Deep Learning for Integro-Differential Modelling

François Lemaire, Louis Roussel

Deep Learning for Integro-Differential Modelling *

François Lemaire[1] and Louis Roussel[1]

Univ. Lille, CNRS, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France
{francois.lemaire,louis.roussel}@univ-lille.fr

Abstract

We investigate the use of deep learning techniques for computing integral equations from systems of nonlinear differential equations. The class of integral equations we consider are very general and difficult to manipulate with purely symbolic computations. In order to rapidly explore and experiment with such complicated expressions, we have trained deep learning models to get valuable insight on the type of expressions that can be expected to be calculated using symbolic methods. The first contribution is a novel algorithm for converting an integral equation into a differential equation. The second is the discovery and resolution, via our trained models, of systems which are currently beyond the reach of classical computer algebra.

1 Introduction

Many models in biology and control theory are written as systems of non differential equations involving unknown parameters. Under suitable assumptions, partial data measurements are sufficient to estimate the values of the parameters. There exists an extensive literature on the parameter estimation problem (see for example [27] and references therein). This article is motivated by the parameter estimation problem using both computer algebra and deep learning techniques.

We illustrate the parameter estimation problem on the simple example from Figure 1 which only involves two variables $x(t)$ and $y(t)$ and a parameter θ. Assuming that experimental data are only available for $y(t)$ (i.e. $x(t)$ is unknown), can we estimate the value of the parameter θ? One classical approach [14] consists of eliminating $x(t)$ from the system S to obtain Input/Output (I/O) equations only involving θ and $y(t)$. This elimination is usually performed using computer algebra algorithms, which are designed for symbolically manipulating equations. Figure 1 presents two I/O equations $f = 0$ and $F = 0$ obtained by eliminating $x(t)$ from S. For brevity, in the rest of the paper, the term Equation f (resp. F) actually means Equation $f = 0$ (resp. $F = 0$). Equation f is a differential I/O equation, while F is an integral I/O equation where the $\int$ is the primitive operator (i.e. $\int u = \int_0^t u(\tau)\mathrm{d}\tau$). Those two I/O equations may be used to estimate the value of θ using the experimental data for $y(t)$.

In case of noisy experimental data (i.e. in case of errors on the measurements), Equation F is usually more suitable since the integration tends to filter the noise (see [6, 21] for a numerical treatment using integral equations, and see [20, Section 7] for examples of integral equations computed from biological examples).

In Figure 1, Equation F can be computed in two ways: either by differential elimination (Algorithm DE) followed by integration (Algorithm INT), or directly by integral elimination (Algorithm IE). Although the DE algorithm is complete (solid arrow on Figure 1), INT and IE are partial algorithms (dashed arrows) implying that integral I/O equations cannot always be computed for general systems.

*accepted to SCML: A Publishing Forum for Symbolic Computation and Machine Learning. https://scml.risc.jku.at/

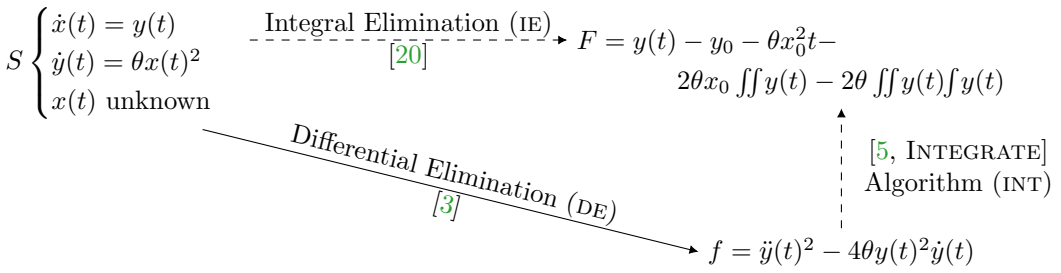

Figure 1: Computing the differential I/O equation $f = 0$ and the integral I/O equation $F = 0$ from the system S.

Improving Algorithms IE and INT is a very complex and time-consuming task. For this reason, we experiment in this article with the use of deep learning techniques to compute integral I/O equations and help improve these algorithms. This idea of using deep learning for getting mathematical insight in difficult problems has already been proved successful for instance for knot theory [10] and algebraic geometry [12].

The contributions presented in this paper demonstrate that the use of deep learning techniques to provide improvement suggestions is a promising approach for integro-differential elimination.

Indeed, thanks to the development of a new algorithm I2D (Section 3.2), we have *trained 4 models* that revealed and solved *new striking systems* that Algorithms IE and INT could not solve (Section 7.2). Those new examples constitute important results since they seem very difficult to create by hand computations. Moreover, those new examples are already a source of inspiration for improving Algorithm IE since logarithms were added in [22] following this work.

The source code associated to the experiment presented in this paper is available at `https://codeberg.org/louis-roussel/IntegroDifferentialDeepLearning`.

Organisation of the Paper Section 3 presents classical algorithms in computer algebra, and presents our new algorithm I2D (which converts an integral equation to a differential equation) along with its associated deep learning model [I2D_b]. Section 4 details the data generation process. Section 5 presents our experiment and the different trained models. Section 6 presents the results of our experiment that are discussed in Section 7. Finally, Sections 7.3 and 8 present some limitations of this work and conclude.

Notations Dotted arrows inside the figures represent trained models. Model names are enclosed with square brackets (e.g. [IE]). The variables $x(t)$ and $y(t)$ denote time functions. First, second and third derivatives of $x(t)$ (resp. $y(t)$) are denoted $\dot{x}(t)$, $\ddot{x}(t), \dddot{x}(t)$ (resp. $\dot{y}(t)$, $\ddot{y}(t), \dddot{y}(t)$). Differentiation w.r.t. t is denoted d/dt or with a single quote ($'$). For brevity, the time dependency is sometimes omitted (e.g. $\dot{x}(t) = x(t) + y(t)$ is simply written $\dot{x} = x + y$). The primitive $\int_0^t u(\tau)\mathrm{d}\tau$ of a real function $u(t)$ is simply denoted $\int u$. A plus sign between two algorithms denote the sequence of the two algorithms (e.g. DE + INT means Algorithm DE followed by INT).

