse.ArgumentParser()
ap.add_argument("-i","--input", type=Path,
 ↪  required=True)
ap.add_argument("-m","--model", type=Path,
 ↪  required=True)
ap.add_argument("-p","--payload", type=float,
 ↪  required=True)
args = ap.parse_args()
dat, mdl, payl = args.input, args.model, args.payload

# Block 2: Parse the Input Robot Code File and Build
 ↪  Input Features
# expression to be recognized in the robot file
pat = re.compile(r"\{\s*X\s*([-\d.eE]+),\s*Y\s*([-\d
 ↪  .eE]+),\s*Z\s*([-\d.eE]+)[^}]*\}")
```

```
# initialization
raw, rows, prev = [], [], None

# robot file scanning (extract coordinates and
 ↪  directions)
for i, line in enumerate(dat.open()):
   raw.append(line)
   m = pat.search(line)
   if not m: continue
   x, y, z = map(float, m.groups())
   s = (0,0,0) if prev is None else np.sign([x-prev[0],
    ↪  y-prev[1], z-prev[2]])
   prev = (x, y, z)
   rows.append(dict(row=i, x_cmd=x, y_cmd=y, z_cmd=z,
    ↪  i=s[0], j=s[1], k=s[2], payload=payl))

# create a structured input matrix for ML model
df = pd.DataFrame(rows)
X = df[['x_cmd','y_cmd','z_cmd','i','j','k','payload
 ↪  ']]

# Block 3: Load ML Model and Predict Errors
# load the pre-trained ML model
models = pickle.load(mdl.open('rb'))

# predict errors and compute corrected positions
err = {c: models[c].predict(X) for c in 'xyz'}
fix = {c: df[f'{c}_cmd'] + err[c] for c in 'xyz'}

# Block 4: Rewrite Robot Code File
# rewrite the original file by replacing position data
lookup = {r: (fix['x'][k], fix['y'][k], fix['z'][k])
 ↪  for k, r in enumerate(df['row'])}
(out_dir := dat.parent / "output").mkdir(parents=True,
 ↪  exist_ok=True)
with (out_dir / (dat.stem + ".dat")).open('w') as f:
   for i, line in enumerate(raw):
      if i in lookup:
         line = pat.sub(lambda _:'{X %.6f, Y %.6f, Z
          ↪  %.6f}' % lookup[i], line, 1)
      f.write(line)

print('Corrected file:', out_dir / (dat.stem + ".dat"))
```

Upon execution, it runs an automated procedure to correct the original robot code, which can be generated, for example, within a commercial offline programming tool such as RoboDK, which natively supports Python-based modules. After identifying the file paths for loading and saving in *Block 1*, *Block 2* reads the input robot code line-by-line, preserving the full file structure while extracting Cartesian coordinate commands formatted as `{X , Y , Z , . . . }` using a regular expression. For each coordinate line, it computes the direction of motion relative to the previous point and aggregates this information along with the payload into a structured feature set. This serves as input to the ML model, loaded from a pickle file in *Block 3*, which predicts the position errors for each direction individually. In *Block 4*, the script computes the corrected coordinates by adding these errors to the original commands. Finally, it reconstructs the robot code by replacing the original coordinate commands with the corrected values in the output file, while leaving other lines unchanged.

5.2. Experimental results

During validation, two sets of new targets were generated within the tested ISO cubic domain, using Latin hypercube sampling and covering different payloads. In the first test, the original training payloads (i.e., 39, 79, and 105 kg) were mounted on the robot, as these will be effectively utilized in the real-world operation of this cell. Each payload was tested at 8 points. Successively, to effectively evaluate the capability of the ML prediction model and, by extension, of the compensation tool, two novel payloads of 65 and 92 kg were tested. These values lie between the original three payloads and were evaluated over 50 target points each, forming the second dataset. The intermediate payloads were derived from the original ones (load 2 and load 3 in Fig. 2) by respectively adding mass to load 2 and removing the fingers and workpiece from load 3.

As the points are approached sequentially during the experiments (from 1 to 24 or 50 in a series of consecutive PTP movements), the effect of the approaching direction is inherently tested. Unlike the training phase, where each position was approached along the principal axes (i.e., $\{(\pm1,0,0),\ (0,\pm1,0),\ (0,0,\pm1)\}$, as specified in Section 4.2), the approach directions in the validation phase involve composite vectors, combining components along the three axes. This setup provides a more realistic and challenging test scenario for the model, closely reflecting the actual conditions encountered during robot operation.

