Learning to Compute Polynomial Products with Transformers

Yuxuan Song, Bin Wang, Changbo Chen

November 16, 2026



https://scml.risc.jku.at

RISC Proceedings on
Symbolic Computation and Machine Learning
Publication Number: 4

https://doi.org/10.35011/risc-proceedings-scml.4



This work is published under the
Creative Commons Attribution 4.0 International License.



This work has
open data inside.

Cite as: Yuxuan Song, Bin Wang, Changbo Chen: Learning to Compute Polynomial Products with Transformers. *RISC Proceedings on Symbolic Computation and Machine Learning*, 4, November 16, 2026. SCML, https://scml.risc.jku.at, doi.org/10.35011/risc-proceedings-scml.4.

The SCML publication forum is dedicated to all research that strives to combine "Symbolic Computation" (SC) and "Machine Learning" (ML) as two major approaches to "Artificial Intelligence", in particular to the application of ML to SC, the application of SC to ML, and the hybrid combination of SC and ML to solving problems. More information about SCML is available at https://scml.risc.jku.at.

Steering Commitee and Editorial Board

Bruno Buchberger RISC Institute, Johannes Kepler Unversity Linz, Austria.
François Charton Meta AI and CERMICS Center, Ecole des Ponts, France.
Matthew England CSMM Center, Coventry University, UK.
Cezary Kaliszyk Computer Science Research Group, University of Melbourne, Australia.
Manuel Kauers Institute for Algebra, Johannes Kepler University Linz, Austria.
Hiroshi Kera Institute for Advanced Academic Research (IAAR), Chiba University, Japan.
Temur Kutsia RISC Institute, Johannes Kepler Unversity Linz, Austria.
Bernhard Moser Software Competence Center Hagenberg (SCCH), Austria.
Markus Schedl Institute for Computational Perception, Johannes Kepler University Linz, Austria.
Wolfgang Schreiner RISC Institute, Johannes Kepler Unversity Linz, Austria.
Martina Seidl Institute for Symbolic Artificial Intelligence, Johannes Kepler University Linz, Austria.
Wolfgang Windsteiger RISC Institute, Johannes Kepler Unversity Linz, Austria.

Scientific Commitee

Jasmin Blanchette Ludwig-Maximilians-Universität München, Germany.
Curtis Bright University of Windsor, Canada.
Swarat Chaudhuri University of Texas, USA.
David Cerna Czech Academy of Sciences, Czech Republic.
Changbo Chen University of Chinese Academy of Sciences, China.
Sebastijan Dumancic Delft University of Technology, The Netherlands.
Johannes Fürnkranz Johannes Kepler University Linz, Austria.
Vijay Ganesh Georgia Institute of Technology, USA.
Amir Hashemi Isfahan University of Technology, Iran.
Sean Holden University of Cambridge, UK.
Yuki Ishihara Nihon University, Japan.
Dietmar Jannach University of Klagenfurt, Austria.
Yuta Kambe Mitsubishi Electric Corporation, Japan.
Ido Kaminer Technion — Israel Institute of Technology, Israel.
Ekaterina Komendantskaya University of Southampton, UK.
Konstantin Korovin University of Manchester, UK.
Gabriel Kronberger University of Applied Sciences Upper Austria, Austria.
Tom Oliver University of Westminster, UK.
Fabricio Olivetti de França Universidade Federal do ABC, Brazil.
Jelle Piepenbrock Eindhoven University of Technology, The Netherlands.
Sebastian Pokutta Technische Universität Berlin, Germany.
Werner M. Seiler University of Kassel, Germany.
Zsolt Zombori HUN-REN Alfréd Rényi Institute of Mathematics, Hungary.

Learning to Compute Products of Polynomials with Transformers *

Yuxuan Song[1,2], Bin Wang[1,2], and Changbo Chen[1,2]

[1] Chongqing Institute of Green and Intelligent Technology, Chinese Academy of Sciences
Chongqing, China
[2] Chongqing School, University of Chinese Academy of Sciences
Chongqing, China

Abstract

Polynomial multiplication is one of the most basic arithmetic operations in symbolic computation. Many advanced computations, such as polynomial pseudo division and resultant calculation, boil down to it, which further establishes the foundation for building sophisticated symbolic solvers. In this work, we present a transformer-based approach to compute product of two polynomials. The goal here is not to accelerate polynomial multiplication. Rather, we demonstrate that learning polynomial multiplication helps on learning pseudo division, which further improves learning of characteristic sets via a pre-training and fine-tuning paradigm.