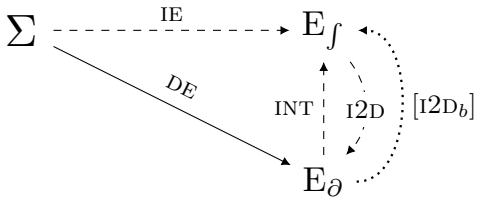

Figure 2: The general computation schemes for integro-differential modelling. Dashed lines represent partial algorithms. The dotted line corresponds to the deep learning model [I2D$_b$] (see Section 3.4).

2 Related Work

The seminal paper [19] shows that Transformer-based [26] architectures can learn symbolic integration, differentiation, and solve differential equations. The authors of [19] obtain very good results for some datasets despite some limitations as detailed in [13].

More recently, there has also been a growing number of work that combine Symbolic Computation and Machine Learning, such as computing Gröbner bases with Transformers [18], finding suitable elimination orders for CAD [24], or selecting strategies for symbolic integration [2].

Symbolic regression has been developed in [15, 16] using genetic algorithms, and in [17, 11] using Transformers. See [9] for an overview of symbolic regression methods. Integral equations could likely be computing using symbolic regressions. However some preliminary tests we conducted using genetic algorithms were not conclusive.

In this paper, we restrict ourselves to the classical Transformer architecture. Variants such as Tree Transformer should be tested, following the work of [25, 1].

3 Algorithms and the Model [I2D$_b$]

Figure 1 is a particular case of the computation approach depicted of Figure 2. The Σ symbol denotes the set of all differential systems of the form $\begin{cases} \dot{x}(t) = p(x(t), y(t)) \\ \dot{y}(t) = q(x(t), y(t)) \end{cases}$ where p and q are polynomials over $\mathbb{Q}$. The set E_∂ contains all differential polynomials in $y(t)$, i.e. polynomial expressions in $y(t)$, $\dot{y}(t)$, $\ddot{y}(t)$, ... The set $\mathrm{E}_\int$ is the set of all expressions which can be obtained from the rational numbers and $y(t)$, and the operations sum, subtraction, division, product, integration, exponential and logarithm. More precisely, the set $\mathrm{E}_\int$ contains $\mathbb{Q}$ and $y(t)$, and for any expressions F and G in $\mathrm{E}_\int$, the expressions $F+G$, $F-G$, F/G, $F \times G$, $\int F$, e^F and $\ln(F)$ are also in $\mathrm{E}_\int$. For example, the expression $F = y^2 - y + y\int y + \ln(y) + \int(\frac{\int(y)}{y}) + \iint y^2$ (see Example 1) belongs to $\mathrm{E}_\int$.

Figure 2 reveals three different paths from Σ to $\mathrm{E}_\int$ which will be used in Section 4 to generate data couples (S, F) in $\Sigma \times \mathrm{E}_\int$.

After presenting algorithms DE, INT and IE in Section 3.1, we introduce a new algorithm I2D in Section 3.2. Section 3.3 details how to check that an element of $\mathrm{E}_\int$ is an integral I/O equations of a system S in Σ. Section 3.4 presents the model [I2D$_b$] (where the suffix b stands for backwards).

3.1 Known Integro-Differential Algorithms

Algorithm DE The differential elimination from Σ to E_∂ is completely algorithmic, and is performed using the ROSENFELD-GRÖBNER Algorithm [8] and its implementation [3].

Algorithm INT Algorithm [5, INTEGRATE (Algorithm 4)], simply denoted INT, is a partial algorithm that can be used to integrate several times an element of E_∂ to produce an element of $\mathrm{E}_\int$. Only polynomials are obtained (i.e. no fractions, exponential or logarithmic terms are introduced). Algorithm INT is not able to compute integrating factors which may be necessary for a successful integration. For instance, the equation $\dot{x}y + \dot{x} - xy$ needs to be multiplied by $\frac{1}{(y+1)^2}$ before being integrated to $\frac{x}{y+1}\cdot$ When it succeeds, the output of INT is always correct and does not need to be verified.

Algorithm IE Algorithm [20, INTEGRAL_ELIMINATION], simply denoted IE, is a partial algorithm for directly computing an element of $\mathrm{E}_\int$ from a system in Σ written in integral form. Equations computed using IE can involve exponentials but no fractions nor logarithms. When it succeeds, the output of IE is always correct and does not need to be verified.

3.2 A New Algorithm I2D

Our algorithm I2D (Algorithm 1) is a partial algorithm that converts using divisions and differentiations an expression in $\mathrm{E}_\int$ into an expression in E_∂ by removing the integrals, exponentials and logarithms. This algorithm will be used to check if an element of $\mathrm{E}_\int$ is a consequence of a system $S \in \Sigma$ (Section 3.3). It will also be used to create data to train the model [$\mathrm{I2D}_b$] which transforms an element of E_∂ into an element of $\mathrm{E}_\int$ (Section 3.4).

The idea of I2D is the following. Consider an expression $F \in \mathrm{E}_\int$ of the form $F = p(\int G)e^H \ln(K)^\alpha + \cdots$ where $p \in \mathrm{E}_\partial$, α positive integer and $G, H, K \in \mathrm{E}_\int$. In order to obtain an expression in E_∂, Algorithm I2D tries to repeatedly remove the integral in front of G, remove the exponential from the term e^H or decrease the exponent α by 1. Algorithm I2D terminates when the expression F is free of integral, exponential and logarithm.

More precisely, Algorithm I2D repeats one of the three following computations (and then takes the numerator) until an equation in E_∂ is obtained. The first one removes the integral in front of G by replacing F by $(\frac{F}{pe^H \ln(K)^\alpha})' = G + \cdots$. The second one removes the exponential in e^H by replacing F by $(\ln(\frac{F}{p(\int G)\ln(K)^\alpha}))' = H' + \cdots$. The third one decreases α by 1 by replacing F by $(\frac{F}{p(\int G)e^H})' = \alpha\frac{K'}{K}\ln(K)^{\alpha-1} + \cdots$.

Algorithm I2D suffers limitations. It may fail for certain inputs, it may not terminate, and the output is not guaranteed "correct". Those limitations are due to the following reasons.

First, the expressions in $\mathrm{E}_\int$ are very difficult to manipulate. Indeed, even simply checking that an expression in $\mathrm{E}_\int$ can be simplified to zero might be undecidable due to the undecidable *identity problem* by Richardson [23, Theorem Two]. Second, the treatments at Lines 14-19 need to be improved to properly handle some checks on the possible cancellation of Q (used to divide R), which requires considering case splits when dividing by zero. Considering a system of equations could help solve this problem, but this is left for future work.