The spatial distribution of the validation points in 3D space, categorized by payload and color-coded for clarity, is shown in Fig. 7a (for 39, 79, and 105 kg) and Fig. 7c (for 65 and 92 kg), while their complete list of coordinates is provided Table 6. The corresponding prediction and compensation results are presented in Fig. 7b and d. These plots demonstrate the effectiveness of the proposed method by comparing the uncorrected and compensated behaviors across all tested targets. Overall, a consistent reduction in AP is observed after compensation. On average, the error is reduced by 91.1%, with the most significant improvements occurring in test-1 (using the original payloads) at point 11, where AP decreased by 98.4%, and in test-2 (using the novel payloads) at point 48 (65 kg) and point 76 (92 kg), with reductions of 96.7% and 97.9%, respectively.

5.3. Discussion

The obtained results confirm that the proposed compensation strategy effectively interpolates to unseen target positions and payload levels within the training domain, while extrapolation outside such domain is not guaranteed and may lead to degraded performance. Despite being trained on a limited number of measured points (i.e., 64 targets for 6 directions and 3 payload levels, leading to 1152 samples), the supervised learning model successfully predicts corrections, maintaining a balance between accuracy, robustness, and applicability. With the selected reduced design grid, each operating configuration, defined by a given tool setup or payload level, requires $4^3 \times 6 = 384$ samples, which can be acquired in approximately 20 min. Consequently, even considering an extreme case of 10 distinct operating configurations for a single robot, the total experimental time (around 3.5 h) would remain practical for industrial deployment. Notably, this level of performance is achieved using relatively simple supervised architectures, without relying on deep or computationally expensive optimization schemes, which typically involve a large number of trainable parameters and require extensive datasets to achieve stable and reliable performance (see, e.g., Refs. [71,72]). This choice ensures robustness and efficiency even under limited-data conditions, a scenario that may arise in industrial contexts where data collection can be expensive, time-consuming, or constrained by operational limitations.

The current version of the framework is tailored to operational scenarios involving pick-and-place, positioning, dispensing, laser processing and inspection tasks, where effects arising from external process forces are minimal. Extending its use to applications with significant interaction forces (e.g., machining, insertion, or interference assembly) would require additional modeling considerations. In particular, incorporating a stiffness model of the manipulator would enable prediction of deflections not only due to gravitational loads but also to external variable loads. As known, the implementation of such model is non-trivial, as joint stiffness parameters are typically proprietary and not

Table 6
Point coordinates and corresponding payloads employed during the validation tests.

Point	Test 1				Test 2							
	x [mm]	y [mm]	z [mm]	Payload [kg]	x [mm]	y [mm]	z [mm]	Payload [kg]	x [mm]	y [mm]	z [mm]	Payload [kg]
1	2063.92	−481.99	762.30	39	1859.54	−254.90	712.02	65	1963.70	−473.79	559.17	92
2	1880.64	306.18	1442.46	39	1843.49	242.15	1228.01	65	2052.29	−210.41	1184.81	92
3	2504.08	−20.46	913.48	39	1913.36	295.15	843.99	65	2244.09	198.48	1253.50	92
4	2209.60	−391.60	1149.55	39	2473.79	437.41	957.90	65	2544.09	391.87	609.56	92
5	2404.56	21.77	535.71	39	2178.86	−24.50	623.57	65	1711.86	−8.60	980.96	92
6	1603.65	288.89	1008.38	39	2024.74	−307.00	1303.18	65	1640.14	406.81	866.62	92
7	1720.99	399.54	961.24	39	1796.39	−285.74	1112.91	65	2085.86	35.63	1210.54	92
8	2319.23	459.62	1249.80	39	2585.33	−18.55	1151.40	65	2229.48	458.38	807.01	92
9	1687.71	−252.03	1467.47	79	2040.87	218.97	1288.56	65	2566.60	130.65	1286.69	92
10	1928.41	−346.41	1386.29	79	2442.31	−191.85	545.98	65	1863.95	−334.57	655.40	92
11	2135.73	56.60	1343.10	79	2279.23	−63.26	1485.15	65	2170.77	−72.28	1343.90	92
12	2583.66	425.57	1176.81	79	1970.17	123.23	984.84	65	2394.33	61.99	1429.94	92