One key insight in this work is to firstly learn an intermediate collected product of polynomials. We then extend an existing scratchpad with bi-level positional coupling technique for learning the addition of multiple unsigned integers to the case of signed integers, paving the way to turn the collected product into the final product. Experiments show that this multi-stage method can predict the result of polynomial multiplication in high accuracy and significantly outperforms the direct one-step learning approach.

1 Introduction

Deep learning and emerging large language models (LLMs) have greatly advanced artificial intelligence (AI), aiming to provide scalable and universal approaches for solving problems within or beyond human capability, where a grand challenge and cutting-edge direction of AI lies in development of its mathematical reasoning abilities [1]. Automatic mathematical reasoning, however, is not a new topic; numerous logic- and computation-oriented methods have been explored [14]. In particular, symbolic computation has remained one of the principal methods for automatic theorem proving [34, 37]. Addressing this reasoning challenge calls for techniques that integrate symbolic computation with machine learning, thereby combining their strengths.

Symbolic computation [11, 31] is featured with high-level algebraic algorithms grounded in rigorous theory and efficient implementations of exact arithmetic operations, such as polynomial multiplication, within computer algebra systems. However, pursuing exact solutions often entails high computational complexity. For example, both Gröbner basis computation for deciding ideal membership [25] and cylindrical algebraic decomposition for quantifier elimination [2] can be doubly exponential in the number of variables. As a result, unrolling these algorithms

*This work is supported by the National Key Research Program of China under Grant No. 2023YFA1009402 and Chongqing Talents Plan Youth Top-Notch Project (2021000263). Corresponding author: Changbo Chen (chenchangbo@cigit.ac.cn).

into basic operations produces "deep" computation graphs, which are often far deeper than typical neural networks, posing unique challenges for integrating machine learning with symbolic methods.

Despite of the challenge, some progress have been made on embracing machine learning for symbolic computation from different fronts. The work [23] demonstrates the power of transformers on symbolic integration. Later, transformers are applied on symbolic regression and computing global Lyapunov functions [12]. The work [22] and [24] explore learning to compute Gröbner bases via transformers. These works focus on directly learning a map from the input to the output of a high-level algorithm, without worrying the "deep computing" nature of existing algorithms, and have achieved certain success on small input and output size problems. One big challenge to train these models is to generate a labeled dataset of moderate size by forward computation [6], as this process involves calling solvers with complexity growing dramatically with the problem size. For this reason, these works heavily rely on backward data synthesis.

Although heuristic strategies are widely used in symbolic algebraic algorithms, they are usually not the first class citizens when designing the algorithms. As a result, a large room is left for designing better heuristics. One successful example is the introducing of satisfiability modulo theory (SMT) for determining wether polynomial systems have real solutions [19]. Supervised machine learning and reinforcement learning are also natural choices for designing better heuristics. Some progress have been made on adopting these learning techniques, including suggesting variable ordering for cylindrical algebraic decomposition via supervised machine learning [8, 13] and reinforcement learning [16, 18], selecting S-pairs for computing Gröbner basis via reinforcement learning [28], selecting polynomial constraints in CAD construction via supervised learning [3], etc.

On the other hand, with the main motivation to understand limitation and learning ability of transformer models, especially large language models, there has been a series of work focusing on learning more basic integer arithmetic. The work [5] demonstrates that training distributions impact learning of the greatest common divisor of integers. The work [4] introduced GPT-3 and evaluated its performance on integer arithmetic, identifying a bottleneck in length generalization, the ability to generalize to numbers with more digits than those encountered during training. Subsequently, the work [27] systematically investigated the impact of tokenization and positional embeddings on length generalization, employing a transformer model trained from scratch to analyze these effects. Since then, a variety of more sophisticated number representation and positional embedding mechanisms have been proposed [20, 15, 38, 26, 9]. In particular, the works [26, 9] introduced abacus embeddings (position coupling), assigning digits in the same place across different numbers identical embeddings, which significantly improved length generalization in both addition and multiplication. More recently, the work [10] extended length generalization to accumulation and more complex multi-digit multiplication by introducing a scratchpad to store intermediate computational results and employing multi-level positional embeddings. In this scheme, not only do digits in the same place share embeddings, but the digits of each input operand and corresponding intermediate results in the scratchpad are also assigned the same embeddings. Experimental results demonstrate that this approach currently outperforms other settings.