Example 1. *We summarise the main steps of Algorithm* I2D *applied to* $F = y^2 - y + y\int y + \ln(y) + \int(\frac{\int(y)}{y}) + \iint y^2$. *Underlined terms corresponds the expression* M *selected at Line 10.*

$$
\begin{aligned}
& y^2 - y + y\int y + \ln(y) + \underline{\int(\tfrac{\int y}{y})} + \iint y^2 \\
& \quad \Big\downarrow \tfrac{d}{dt} \text{ then take the numerator} \\
& (y\dot{y} + 1)\int y + \underline{y\int y^2} + \dot{y} + y^3 - y\dot{y} + 2y^2\dot{y} \\
& \quad \Big\downarrow \times\tfrac{1}{y} \text{ then } \tfrac{d}{dt} \text{ then take the numerator} \\
& y^4 + 3y^3\dot{y} + 2y^3\ddot{y} + \underline{(y^2\ddot{y} - \dot{y})\int y} - y^2\ddot{y} + y^2 + y\ddot{y} - \dot{y}^2 + 2y^2\dot{y}^2 \\
& \quad \Big\downarrow \times\tfrac{1}{y^2\ddot{y}-\dot{y}} \text{ then } \tfrac{d}{dt} \text{ then take the numerator} \\
& -y^5\dddot{y} + 2y^4\dot{y}\ddot{y} - 3y^4\dot{y}\dddot{y} + 4y^4\ddot{y}^2 + 3y^3\dot{y}^2\ddot{y} - 2y^3\dot{y}^2\dddot{y} + 6y^3\dot{y}\ddot{y}^2 + y^3\ddot{y} + \cdots .
\end{aligned}
$$

3.3 Checking an I/O Integral Equation

Checking whether a differential I/O equation f is a consequence of a differential system S in Σ can be solved by checking that the normal form [4] of f w.r.t. S equals zero. Technically, this comes from the fact that S is already a regular differential chain [7] for a suitable ranking (i.e. the orderly ranking $x < y < \dot{x} < \dot{y} < \cdots$).

However, checking whether an integral I/O equation F is a consequence of a differential system S is a delicate problem, and still an open question to our knowledge. A partial answer to this problem consists of using our algorithm I2D from Section 3.2. We convert F to a differential equation f using I2D, and we check if f is a consequence of S (as in the previous paragraph). We explain below that this process is not completely satisfactory since it will only detect whether F is a consequence of S up to some extra terms involving integration constants.

Consider the system $S = \begin{cases} \dot{x} = p(x,y) \\ \dot{y} = y \end{cases}$ and $F = \frac{1}{y} + \int\frac{1}{y}$. Computations show that $f = \text{I2D}(F) = -\dot{y} + y$ which is obviously a consequence of S. However F cannot be obtained from S for the following reason: any solution of S satisfies $y(t) = \lambda e^t$ but plugging any such solution in F yields $1/\lambda$ and not zero. This is expected, since dividing $y' - y$ by y^2 and integrating yields $\frac{1}{y} - \frac{1}{y_0} + \int\frac{1}{y}$ which is equal to F up to the term $\frac{1}{y_0}$.

For more complex examples, see Section 7.2 where the F_{ics} expressions are obtained from f by hand by carefully handling initial conditions. For future work, we might improve the I2D algorithm so that it both computes f, and a corrected version F_{ics} of F involving integration constants.

For very particular systems, it is sometimes necessary to differentiate f once or twice before checking that it is a consequence of S. For example, consider $F = y - t$ and $S = \begin{cases} \dot{x} = p(x,y) \\ \dot{y} = 1 \end{cases}$, for some polynomial p. Algorithm I2D(F) returns F itself. The normal form of F w.r.t. S is F itself (thus nonzero), whereas the normal of $F' = \dot{y} - 1$ w.r.t. S is zero. The same difficulty occurs for $F = y - t^2/2$ and $S = \begin{cases} \dot{x} = 1 \\ \dot{y} = x \end{cases}$, where F needs to be differentiated twice.

Summarising the previous arguments yields Algorithm CheckIOInt (Algorithm 2), which takes as input an integral I/O equation $F \in \mathrm{E}_{\int}$ and a dynamical system $S \in \Sigma$ and outputs True if F is a consequence of S up to some terms involving initial conditions. Algorithm CheckIOInt suffers the same limitations as I2D.

Algorithm 1: I2D(F)

Input: $F \in \mathrm{E}_{\int}$
Output: Return either FAIL, or $f \in \mathrm{E}_{\partial}$. See Section 3.2.

begin
  $f \leftarrow F$;
  while f *contains* $\int$, exp *or* ln *(i.e.* $f \notin \mathrm{E}_{\partial}$*)* **do**
    $f \leftarrow$ the numerator of f;
    expand f using the distributivity of $\times$ (i.e. using $(A+B)\times C = A\times C + B\times C$);
    expand in f any product of integrals using the rule $(\int U)\times(\int V) = \int U\int V + \int V\int U$ [20, Section 2.1];
    try to write f as a sum $\sum_{i=1..n} p_i I_i E_i L_i$, where the p_i are differential polynomials; the I_i, E_i and L_i are either equal to 1 or are respectively expressions of the form $\int A$, e^A and $\ln(A)^\alpha$ where $A \in \mathrm{E}_{\int}$ and α is a positive integer;
    return FAIL if it is not possible;
    return p_1 if $n = 1$;
    Let M be a greatest expression among all the I_i, E_i or L_i according to the lexicographic ordering $(\#\int, \#\exp, \#\ln)$.
    `// More precisely, choose an expression` M `with the highest number of integral operators; or in case of ties, an expression with the highest number of exponential operators; or in case of ties, an expression with the highest number of logarithm operators.`
    By collecting f by M, we write $f = Q\times M + R$;
    `// The case` $Q = 0$ `is excluded in this version of the algorithm.`
    if $R = 0$ **then**
      $f \leftarrow$ I2D(Q) $\times$ I2D(M)
    else if M *can be written as* $\int A$ **then**
      $f \leftarrow A - (\frac{-R}{Q})'$;
    else if M *can be written as* e^A **then**
      $f \leftarrow A' - (\ln(\frac{-R}{Q}))'$;
    else
      $f \leftarrow (\ln(A)^\alpha)' - (\frac{-R}{Q})'$; `//` M `can be written as` $\ln(A)^\alpha$
  return f;

Algorithm 2: CHECKIOINT(F, S)

Input: $F \in \mathrm{E}_{\int}$ and $S \in \Sigma$
Output: return True or False. See Section 3.3.

begin
  $f \leftarrow$ I2D(F);
  return False if I2D(F) has failed;
  `//` f `is in` E_{∂}
  if *one of the normal forms of* f, f' *and* f'' *w.r.t.* S *is zero* **then return** True;
  return False

3.4 The Integration Model [I2D$_b$]

As described in Section 3.1, the INT algorithm has strong limitations and is difficult to improve. Thus, we trained a deep learning model denoted [I2D$_b$] which is the "inverse" of Algorithm I2D (hence the suffix b for backward). This model is more powerful than INT since it deals with exponentials, logarithms, and integrating factors. It will prove particularly helpful in Section 4.3 to generate various and interesting expressions in $\mathrm{E}_\int$ involving fractions, exponentials and logarithms, which *could not have been easily produced otherwise*. This section details how to train the model [I2D$_b$].