13	2040.31	−137.05	923.20	79	2454.90	396.65	1252.22	65	1916.54	314.94	691.32	92
14	2098.71	145.92	842.91	79	2390.39	−370.04	1191.41	65	1673.19	55.60	565.82	92
15	2254.14	−75.77	584.22	79	2165.30	180.01	1360.87	65	2121.58	495.28	1008.90	92
16	2532.09	−441.46	1120.36	79	2419.43	32.89	648.54	65	1615.93	−436.20	534.90	92
17	1660.83	102.65	1316.83	105	2508.97	−100.94	963.01	65	2320.37	−174.05	1318.50	92
18	1892.74	361.21	729.50	105	2054.86	335.73	1094.47	65	1693.59	−117.57	1408.22	92
19	1793.99	−183.82	1280.91	105	1937.96	−147.96	1467.55	65	1761.25	−414.22	1146.52	92
20	2265.31	−324.75	581.79	105	2252.86	−346.67	1388.23	65	2152.80	−293.22	748.24	92
21	2432.39	−222.63	792.73	105	2510.23	115.41	1440.73	65	2289.82	−348.79	1049.33	92
22	1984.48	230.72	678.07	105	2311.41	81.28	1018.28	65	1990.08	88.74	1470.55	92
23	2371.11	−98.66	660.50	105	2001.81	−213.66	1029.99	65	2264.50	−159.81	1393.52	92
24	1837.04	180.02	1077.17	105	2369.99	220.76	1409.57	65	2469.33	160.85	1091.38	92
25					2559.18	491.24	518.20	65	2346.96	−46.74	1224.70	92
26					2225.85	461.19	1421.91	65	2020.41	−263.11	1320.07	92
27					2205.61	−400.12	922.31	65	2192.22	111.66	1034.70	92
28					1903.48	−272.21	531.22	65	1824.33	−491.09	504.16	92
29					1728.08	−496.02	1204.37	65	2354.32	19.40	797.31	92
30					1815.05	55.34	617.03	65	1946.59	462.67	979.29	92
31					1755.03	−458.65	587.60	65	1655.37	−137.89	950.92	92
32					2139.13	−135.27	1054.79	65	1905.07	−29.94	1368.74	92
33					1643.24	303.47	751.09	65	1888.36	375.81	595.38	92
34					2533.35	8.53	823.12	65	1592.94	−234.43	857.10	92
35					2298.95	−175.85	896.08	65	1736.27	246.01	666.46	92
36					1590.85	−326.58	913.92	65	2490.84	−369.22	772.92	92
37					1652.14	−84.47	870.33	65	2290.30	336.59	1169.79	92
38					1882.87	413.75	1073.24	65	2479.31	−195.89	826.11	92
39					2119.07	−431.69	664.03	65	1831.88	262.23	728.09	92
40					1783.07	−56.60	680.25	65	2520.47	217.15	629.88	92
41					1617.54	274.19	576.82	65	1772.21	−318.79	704.47	92
42					2334.07	−468.05	815.26	65	2589.52	439.99	1266.30	92
43					1955.08	162.76	1265.96	65	2107.43	−248.78	913.61	92
44					2387.49	447.71	732.49	65	1795.19	159.78	1123.36	92
45					2093.55	−239.82	1160.53	65	2422.18	235.31	1446.78	92
46					2089.50	−384.07	1120.55	65	1982.13	−94.14	928.26	92
47					1681.71	143.46	1327.13	65	2137.96	346.12	1079.62	92
48					1695.37	366.48	1342.80	65	2389.52	−445.30	1483.26	92
49					1734.37	75.02	771.96	65	2442.25	−386.53	897.72	92
50					2242.65	350.48	787.36	65	2048.87	298.14	1103.19	92

disclosed by manufacturers. Consequently, they must be identified experimentally or estimated through data-driven methods, which may require the installation of additional force/torque sensors [24,25]. Therefore, at the industrial level, the inclusion of a stiffness model should be considered only when the application requires high accuracy under specific load conditions, as its implementation entails additional time and resource investment.

Furthermore, the current implementation focuses exclusively on positional compensation. In this research, the ML-based error predictor is trained for a single robot orientation, though it can be extended to multiple application-relevant orientations by expanding the training dataset and feature space. Orientation errors could also be incorporated, provided that the adopted measurement system supplies full 6D pose information. This extension would enhance the framework's capability, particularly in applications involving tool alignment or orientation-sensitive tasks. It should also be noted that the proposed strategy is designed for PTP correction, rather than continuous path compensation. Although a complex path could, in principle, be discretized into a sequence of closely spaced PTP motions, this approach is generally inefficient for industrial controllers, as it increases computational load and likely degrade motion performance. Nonetheless, the actual ML-based error predictor can effectively support path planning, offering insight into regions of the workspace where the robot is more prone to positioning deviations. Such information can be exploited during offline path generation to improve accuracy by design, even though real-time path correction would still require integration at lower control layers and closed-loop feedback mechanisms.