A recent work [17] involving the third author demonstrates that pretraining on shallow basic tasks, such as learning exponent vectors of polynomial products and key feature vectors of resultants, can help learning a deep complex task, that is selecting the optimal variable ordering for cylindrical algebraic decomposition, in the way of fine-tuning pretrained models. This work inspires us to investigate whether products of polynomials can be exactly learned

in high accuracy and whether polynomial multiplication can help on learning more advanced operations and other complex symbolic computation tasks.

Learning products of polynomials via transformers has been included as an example to illustrate the CALT library [21] for learning symbolic computations. In particular, the authors show that predicting the intermediate product of fewer polynomials helps on improving the prediction of final product on simple bi-variate polynomials. In this paper, we present, to our knowledge, the first dedicated method for learning products of two polynomials over the ring of integers with small transformer models. We emphasize that the goal here is not to accelerate polynomial multiplication with neural models. Rather, we would like to demonstrate that learning such a basic operation well may serve as a cornerstone toward handling more complex tasks, such as pseudo division, and computation of characteristic sets [29, 34]. The main contributions of this work are as follows:

(*i*) We extend the scratchpad with bi-level position coupling introduced [10] for learning the addition of multiple unsigned integers to the case of signed integers.

(*ii*) We propose an effective transformer-based multi-stage learning method for polynomial multiplication. The core idea is to firstly learn an intermediate collected product and then incorporating (*i*) to calculate the cumulative sum of integer coefficients. We demonstrate how to handle long polynomials multiplication by further introducing models to learn the addition of polynomials in collected form.

(*iii*) We propose a bi-level pretraining & fine-tuning scheme to learn computation of characteristic sets of polynomial equations based on the models learning the collected products of polynomials.

Experiments on a variety of datasets show that the multi-stage small transformer-based method can effectively learn both univariate and multivariate integer polynomial multiplication and the pretraining & fine-tuning method based on multiplication models substantially improves the learning of pseudo division and computation of characteristic sets. The code and more experimental results are available at `https://github.com/roicar/PMT`. The datasets supporting this work are available at `https://www.scidb.cn/en/s/FJnYFj`.

2 Background

In this section, we briefly recall the operations/algorithms to learn. Throughout this paper, let $\mathbb{A}$ be a commutative ring with unity, $\mathbb{Z}$ be the ring of integers, $\mathbb{Z}_p$ be the finite field with p prime, $\mathbb{Q}$ be the field of rational numbers, $\mathbb{A}^* = \mathbb{A} \setminus \{0\}$, $\mathbf{x} = x_1, \ldots, x_n$ be a sequence of n variables. In this work, the ring $\mathbb{A}$ concerned is $\mathbb{Z}$, $\mathbb{Z}_p$, $\mathbb{Q}$, the polynomial ring $\mathbb{Z}[x_{i_1}, \ldots, x_{i_r}]$, $\mathbb{Z}_p[x_{i_1}, \ldots, x_{i_r}]$, or the polynomial ring $\mathbb{Q}[x_{i_1}, \ldots, x_{i_r}]$, where $1 \leq i_j \leq n$ and $j = 1, \ldots, r$.

Let $f = \sum_{i=1}^{s} c_i \prod_{j=1}^{n} x_j^{e_{ij}}$ be a polynomial in $\mathbb{Q}[\mathbf{x}]$. The i-th term ($1 \leq i \leq s$) of polynomial f consists of the coefficient c_i and the monomial $\prod_{j=1}^{n} x_j^{e_{ij}}$. Two terms endowed with identical monomials are referred to as *like terms*. A polynomial in $\mathbb{Q}[\mathbf{x}]$ can be written in short as $\sum_{\alpha \in A} a_\alpha \mathbf{x}^\alpha$, where A is a finite set of n-tuples of non-negative integers, $a_\alpha \in \mathbb{Q}$, and $\mathbf{x}^\alpha = x_1^{\alpha_1} x_2^{\alpha_2} \cdots x_n^{\alpha_n}$. Given two polynomials $f = \sum_{\alpha \in A} a_\alpha \mathbf{x}^\alpha$ and $g = \sum_{\beta \in B} b_\beta \mathbf{x}^\beta$, the multiplication of two polynomials f and g is defined as $fg = \sum_\gamma (\sum_{\gamma = \alpha + \beta} a_\alpha b_\beta) \mathbf{x}^\gamma$.