Dataset Generation First, we need pairs (f, F) in $\mathrm{E}_\partial \times \mathrm{E}_\int$ where F involves exponentials, logarithms, and fractions. To generate a pair, we pick a random integral expression F of $\mathrm{E}_\int$ (by generating trees as explained later in Section 4.1), and compute $f = \text{I2D}(F)$. If the algorithm I2D succeeds within 10 seconds and if f contains at least a derivative of $y(t)$, the pair (f, F) is encoded into a pair of lists of tokens (see details in Section 4.1). The pair is kept if the number of tokens of f (resp. F) is smaller than 500 (resp. 200).

Finally, we generate a training (resp. test) dataset containing 1400000 (resp. 20000) unique pairs. Note that the limit of 500 tokens is a compromise between training time and the capacity of the GPUs we use. Moreover, expressions F with around 200 tokens tend to produce expressions $f = \text{I2D}(F)$ with around 500 tokens.

In fact, expressions are obtained by generating two different types of expressions in equal numbers to balance the dataset between expressions with and without exponentials/logarithms. The first type of expressions involves exp and ln and are generated with a number of nodes randomly chosen between 2 and 15. The internal nodes I, the leaves L, and their associated probabilities are defined as follows: $I = [(+, 15/72), (-, 15/72), (\times, 10/72), (\div, 10/72), (\int, 12/72), (\exp, 5/72), (\ln, 5/72)]$, and $L = [(y, 1)]$. The second type of expressions does not involve exp and ln and are generated with a number of nodes randomly chosen between 2 and 15. The internal nodes I, the leaves L, and their associated probabilities are defined as follows: $I = [(+, 15/90), (-, 15/90), (\times, 10/90), (\div, 30/90), (\int 20/90)]$, and $L = [(y, 1)]$.

Note that we do not generate t in leaves, but it can be produced when generated expressions are simplified using SymPy (e.g. $\int x/x$ is simplified into 1 and then $\int 1$ is simplified into t).

Training The training process is the same as in Section 5.2 below.

Evaluation To check if a prediction $\hat{F}$ of an input f is correct, we check that expanding the expression $f - \text{I2D}(\hat{F})$ yields zero. If I2D returns FAIL or does not terminate within 10 seconds, the prediction is considered wrong.

Results Using the test dataset of 20000 pairs, we obtain a success rate of 53,24% (resp. 79,92%) with a beam search of size 1 (resp. size 10) and with the evaluation method from the previous paragraph.

4 Data Encoding and Generation

This section presents how to generate and encode expressions (Section 4.1) and produce a pair (S, F) (Sections 4.2 and 4.3) where $S \in \Sigma$ and $F \in \mathrm{E}_\int$. Those pairs will be used to train deep learning models that go from Σ to $\mathrm{E}_\int$.

4.1 Generate and Encode an Expression

Following [19], we manipulate our expressions as trees where the nodes are operators and the leaves are time-dependent functions ($x(t)$ and $y(t)$), the time t, and rational numbers. To generate an expression, we generate a random tree by picking random nodes (e.g. $+$, $-$, $\times$, $\int$, d/dt, ...) and leaves ($x(t)$, $y(t)$, a_0, t, ...) according to a fixed probability of appearance. Note that we do not generate rationals in leaves, but (small) rationals will be produced during the simplications using SymPy (e.g. $x+x$ is simplified into $2\times x$, x/x is simplified into 1, ...).

Those trees are then translated to lists of tokens using the prefix notation. We denote by `Add`, `Mul`, `Integral`, `Derivative`, `exp`, `log`, `pow` the tokens representing the operators $+$, $\times$, $\int$, d/dt, exp, ln and powers. Moreover, integers are encoded using signed positional notation (e.g. 13 is encoded as `[INT+ 1 3]`, -15 as `[INT- 1 5]`). For instance, encoding $y(t) - \int y(t)^2 + \int(e^{y(t)})$ yields `[Add Add Mul INT- 1 Integral pow y INT+ 2 Integral exp y y]`.

4.2 Generate a Dynamical System

To generate a system in Σ, we simply generate and encode (see Section 4.1) two polynomials expressions $p(x(t), y(t))$ and $q(x(t), y(t))$ with a number of nodes randomly chosen between 1 and 15. The internal nodes I, the leaves L, and their associated probabilities are defined as follows: $I = [(+, 8/26), (-, 8/26), (\times, 10/26)]$ and $L = [(x, 1/2), (y, 1/2)]$.

The expression $\dot{x}(t) = p$ is then encoded by the list starting with `Eq Derivative x` followed by the encoding of p. The expression $\dot{y}(t) = q$ is encoded in the same way. Finally, the system is encoded by concatenating the two equations separated by the token ",". As an example, the system $\{\dot{x}(t) = x(t) + y(t), \dot{y}(t) = x(t)y(t)\}$ is encoded by [`Eq Derivative x Add x y , Eq Derivative y Mul x y`].

4.3 Generate a Pair in $\Sigma \times \mathrm{E}_\int$

Figure 2 illustrates three different approaches to compute an integral I/O equation $F \in \mathrm{E}_\int$ from a dynamical system $S \in \Sigma$. This section details how to generate a pair (S, F) using each approach. All pairs are then encoded with the process described above in Section 4.1.

Using IE Generate a dynamical system $S \in \Sigma$ (Section 4.2). Transform S into integral form $T = \{x(t) = x(0) + \int(p(x(t), y(t)),\ y(t) = y(0) + \int(q(x(t), y(t)))\}$. We choose a specific initial condition by replacing $x(0)$ and $y(0)$ by 1 in T. Then compute IE(T) and pick an integral I/O equation $F \in \mathrm{IE}(T) \cap \mathrm{E}_\int$ if such an F exists. Repeat this process until a pair is found.