6. Conclusions

This study presents an ML-based engineering tool for enhancing IR position accuracy by combining laser tracker technology with ML predictive models. The tool, implemented in Python to ensure cross-platform applicability, can be integrated within commercial offline robotic programming platforms and Digital Twins to predict directional

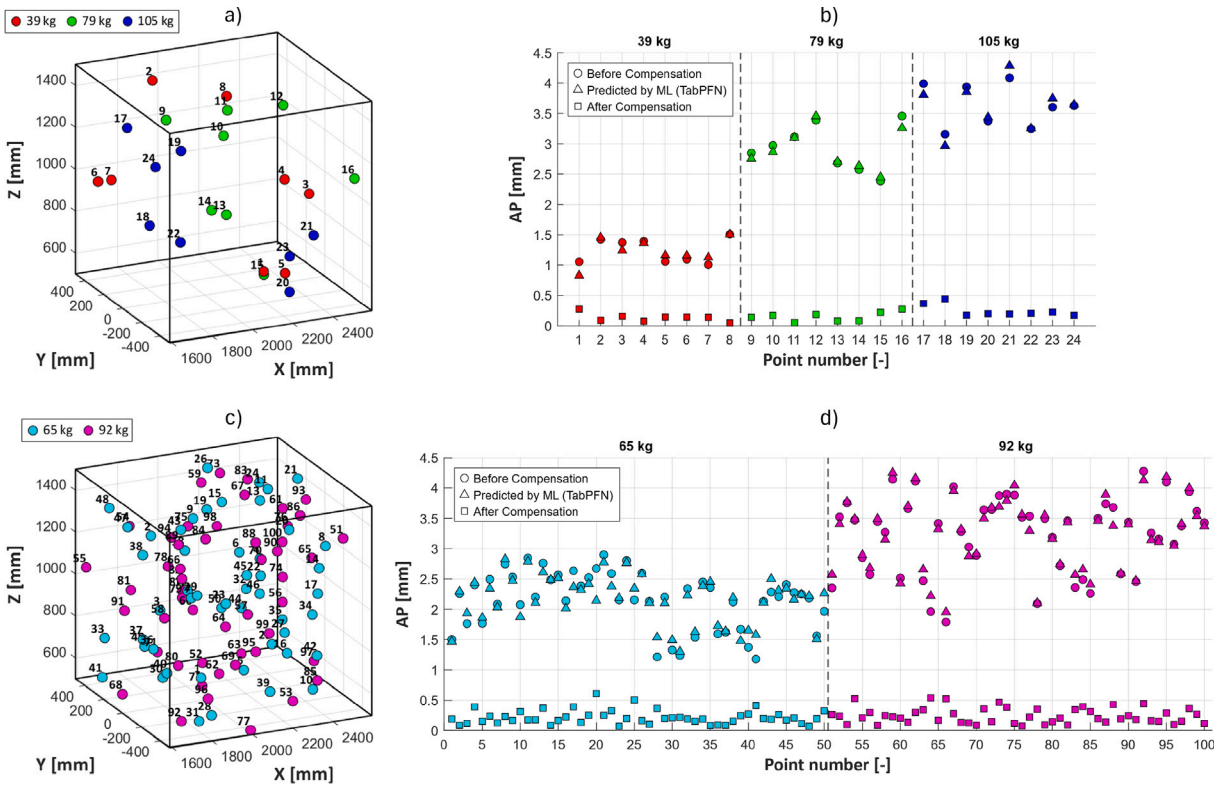

Fig. 7. Validation of the compensation framework across two test datasets: (a) 3D distribution of 24 test poses for test 1 with original payloads (39, 79, 105 kg); (b) comparison of uncorrected, ML-predicted, and compensated errors for test 1; (c) 3D distribution of 100 test poses for test 2 with new payloads (65, 92 kg); (d) comparison of uncorrected, ML-predicted, and compensated errors for test 2.

position errors across the robot spatial domain and update target coordinates without altering program logic. After an initial overview of the adopted compensation framework, the paper details the experimental setup used for development and validation, involving a high-payload KUKA robot and a FARO laser tracker. An ISO 9283-based sensitivity analysis was conducted to assess the influence of key operating parameters (speed, payload, approach direction, and point location) on the error index AP. Results showed that AP is primarily affected by payload and approach direction, while speed in the 10%–50% range has minimal impact. These findings guided the construction of balanced training datasets to ensure that the ML models captured the dominant error sources.