Given a variable ordering $x_1 < \cdots < x_n$ and a polynomial $f \notin \mathbb{Q}$, let $x = x_i$ be the largest variable appearing in f, which is called the *main variable* of f. Let $\mathbb{A} = \mathbb{Q}[x_1, \ldots, x_{i-1}]$. Then f can also be regarded as a univariate polynomial in $\mathbb{A}[x]$. The leading coefficient, degree and

leading monomial of f are denoted respectively as $\mathrm{lc}(f)$, $\deg(f)$, and $\mathrm{lm}(f)$, which are also referred to as the initial, the main degree, and the rank of f respectively, denoted by $\mathrm{init}(f)$, $\mathrm{mdeg}(f)$ and $\mathrm{rank}(f)$. If $f \in \mathbb{Q}$, we can extend the above notions by treating f as a univariate polynomial in $\mathbb{A}[x_1]$.

Let $f \in \mathbb{A}[x]$ and $g \in \mathbb{A}^*[x]$. If there exist $m \in \mathbb{A}$ and $q, r \in \mathbb{A}[x]$ such that $mf = qg + r$ and $\deg(r) < \deg(g)$, then we call m the multiplier, q the pseudo quotient, and r the pseudo remainder. Since the polynomial ring $\mathbb{A}$ concerned in this paper is an integral domain, if we choose $m = \mathrm{lc}(g)^{\max(\deg(f)-\deg(g)+1,0)}$, then r is uniquely defined and we write $r = \mathrm{prem}(f, g)$.

Given a sequence of ordered variables $\mathbf{x} = x_1 < \cdots < x_n$ and two polynomials $f, g \in \mathbb{Q}[\mathbf{x}]$, where $g \notin \mathbb{Q}$, we say that f *is reduced w.r.t.* g if $\deg(f, \mathrm{mvar}(g)) < \mathrm{mdeg}(g)$. Given $G \subset \mathbb{Q}[\mathbf{x}] \backslash \mathbb{Q}$, f *is reduced w.r.t.* G if f is reduced w.r.t. every $g \in G$. Let $T = \{t_1, \ldots, t_s\} \subset \mathbb{Q}[\mathbf{x}] \backslash \mathbb{Q}$ be a finite non-empty set of polynomials with distinct main variables such that $\mathrm{mvar}(t_1) < \cdots < \mathrm{mvar}(t_s)$. We call T a *triangular set*. We say T *is reduced* if $T = \emptyset$, or $\{t_1, \ldots, t_{s-1}\}$ is reduced and t_s is reduced w.r.t. $\{t_1, \ldots, t_{s-1}\}$.

Definition 1. *Given $C \subset \mathbb{Q}[\mathbf{x}]$, C is called an ascending chain if one of the following holds:*

- *$C = \{c\}$, where $c \in \mathbb{Q}^*$, called a* contradiction chain*;*
- *C is a reduced triangular set of $\mathbb{Q}[\mathbf{x}]$.*

Let $T = \{t_1, \ldots, t_s\}$ be a contradiction chain or a triangular set of $\mathbb{Q}[\mathbf{x}]$. If $T = \emptyset$, define $\mathrm{prem}(f, T) = f$. Otherwise, define recursively $\mathrm{prem}(f, T) := \mathrm{prem}(\mathrm{prem}(f, t_s), \{t_1, \ldots, t_{s-1}\})$. Let $F \subset \mathbb{Q}[\mathbf{x}]$, define $\mathrm{prem}(F, T) = \{\mathrm{prem}(f, T) \mid f \in F\}$. Note that if T is a contradiction chain, then $\mathrm{prem}(F, T) = \{0\}$ always holds.

Definition 2. *Given a non-empty set of polynomials $F \subset \mathbb{Q}^*[\mathbf{x}]$, an ascending chain C is a* characteristic set *of F if $C \subset \langle F \rangle$ and $\mathrm{prem}(F, C) = \{0\}$.*

Here, the notion of characteristic set is defined in the sense of Wu [34]. Characteristic set was initially introduced by Ritt for studying differential algebra [29]. In Ritt's definition, one requires that $\langle F \rangle$ is a prime ideal and $\mathrm{prem}(\langle F \rangle, C) = \{0\}$. Wu's method gets rid of this strong assumption and has found wide applications in elementary geometry theorem proving and other areas [35]. The Ritt-Wu characteristic set method provides an effective way to decompose the zero set of a system of polynomial equations into a union of zeros of triangular sets, a process known as triangular decomposition [32, 7]. We remark that in Wu's original algorithm, the main basic operation is polynomial pseudo division. Note that Definition 2 provides also an effective way to check if a given set of polynomials is a characteristic set.