Using DE+INT Generate a dynamical system $S \in \Sigma$ (Section 4.2) and compute the differential I/O equation $f \in \mathrm{E}_\partial$ with DE. Using the INT algorithm on f returns an integral I/O equation $F \in \mathrm{E}_\int$ if it succeeds. Replace $y(0)$ by 1 in F. Repeat this process until a pair is found.

Using DE+[I2D$_b$] Follow the same method as in the previous paragraph DE+INT, by replacing INT with the model [I2D$_b$]. The output F of [I2D$_b$] is verified with Algorithm CheckIOInt (Section 3.3) with a timeout value of 10 seconds. Repeat this process until a pair is found.

5 Experiment

Our main experiment aims at comparing the success rate of IE and DE+INT with deep learning models trained to perform the integral elimination. Sections 5.1 and 5.2 present the datasets and the models. Finally, Section 5.3 gives details on the evaluation of our different models and approaches.

5.1 Datasets

This section details the five datasets used for the experiment. Using the three generation processes described in Section 4.3, we can easily build the first three datasets. The fourth one simply is the aggregation (duplicates are filtered) of the three previous datasets. The last one ($\text{Test}_{\text{Unlabeled}}$) is an unlabelled dataset which only contains systems (i.e. without the solution). This dataset is used only to evaluate the models and algorithms on more general data.

Three Datasets Created using Section 4.3 For each generation process given in Section 4.3, we generate 102 000 unique pairs in $\Sigma \times \mathrm{E}_{\int}$. When generating a pair, a timeout of 10 seconds is used since IE and INT can take a lot of time. Moreover, we only keep the pairs (Σ, F) where the computed integral I/O equation F is less than 200 tokens to match the size of the [I2D$_b$] outputs. We also ensure that F contains integrals.

For each of the three datasets, splitting the 102 000 unique pairs gives us the train (100 000 pairs), test (1000 pairs) and valid (1000 pairs) datasets. We denote by Train_{IE}, $\text{Train}_{\text{DE+INT}}$ and $\text{Train}_{\text{DE+[I2D}_b\text{]}}$ (resp. Test_{IE}, $\text{Test}_{\text{DE+INT}}$ and $\text{Test}_{\text{DE+[I2D}_b\text{]}}$) the train (resp. test) datasets obtained using IE, DE+INT and DE+[I2D$_b$].

Aggregate the Three Datasets Using those datasets, we create a new dataset by aggregating (and filtering the duplicates) the 3 previous train, test and valid datasets. We thus obtain the train (298 732 pairs), test (3 000 pairs) and valid (3 000 pairs) datasets. We denote by $\text{Train}_{\text{MIX}}$ (resp. Test_{MIX}) the train (resp. test) dataset.

An Unlabelled Dataset Finally, we create an unlabelled test dataset that contains 10 000 systems in Σ obtained by only generating random systems $S \in \Sigma$ (i.e. without the corresponding integral equations). This dataset (denoted $\text{Test}_{\text{Unlabeled}}$) will only be used to evaluate our different models/algorithms.

5.2 Training

We use a classical Transformer architecture [26] with 8 attention heads, 6 layers and a dimension of 512. The training is done on a A-100 GPU using the Adam optimiser, a learning rate of 10^{-4} and a batch size of 32 pairs. We picked the best models (i.e. with the minimal loss) on the validation datasets over 100 epochs. This typically requires between 10 and 20 hours in total.

Using the 4 training datasets from Section 5.1, we obtained the following models that perform the integral elimination: a) [IE] trained using Train_{IE}, b) [DE+INT] trained using $\text{Train}_{\text{DE+INT}}$, c) [DE+[I2D$_b$]] trained using $\text{Train}_{\text{DE+[I2D}_b\text{]}}$, d) [MIX] trained using $\text{Train}_{\text{MIX}}$.

5.3 Evaluation

This section presents how we evaluate each model/algorithm. All pairs are checked with a timeout of 10 seconds. A pair is considered incorrect if the timeout is reached.

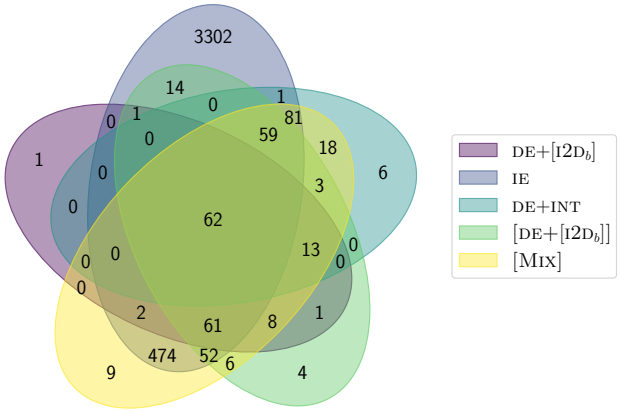

Figure 3: Venn diagram of the treated examples by DE+[I2D$_b$] (pointing west), IE (pointing north), DE+INT (pointing east), [DE+[I2D$_b$]] (pointing south-east) and [MIX] (pointing south-west) on the Test$_{\text{Unlabeled}}$ dataset.

Evaluation of the Deep Learning Models [IE], [DE+INT], [DE+[I2D$_b$]] and [MIX] A straightforward evaluation (denoted `Syntax`) consists in comparing syntactically the predicted list of tokens to the expected list of tokens. Obviously, this evaluation cannot be applied to the unlabelled dataset (Test$_{\text{Unlabeled}}$). The `Syntax` evaluation is very strict and will likely reject mathematically correct answers. To overcome this problem, we use the CHECKIOINT verification described in Section 3.3. This verification is used to evaluate the outputs of a beam search of size 1 and 10 (`Beam1` and `Beam10`).

Evaluation of the Hybrid Method DE+[I2D$_b$] We use the CHECKIOINT verification on the output given by the method DE+[I2D$_b$]. Because this method is hybrid, we only used a beam search of size 1 for [I2D$_b$] for simplicity reasons.

Evaluation of IE and DE+INT To evaluate IE and DE+INT on a dataset, we simply compute the percentage of systems successfully treated. Indeed, those algorithms always return valid outputs.