A comparative evaluation of several ML models showed that TabPFN achieved the highest generalization performance on small-to-medium datasets without requiring any hyperparameter tuning. In contrast, the MAE and RMSE values of the other models ranged from approximately 50% to over 400% higher than those of TabPFN, with training times spanning from 20 s to more than 4000 s. Regarding dataset size, a dedicated study revealed that a uniform spatial grid with granularity $d = 4$ (64 points within the cube), combined with six approach directions and three payload levels (yielding 1152 samples), achieved nearly the same predictive accuracy as denser grids, while reducing data acquisition time on the physical cell from 3.5 h to just 1 h. Validation tests conducted on the KUKA robot using novel datasets sampled with Latin Hypercube confirmed the tool's effectiveness, reducing AP by up to 98.4%, i.e. from 2.38 mm to 0.03 mm. Once trained on the 64-point grid and test set, the robot can operate long-term with improved accuracy. Since the robot's kinematic configuration remains unchanged, the same model remains valid even when joint-1 is rotated, extending applicability across different cell layouts.

Future work will focus on extending the framework to scenarios involving external process forces and orientation-dependent tasks, through the integration of manipulator stiffness models and 6D measurement data. Validation campaigns are also planned on IR from different manufacturers, in collaboration with industrial partners. To facilitate the reproduction of the results and support further developments, the research data related to this work are made available through an online repository linked in the supplementary material.

CRediT authorship contribution statement

Giuseppe Romano: Validation, Software, Methodology, Investigation, Data curation. **Pietro Bilancia:** Writing – original draft, Supervision, Methodology, Conceptualization. **Alberto Locatelli:** Writing – original draft, Data curation, Conceptualization. **Mirko Mucciarini:** Software, Formal analysis, Data curation. **Manuel Iori:** Writing – original draft, Supervision, Formal analysis. **Marcello Pellicciari:** Writing – original draft, Supervision, Conceptualization.

Declaration of competing interest

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

Acknowledgment

We thank seven anonymous reviewers for their helpful comments.

Appendix A. Supplementary material

https://data.mendeley.com/datasets/6k37n6v5k5/1 (DOI: 10.17632/6k37n6v5k5.1)

Data availability

Data will be made available on request.

References

[1] T. Zhang, F. Peng, Z. Yang, X. Tang, J. Yuan, R. Yan, Digital twin-driven staged error prediction and compensation framework for the whole process of robotic machining, J. Manuf. Syst. 83 (2025) 252–283.

[2] C. Song, Z. Liu, X. Wang, T. Qiu, Z. Liang, W. Shen, Y. Gao, S. Ma, Research on posture optimization and accuracy compensation technology in robotic side milling, Meas. Sci. Technol. 35 (12) (2024) 125004.

[3] Y. Jiang, Z. Huang, B. Yang, W. Yang, A review of robotic assembly strategies for the full operation procedure: planning, execution and evaluation, Robot. Comput.-Integr. Manuf. 78 (2022) 102366.

[4] S. Asif, M. Bueno, P. Ferreira, P. Anandan, Z. Zhang, Y. Yao, G. Ragunathan, L. Tinkler, M. Sotoodeh-Bahraini, N. Lohse, et al., Rapid and automated configuration of robot manufacturing cells, Robot. Comput.-Integr. Manuf. 92 (2025) 102862.

[5] Y. Gao, T. Qiu, C. Song, S. Ma, Z. Liu, Z. Liang, X. Wang, Optimizing the performance of serial robots for milling tasks: A review, Robot. Comput.-Integr. Manuf. 94 (2025) 102977.

[6] R.A. Boby, A. Klimchik, Combination of geometric and parametric approaches for kinematic identification of an industrial robot, Robot. Comput.-Integr. Manuf. 71 (2021) 102142.

[7] I.O. for Standardization, ISO 9283:1998 - Manipulating industrial robots – Performance criteria and related test methods, 1998, Available at: https://www.iso.org/standard/31187.html.

[8] Y. Liu, J. Zhou, Y. Li, Y. Zhang, Y. He, J. Wang, A high-accuracy pose measurement system for robotic automated assembly in large-scale space, Measurement 188 (2022) 110426.

[9] B. Tipary, G. Erdős, Tolerance analysis for robotic pick-and-place operations, Int. J. Adv. Manuf. Technol. 117 (5) (2021) 1405–1426.

[10] Y. He, X. Wang, Path planning of multiple spot-welding digital twin robots based on reinforcement pointer network, Int. J. Adv. Manuf. Technol. (2025) 1–16.

[11] N. Shen, Z. Guo, J. Li, L. Tong, K. Zhu, A practical method of improving hole position accuracy in the robotic drilling process, Int. J. Adv. Manuf. Technol. 96 (5) (2018) 2973–2987.

[12] A. Verl, A. Valente, S. Melkote, C. Brecher, E. Ozturk, L.T. Tunc, Robots in machining, CIRP Ann 68 (2) (2019) 799–822.

[13] M. Schmitz, J. Wiartalla, M. Gelfgren, S. Mann, B. Corves, M. Hüsing, A robot-centered path-planning algorithm for multidirectional additive manufacturing for WAAM processes and pure object manipulation, Appl. Sci. 11 (13) (2021) 5759.

[14] W. Ma, T. Hu, C. Zhang, T. Zhang, A robot motion position and posture control method for freeform surface laser treatment based on NURBS interpolation, Robot. Comput.-Integr. Manuf. 83 (2023) 102547.

[15] Z. Liao, S. Li, F. Xie, G. Yang, X. Lin, Z. Xu, X. Zhou, A physical-simulation synergy approach for high-uniformity robotic gluing, Robot. Comput.-Integr. Manuf. 94 (2025) 102961.

[16] R. Li, N. Ding, Y. Zhao, H. Liu, Real-time trajectory position error compensation technology of industrial robot, Measurement 208 (2023) 112418.

[17] S. Ferrarini, P. Bilancia, R. Raffaeli, M. Peruzzini, M. Pellicciari, A method for the assessment and compensation of positioning errors in industrial robots, Robot. Comput.-Integr. Manuf. 85 (2024) 102622.

[18] T.A. Khaled, O. Akhrif, I.A. Bonev, Dynamic path correction of an industrial robot using a distance sensor and an ADRC controller, IEEE/ASME Trans. Mechatronics 26 (3) (2020) 1646–1656.

[19] Z. Wang, R. Zhang, P. Keogh, Real-time laser tracker compensation of robotic drilling and machining, J. Manuf. Mater. Process. 4 (3) (2020) 79.

[20] Q. Bai, P. Li, W. Tian, J. Shen, B. Li, J. Hu, Vision guided dynamic synchronous path tracking control of dual manipulator cooperative system, J. Manuf. Sci. Eng. 145 (12) (2023) 121003.

[21] W. Wang, W. Tian, W. Liao, B. Li, J. Hu, Error compensation of industrial robot based on deep belief network and error similarity, Robot. Comput.-Integr. Manuf. 73 (2022) 102220.

[22] L. Bo, T. Wei, H. Fangfang, C. Guangyu, L. Yufei, et al., Positioning error compensation of an industrial robot using neural networks and experimental study, Chin. J. Aeronaut. 35 (2) (2022) 346–360.

[23] A. Le Reun, K. Subrin, A. Dubois, S. Garnier, Thermal drift and backlash issues for industrial robots positioning performance, Robotica 40 (9) (2022) 2933–2952.

[24] J. Selvaganesan, E. Paul, A. Klimchik, N. Arunachalam, A comprehensive framework for stiffness modeling of industrial robot for machining applications, J. Manuf. Process. 152 (2025) 1111–1122.

[25] A. Klimchik, S. Caro, Y. Wu, D. Chablat, B. Furet, A. Pashkevich, Stiffness modeling of robotic manipulator with gravity compensator, in: Computational Kinematics: Proceedings of the 6th International Workshop on Computational Kinematics, CK2013, Springer, 2013, pp. 185–192.

[26] T. Chen, W. Yang, S. Li, X. Luo, Data-driven calibration of industrial robots: A comprehensive survey, IEEE/CAA J. Autom. Sin. 12 (8) (2025) 1544–1567.

[27] K. Deng, D. Gao, S. Ma, C. Zhao, Y. Lu, Elasto-geometrical error and gravity model calibration of an industrial robot using the same optimized configuration set, Robot. Comput.-Integr. Manuf. 83 (2023) 102558.

[28] H. Pan, Y. Cai, B. Jia, L. Li, L. Chen, Improving the absolute positioning accuracy of industrial robots based on OP-ELM and an enhanced backtracking search algorithm, Intell. Serv. Robot. 18 (2) (2025) 247–260.