6 Results

From a system $S \in \Sigma$, we have 7 methods to compute an integral I/O equation $F \in \mathrm{E}_{\int}$. This section presents a comparison of those 7 methods using the 5 datasets described in Section 5.1. The models are [DE+INT], [DE+[I2D$_b$]], [IE] and [MIX] (see Section 5.2). The three others methods are IE, DE+INT and DE+[I2D$_b$] (see Figure 2). Results are discussed in Section 7.

Summary of the Success Rates Table 1 presents the success rate obtained by the 7 considered models/algorithms on the 5 datasets. In this experiment, timeouts account for 16.29% of the errors of the deep learning based approaches.

	[DE+INT]			[DE+[I2D$_b$]]			[IE]			[MIX]			DE+INT	DE+[I2D$_b$]	IE
	Syntax	Beam1	Beam10	Syntax	Beam1	Beam10	Syntax	Beam1	Beam10	Syntax	Beam1	Beam10			
Test$_{\text{DE+INT}}$	89,60	90,30	90,30	2,40	30,7	37,20	0,00	61,00	54,60	73,30	88,00	91,40	100,00	23,90	76,40
Test$_{\text{DE+[I2D}_b\text{]}}$	1,80	43,10	42,20	83,40	88,20	87,40	0,00	41,40	43,10	66,00	88,00	95,7	46,60	100,00	83,10
Test$_{\text{IE}}$	0,00	17,20	17,40	0,00	13,20	19,30	73,70	76,70	53,00	49,10	70,90	76,10	17,80	8,70	100,00
Test$_{\text{MIX}}$	30,46	50,33	50,90	28,60	43,80	48,00	24,56	60,30	50,93	62,79	82,50	87,70	54,80	44,20	86,50
Test$_{\text{Unlabeled}}$	N/A	2,32	2,34	N/A	2,07	2,55	N/A	6,27	5,50	N/A	6,77	8,04	2,43	1,49	41,09

Table 1: Comparison of the success rate of the 7 approaches to compute an integral I/O equation from a dynamical system on 5 different datasets. The `Syntax` evaluation cannot be computed for the Test$_{\text{Unlabeled}}$ dataset since outputs are unknown (those cases are marked with N/A).

Venn Diagram of Test$_{\text{Unlabeled}}$ Dataset Figure 3 presents a Venn diagram obtained on the Test$_{\text{Unlabeled}}$ dataset in the following manner. For each of the five models/algorithms from the Figure 3 legend, we build the set of elements of Test$_{\text{Unlabeled}}$ which are successfully treated. This produces five sets with many overlaps. As usual, the numbers indicate the cardinality of all possible intersections of the five sets. Note that we limited ourselves to five methods/algorithms to obtain a readable diagram.

7 Discussion

7.1 Comparison of the Success Rates

We discuss some results given by Table 1 that we found the most interesting and meaningful.

Performance using `Beam1` and `Beam10` Evaluations The `Beam10` evaluation we implemented does not necessarily contain the `Beam1` solution (which is greedy). This explains why some `Beam10` success rates are sometimes lower than the `Beam1` ones.

Performance of IE Compared to DE+INT Algorithm IE is always better than DE+INT except on Test$_{\text{DE+INT}}$ where IE has a success rate of 76,4% versus 100%. This was unexpected since IE involves more complicated machinery and produces more general expressions than INT.

Example 2 shows an example handled by DE+INT but not by IE. This example might be useful for improving IE.

Example 2. *The expression f is the differential I/O equation in* E_∂ *computed by* DE*, and F is the integral I/O equation in* $\mathrm{E}_\int$*.*

$$S\colon \begin{cases} \dot{x} = x^3 - xy \\ \dot{y} = -x^2 y \end{cases} \qquad \begin{aligned} f &= 2y^2\dot{y} + y\ddot{y} + \dot{y}^2 \\ F &= 6y_0^2x_0^2t + 3(y^2 - y_0^2) + 4\textstyle\int(y^3 - y_0^3) \end{aligned}$$

Performance of [DE+[I2D$_b$]], DE+[I2D$_b$], [DE+INT] and DE+INT Those are the four approaches based on DE. The model [DE+INT] performs almost as well as DE+INT. Algorithm DE+INT performs better than DE+[I2D$_b$] and [DE+[I2D$_b$]] except on Test$_{\text{DE+[I2D}_b\text{]}}$ which is expected since Test$_{\text{DE+[I2D}_b\text{]}}$ contains fractions, exponential and logarithms (see Section 3.4).

Finally, [DE+[I2D$_b$]] tends to be slightly better than DE+[I2D$_b$]. A possible interpretation is that [I2D$_b$] has not been specifically trained on equations produced by IE, unlike the model [DE+[I2D$_b$]].

	[DE+INT]			[DE+[I2D$_b$]]			[IE]			[MIX]			DE+INT	DE+[I2D$_b$]	IE
	Syntax	Beam1	Beam10	Syntax	Beam1	Beam10	Syntax	Beam1	Beam10	Syntax	Beam1	Beam10			
Test$_{\text{UnlabeledFiltered}}$	N/A	3,20	3,22	N/A	2,63	3,24	N/A	8,13	7,12	N/A	8,73	10,12	3,35	1,92	18,12

Table 2: Comparison of the success rate of the 7 approaches to compute an integral I/O equation from a dynamical system on the Test$_{\text{UnlabeledFiltered}}$ obtained after removing from Test$_{\text{Unlabeled}}$ all the pairs (S, F) in $\Sigma \times \mathrm{E}_{\int}$ where F contains more than 200 tokens. The `Syntax` evaluation cannot be computed for the Test$_{\text{Unlabeled}}$ dataset since outputs are unknown (those cases are marked with N/A).

Performance of IE Compared to Deep Learning Approaches On the Test$_{\text{DE+INT}}$, Test$_{\text{DE+[I2D}_b\text{]}}$ and Test$_{\text{IE}}$ datasets in Table 1, [MIX] is the best model, and thus seems to be a good synthesis of [DE+INT], [DE+[I2D$_b$]] and [IE]. It even outperforms our Algorithm IE on the Test$_{\text{DE+INT}}$ and Test$_{\text{DE+[I2D}_b\text{]}}$ datasets.

Nevertheless, on the Test$_{\text{Unlabeled}}$ dataset, [MIX] has a success rate of only 8,04% while IE has 41,09% which means that IE is better on random systems. This result is in fact biased since all the deep learning models are trained to output an equation of size less than 200 tokens but IE does not have such limitations. When evaluating IE on Test$_{\text{Unlabeled}}$, we did not restrict the length of IE outputs. Among the 41,09% of systems treated by IE, the mean size of the output was 3063 tokens. To make a fair comparison, we have filtered Test$_{\text{Unlabeled}}$ by excluding the 2805 pairs out of 10000 for which IE computes an output with size more than 200 tokens. This filtered set is denoted Test$_{\text{UnlabeledFiltered}}$. Table 2 presents the success rate of each approach for this filtered set. Table 2 shows that the lead of IE is clearly reduced.

7.2 Examples Treated by Deep Learning Models Only

This section presents some examples (from Figure 3) verified by hand where deep learning models succeed whereas both DE+INT and IE failed. Those examples are by themselves contributions. Indeed, they are very difficult to obtain by hand, and are a source of inspiration for enhancing our algorithm IE. Moreover, it is quite surprising that those examples were found by models trained on data produced a partial algorithm (I2D). Those examples clearly show the interest of deep learning models. We present a selection of examples of various styles treated by the model [DE+[I2D$_b$]].

In all examples, f is the differential I/O equation in E_{∂} computed by DE, and F is the integral I/O equation in $\mathrm{E}_{\int}$ obtained by [DE+[I2D$_b$]]. Note that the equations F do not contain the initial conditions. Additional (hand) computations are needed to retrieve the general integral I/O equations F_{ics}. Automating this process will be addressed in future work.

Example 3 (A polynomial example)**.**

$$S\colon \begin{cases} \dot{x} = x^2 - y^2 - x + 2y \\ \dot{y} = xy \end{cases} \qquad \begin{aligned} f &= y^4 - 2y^3 + y\dot{y} + y\ddot{y} - 2\dot{y}^2 \\ F &= -y + 2\int ty^2 - \int(y^2 \int y) + \int y \\ F_{ics} &= -yy_0^2 + y_0^3 + 2y_0^2 \int ty^2 + y_0^2 \int(y - y^2 \int y) + (x_0y_0 - y_0) \int y^2 \end{aligned}$$

Example 4 (Example involving a fraction)**.**

$$S\colon \begin{cases} \dot{x} = x^3 \\ \dot{y} = 2x - y \end{cases} \qquad \begin{aligned} f &= 4\ddot{y} - \dot{y}^3 - 3y\dot{y}^2 - 3\dot{y}y^2 + 4\dot{y} - y^3 \\ F &= y + \int y + 4\int \tfrac{1}{y + \int y} \\ F_{ics} &= y - y_0 + \int y + 4x_0 \int \tfrac{1}{x_0 \int y + x_0 y - x_0 y_0 - 2} \end{aligned}$$

Example 5 (Example involving logarithms)**.**

$$S\colon \begin{cases} \dot{x} = y^2 - x^2 - x - xy \\ \dot{y} = -xy \end{cases} \qquad \begin{aligned} f &= y^4 + y^2\dot{y} + y\dot{y} + y\ddot{y} - 2\dot{y}^2 \\ F &= y + \int y^2 \ln(y) + \int y^2 \int y - \int y \\ F_{ics} &= y - y_0 + (\tfrac{x_0}{y_0} + \tfrac{1}{y_0}) \int y^2 + \int y^2 \ln(\tfrac{y}{y_0}) + \int y^2 \int y - \int y \end{aligned}$$

Example 6 (Example involving both logarithms and fractions)**.** *This example seems very difficult since it involves a logarithm stuck inside a fraction.*

$$S\colon \begin{cases} \dot{x} = 4x^2y - 2x^3 \\ \dot{y} = y^2 - xy \end{cases} \qquad \begin{aligned} f &= 2y^6 - 2y^4\dot{y} - y^3\dot{y} - 2y^2\dot{y}^2 + y^2\ddot{y} - y\dot{y}^2 + 2\dot{y}^3 \\ F &= -2y + \int \tfrac{y}{\ln(y) + \int y} + 2\int y^2 \\ F_{ics} &= 2y_0 - 2y - 2x_0y_0 \int \tfrac{y}{y_0 - 2x_0y_0(\ln(\frac{y}{y_0}) + \int y)} + 2\int y^2 \end{aligned}$$

7.3 Limitations

The models considered in this experiment only involve two variables x and y. They do not include any parameters, nor generic initial conditions for x and y. Moreover, the outputs of the models are limited to 200 tokens, which is rather small for representing equations. In practice, interesting models involve at least 3 or 4 variables and at least 3 or 4 parameters. We do not know yet how many resources should be necessary to handle such models.

The $\text{Test}_{\text{Unlabeled}}$ (see Table 1) shows that our models struggle with random inputs. This means that our models do not generalise well. It might be due to the limit of 200 tokens for outputs, too few data and insufficiently varied data.

Algorithm I2D performs well during the validation (which is not too surprising) but fails quite a lot during the generation of the data. Most of the time for generating all the data was spent in generating data using Algorithm I2D. It took around 50 modern computers (with a Intel Xeon Gold 5220 / 18 cores) 5 days each to generate all our datasets.

8 Conclusion

We have discussed the interest of deep learning techniques to perform integral elimination. The outcome is very promising, especially by exhibiting new interesting examples unhandled by our first version [20] of the INTEGRAL_ELIMINATION algorithm. Those examples have already been used to improve INTEGRAL_ELIMINATION by introducing logarithms (see [22]). In Table 1, we obtained (using IE [20]) an integral I/O equation for 41,09% systems of the $\text{Test}_{\text{Unlabeled}}$ dataset. Using our new version [22] (with logarithms), we are now able to treat 45,06% of those systems (including Example 5). Examples involving fractions still need to be investigated.

Acknowledgments

This work has been partially supported by the THIA ANR Program “AI_PhD@Lille”, from the French National Research Agency, grant ANR-20-THIA-0014.

Experiments presented in this paper were carried out using the Grid’5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations (see https://www.grid5000.fr).

References

[1] Rashid Barket, Matthew England, and Jürgen Gerhard. *Tree-Based Deep Learning for Ranking Symbolic Integration Algorithms*. 2025. arXiv: 2508.06383 [cs.SC].

[2] Rashid Barket et al. "Transformers to Predict the Applicability of Symbolic Integration Routines". In: *The 4th Workshop on Mathematical Reasoning and AI at NeurIPS'24*. 2024. URL: https://openreview.net/forum?id=b2Ni828As7.