[29] B. Jia, H. Pan, Y. Cai, L. Zhang, X. Chen, L. Chen, Enhancing the absolute positioning accuracy of welding robots based on joint error compensation, Eng. Appl. Artif. Intell. 158 (2025) 111302.

[30] X. Zhu, H. Zhang, Z. Liu, C. Cai, L. Fu, M. Yang, H. Chen, Deep learning-based interpretable prediction and compensation method for improving pose accuracy of parallel robots, Expert Syst. Appl. 268 (2025) 126289.

[31] J. Zhang, J. Lin, Y. Gao, Z. Wang, F. Xu, J. Zhu, Efficient positioning error compensation for robots in wire arc hybrid manufacturing systems, Robot. Comput.-Integr. Manuf. 95 (2025) 103040.

[32] T. Zhang, F. Peng, R. Yan, X. Tang, R. Deng, J. Yuan, Quantification of uncertainty in robot pose errors and calibration of reliable compensation values, Robot. Comput.-Integr. Manuf. 89 (2024) 102765.

[33] C.M. Bishop, Neural networks and their applications, Rev. Sci. Instrum. 65 (6) (1994) 1803–1832.

[34] L. Grinsztajn, E. Oyallon, G. Varoquaux, Why do tree-based models still outperform deep learning on typical tabular data? Adv. Neural Inf. Process. Syst. 35 (2022) 507–520.

[35] N. Sünderhauf, O. Brock, W. Scheirer, R. Hadsell, D. Fox, J. Leitner, B. Upcroft, P. Abbeel, W. Burgard, M. Milford, et al., The limits and potentials of deep learning for robotics, Int. J. Robot. Res. 37 (4–5) (2018) 405–420.

[36] RoboDK, Robot Calibration Package, 2025, (Accessed 08 August 2025). https://robodk.com/robot-calibration.

[37] P. Bilancia, S. Ferrarini, R. Berni, M. Pellicciari, Assessing path accuracy in industrial robots via ballbar technology, Ind. Robot.: Int. J. Robot. Res. Appl. 52 (4) (2025) 477–490.

[38] M. Slamani, A. Nubiola, I.A. Bonev, Modeling and assessment of the backlash error of an industrial robot, Robotica 30 (7) (2012) 1167–1175.

[39] N. Hollmann, S. Müller, L. Purucker, A. Krishnakumar, M. Körfer, S.B. Hoo, R.T. Schirrmeister, F. Hutter, Accurate predictions on small data with a tabular foundation model, Nature 637 (8045) (2025) 319–326.

[40] S. Ruiz-Villafranca, J. Roldán-Gómez, J.M.C. Gómez, J. Carrillo-Mondéjar, J.L. Martinez, A TabPFN-based intrusion detection system for the industrial internet of things, J. Supercomput. 80 (14) (2024) 20080–20117.

[41] P. Bilancia, A. Locatelli, A. Tutarini, M. Mucciarini, M. Iori, M. Pellicciari, Online motion accuracy compensation of industrial servomechanisms using machine learning approaches, Robot. Comput.-Integr. Manuf. 91 (2025) 102838.

[42] M. Slamani, A. Joubair, I.A. Bonev, A comparative evaluation of three industrial robots using three reference measuring techniques, Ind. Robot.: Int. J. 42 (6) (2015) 572–585.

[43] P. Bilancia, J. Schmidt, R. Raffaeli, M. Peruzzini, M. Pellicciari, An overview of industrial robots control and programming approaches, Appl. Sci. 13 (4) (2023) 2582.

[44] A.A. Hayat, R.A. Boby, S.K. Saha, A geometric approach for kinematic identification of an industrial robot using a monocular camera, Robot. Comput.-Integr. Manuf. 57 (2019) 329–346.

[45] P. Aivaliotis, Z. Arkouli, K. Georgoulias, S. Makris, Degradation curves integration in physics-based models: Towards the predictive maintenance of industrial robots, Robot. Comput.-Integr. Manuf. 71 (2021) 102177.

[46] R. API, Robot Application Programming Interfaces, 2025, (Accessed 08 August 2025). https://robodk.com/doc/en/RoboDK-API.html.

[47] L. Du Peloux, M. Fago, KUKAVARPROXY server, 2025, (Accessed 08 August 2025). https://github.com/ImtsSrl/KUKAVARPROXY.git.

[48] ABB, Robot web services, 2025, (Accessed 08 August 2025). https://developercenter.robotstudio.com/api/rwsApi/.

[49] M. Gadaleta, G. Berselli, M. Pellicciari, F. Grassia, Extensive experimental investigation for the optimization of the energy consumption of a high payload industrial robot with open research dataset, Robot. Comput.-Integr. Manuf. 68 (2021) 102046.