[3] François Boulier. *The DifferentialAlgebra project*. 2023. URL: https://codeberg.org/francois.boulier/DifferentialAlgebra.

[4] François Boulier and François Lemaire. "A Normal Form Algorithm for Regular Differential Chains". In: *Mathematics in Computer Science* 4.2 (2010), pp. 185–201. DOI: 10.1007/s11786-010-0060-3.

[5] François Boulier et al. "Additive Normal Forms and Integration of Differential Fractions". In: *Journal of Symbolic Computation* 77 (2016), pp. 16–38. DOI: 10.1016/j.jsc.2016.01.002.

[6] François Boulier et al. "An Algorithm for Converting Nonlinear Differential Equations to Integral Equations with an Application to Parameter Estimation from Noisy Data". In: *Computer Algebra in Scientific Computing*. Ed. by Vladimir P. Gerdt et al. Cham: Springer International Publishing, 2014, pp. 28–43. ISBN: 978-3-319-10515-4. DOI: 10.1007/978-3-319-10515-4_3.

[7] François Boulier et al. "An Equivalence Theorem For Regular Differential Chains". In: *Journal of Symbolic Computation* 93 (July 2019), pp. 34–55. DOI: 10.1016/j.jsc.2018.04.011.

[8] François Boulier et al. "Computing representations for radicals of finitely generated differential ideals". In: *Applicable Algebra in Engineering, Communication and Computing* 20.1 (Apr. 2009), pp. 73–121. DOI: 10.1007/s00200-009-0091-7.

[9] Beatriz R. Brum et al. *Discovering Equations From Data: Symbolic Regression in Dynamical Systems*. 2025. DOI: 10.48550/arXiv.2508.20257.

[10] Tom Coates, Alexander Kasprzyk, and Sara Veneziale. "Machine learning detects terminal singularities". In: *Advances in Neural Information Processing Systems*. Ed. by A. Oh et al. Vol. 36. Curran Associates, Inc., 2023, pp. 67183–67194. URL: https://proceedings.neurips.cc/paper_files/paper/2023/file/d453490ada2b1991852f053fbd213a6a-Paper-Conference.pdf.

[11] Stéphane d'Ascoli et al. "ODEFormer: Symbolic Regression of Dynamical Systems with Transformers". In: *The Twelfth International Conference on Learning Representations*. 2024. URL: https://openreview.net/forum?id=TzoHLiGVMo.

[12] Alex Davies et al. "Advancing mathematics by guiding human intuition with AI". In: *Nature* 600.7887 (Dec. 2021), pp. 70–74. ISSN: 1476-4687. DOI: 10.1038/s41586-021-04086-x.

[13] Ernest Davis. *The Use of Deep Learning for Symbolic Integration: A Review of (Lample and Charton, 2019)*. 2019. DOI: 10.48550/arXiv.1912.05752.

[14] Michel Fliess. "Automatique et corps différentiels". In: *Forum Mathematicum - FORUM MATH* 1 (Jan. 1989), pp. 227–238. DOI: 10.1515/form.1989.1.227.

[15] Louis Goupil et al. "Tree Based Diagnosis Enhanced with Meta Knowledge". In: *34th International Workshop on Principles of Diagnosis (DX'23)*. Loma Mar, United States, Sept. 2023. URL: https://hal.science/hal-04186400.

[16] Louis Goupil et al. "Tree based Diagnosis Enhanced with Meta Knowledge Applied to Dynamic Systems". In: *IFAC-PapersOnLine* 58.4 (2024). 12th IFAC Symposium on Fault Detection, Supervision and Safety for Technical Processes SAFEPROCESS 2024, pp. 1–6. ISSN: 2405-8963. DOI: 10.1016/j.ifacol.2024.07.184.

[17] Pierre-Alexandre Kamienny et al. "End-to-end Symbolic Regression with Transformers". In: *Advances in Neural Information Processing Systems*. Ed. by Alice H. Oh et al. 2022. URL: https://openreview.net/forum?id=GoOuIrDHG_Y.

[18] Hiroshi Kera et al. *Learning to Compute Gröbner Bases*. 2024. DOI: 10.48550/arXiv.2311.12904.

[19] Guillaume Lample and François Charton. "Deep Learning for Symbolic Mathematics". In: *The Ninth International Conference on Learning Representations (ICLR)*. Virtual only, June 2019, p. 24. DOI: 10.48550/arXiv.1912.01412.

[20] François Lemaire and Louis Roussel. "Contribution to Integral Elimination". In: *Computer Algebra in Scientific Computing*. Ed. by François Boulier et al. Vol. 14938. Rennes, France: Springer Nature Switzerland, Sept. 2024, pp. 215–235. DOI: 10.1007/978-3-031-69070-9_13.

[21] François Lemaire and Louis Roussel. "Parameter Estimation using Integral Equations". In: *Maple Transactions*. Proceedings of the Maple Conference 2023 4.1 (June 2024). DOI: 10.5206/mt.v4i1.17126.

[22] François Lemaire and Louis Roussel. "Recent Advances on Integral Elimination". In: *Proceedings of the 2025 International Symposium on Symbolic and Algebraic Computation*. ISSAC '25. Association for Computing Machinery, 2025, pp. 249–257. ISBN: 9798400720758. DOI: 10.1145/3747199.3747568.

[23] Daniel Richardson. "Some Undecidable Problems Involving Elementary Functions of a Real Variable". In: *The Journal of Symbolic Logic* 33.4 (1969), pp. 514–520. DOI: 10.2307/2271358.

[24] Tereso del Río and Matthew England. "Lessons on Datasets and Paradigms in Machine Learning for Symbolic Computation: A Case Study on CAD". In: *Mathematics in Computer Science* 18.3 (Sept. 2024), p. 17. DOI: 10.1007/s11786-024-00591-0.

[25] Vighnesh Shiv and Chris Quirk. "Novel Positional Encodings To Enable Tree-Based Transformers". In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach et al. Vol. 32. Curran Associates, Inc., 2019. URL: https://proceedings.neurips.cc/paper_files/paper/2019/file/6e0917469214d8fbd8c517dcdc6b8dcf-Paper.pdf.

[26] Ashish Vaswani et al. "Attention is All you Need". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[27] Eric Walter and Luc Pronzato. *Identification of parametric models*. en. 1997th ed. Communications and Control Engineering. Berlin, Germany: Springer, Jan. 1997. ISBN: 3-540-76119-5.