[50] J. Santolaria, J. Conte, M. Ginés, Laser tracker-based kinematic parameter calibration of industrial robots by improved CPA method and active retroreflector, Int. J. Adv. Manuf. Technol. 66 (9) (2013) 2087–2106.

[51] P. Bilancia, S. Ferrarini, R. Berni, M. Pellicciari, Assessing path accuracy in industrial robots via ballbar technology, Ind. Robot.: Int. J. Robot. Res. Appl. (2024).

[52] K.-T. Fang, D.K. Lin, P. Winker, Y. Zhang, Uniform design: theory and application, Technometrics 42 (3) (2000) 237–248.

[53] W.-L. Loh, On latin hypercube sampling, Ann. Stat. 24 (5) (1996) 2058–2080.

[54] B. Li, W. Tian, C. Zhang, F. Hua, G. Cui, Y. Li, Positioning error compensation of an industrial robot using neural networks and experimental study, Chin. J. Aeronaut. 35 (2) (2022) 346–360.

[55] X. Zhu, Z. Liu, C. Cai, M. Yang, H. Zhang, L. Fu, J. Zhang, Deep learning-based predicting and compensating method for the pose deviations of parallel robots, Comput. Ind. Eng. 191 (2024) 110179.

[56] M. Bai, M. Zhang, H. Zhang, M. Li, J. Zhao, Z. Chen, Calibration method based on models and least-squares support vector regression enhancing robot position accuracy, IEEE Access 9 (2021) 136060–136070.
[57] T. Brown, B. Mann, N. Ryder, M. Subbiah, J.D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, D. Amodei, Language models are few-shot learners, in: H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, H. Lin (Eds.), Advances in Neural Information Processing Systems, Vol. 33, 2020, pp. 1877–1901.
[58] H. Zou, T. Hastie, Regularization and variable selection via the elastic net, J. R. Stat. Soc. Ser. B Stat. Methodol. 67 (2) (2005) 301–320.
[59] A.J. Smola, B. Schölkopf, A tutorial on support vector regression, Stat. Comput. 14 (3) (2004) 199–222.
[60] A. Natekin, A. Knoll, Gradient boosting machines, a tutorial, Front. Neurorobotics 7 (DEC) (2013).
[61] T. Chen, C. Guestrin, XGBoost: A scalable tree boosting system, in: Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining 13-17-August-2016, 2016, pp. 785–794.
[62] L. Prokhorenkova, G. Gusev, A. Vorobev, A.V. Dorogush, A. Gulin, CatBoost: unbiased boosting with categorical features, Adv. Neural Inf. Process. Syst. 31 (2018).
[63] S. Müller, N. Hollmann, S. Pineda, J. Grabocka, F. Hutter, Transformers can do Bayesian inference, in: ICLR 2022 - 10th International Conference on Learning Representations, 2022.
[64] T. Zhang, H. Sun, F. Peng, X. Tang, R. Yan, R. Deng, An online prediction and compensation method for robot position errors embedded with error-motion correlation, Meas.: J. Int. Meas. Confed. 234 (2024).
[65] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: Machine learning in python, J. Mach. Learn. Res. 12 (2011) 2825–2830.
[66] Pedregosa, et al., Scikit-learn: Machine learning in python, 2010, URL: https://scikit-learn.org/stable/index.html.
[67] S. Arlot, A. Celisse, A survey of cross-validation procedures for model selection, Stat. Surv. 4 (2010) 40–79.
[68] F. Petropoulos, D. Apiletti, V. Assimakopoulos, M.Z. Babai, D.K. Barrow, S.B. Taieb, C. Bergmeir, R.J. Bessa, J. Bijak, J.E. Boylan, et al., Forecasting: theory and practice, Int. J. Forecast. 38 (3) (2022) 705–871.
[69] W. Wang, W. Tian, W. Liao, B. Li, J. Hu, Error compensation of industrial robot based on deep belief network and error similarity, Robot. Comput.-Integr. Manuf. 73 (2022) 102220.
[70] T. Akiba, S. Sano, T. Yanase, T. Ohta, M. Koyama, Optuna: A next-generation hyperparameter optimization framework, in: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2019, pp. 2623–2631.
[71] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, 2016.
[72] F.J. Moreno-Barea, J.M. Jerez, L. Franco, Improving classification accuracy using data augmentation on small data sets, Expert Syst. Appl. 161 (2020) 113696.