Example 1. *Let $F = \{f_1, f_2, f_3\} = \{x_3x_2 + x_2, x_3^3 - x_1, x_3 + 2x_2\}$. One can check by Definition 2 that $C = \{x_1^4 + 3x_1^3 + 3x_1^2 + x_1, -x_2x_1 - x_2, -x_3x_1 - x_3\}$ is a characteristic set of F w.r.t. the variable ordering $x_1 < x_2 < x_3$. Note that the property $C \subset \langle F \rangle$ can be checked by conducting ideal membership test, for instance with Gröbner basis computations.*

3 Methodology

3.1 Tokenization

To better encode the input and output polynomials, four different tokenization schemes are compared, illustrated in Table 1. There are two levels of tokenization in each scheme. The first level focuses on representing expression: (i) **Implicit (Im)**: default expression in common

mathematical software such as Maple; (*ii*) **Explicit (Ex)**: explicit expression with visible unit coefficients and exponents. The second level is the tokenization of numbers: (*i*) **Unsigned (Us)**: the whole unsigned part of every number is regarded as a token; (*ii*) **Digits-split (Ds)**: unsigned part of every number is split into tokens digit by digit.

Table 1: Tokenization Schemes.

Scheme	$36 * x_0^9 * x_1^3 + x_1 * x_2^{12}$
ImUs	[36, $*$, $x0$, $\wedge$, 9, $*$, $x1$, $\wedge$, 3, $+$, $x1$, $*$, $x2$, $\wedge$, 12]
ImDs	[3, 6, $*$, $x0$, $\wedge$, 9, $*$, $x1$, $\wedge$, 3, $+$, $x1$, $*$, $x2$, $\wedge$, 1, 2]
ExUs	[$+$, 36, $*$, $x0$, $\wedge$, 9, $*$, $x1$, $\wedge$, 3, $+$, 1, $*$, $x1$, $\wedge$, 1, $*$, $x2$, $\wedge$, 12]
ExDs	[$+$, 3, 6, $*$, $x0$, $\wedge$, 9, $*$, $x1$, $\wedge$, 3, $+$, 1, $*$, $x1$, $\wedge$, 1, $*$, $x2$, $\wedge$, 1, 2]

3.2 Multi-stage method

As aforementioned, a polynomial can be represented as a sequence of tokens, and therefore polynomial multiplication can be readily mapped to a sequence-to-sequence task, naturally aligning with the encoder-decoder architecture of the transformer model. In the most naive case, called *Direct Strategy* (Strategy 1), we employ the encoder-decoder transformer model to directly learn to predict the final product of two polynomials, treating polynomial multiplication as an atomic operation.

However, if we recall the schoolbook algorithm for polynomial multiplication, the whole computation actually consists of several steps: firstly, the pairwise multiplication of terms is conducted; secondly, the coefficients of like terms are collected; finally, the product is obtained by summing collected coefficients. Aiming to pay extra attention to the intermediate steps, we introduce an intermediate form of the polynomial product, called *collected product*, that is the product after collecting coefficients of like term in the second step without further simplifications. For instance, given two polynomials f and g:

$$f = 5 * x^3 + 3 * x^2 - 4 * x - 8, \quad g = 2 * x^3 - 6 * x^2 - 4 * x - 5, \tag{1}$$

their collected product is

$$\begin{aligned} f \otimes g = & 10 * x^6 \, (-30 + 6) * x^5 \, (-20 - 18 - 8) * x^4 \, (-25 - 12 + 24 - 16) * \\ & x^3 \, (-15 + 16 + 48) * x^2 \, (+20 + 32) * x + 40, \end{aligned} \tag{2}$$

where $f \otimes g$ is called collected product for short.

Thus, the learning task for polynomial multiplication can be divided into a multi-step learning framework: learn the collected product first and then learn to combine the like terms in the collected product. This strategy, called *Collect-Combine Strategy* (Strategy 2), carries out corresponding two subtasks, which trains two separate models for each subtask and sequentially conducts inference on the two models to generate final product. One should nonetheless note that the second subtask of simplification in Strategy 2 involves performing multi-operand integer addition, requiring special attention. As mentioned earlier, there exists a recent state-of-the-art solution for addition of multiple unsigned integers [10]. Inspired by this, Strategy 3, called *Collect-Accumulate Strategy*, is proposed: first, learn the collected product; second, learn to add collected coefficients; third, replace collected coefficients with sums yielded in the second

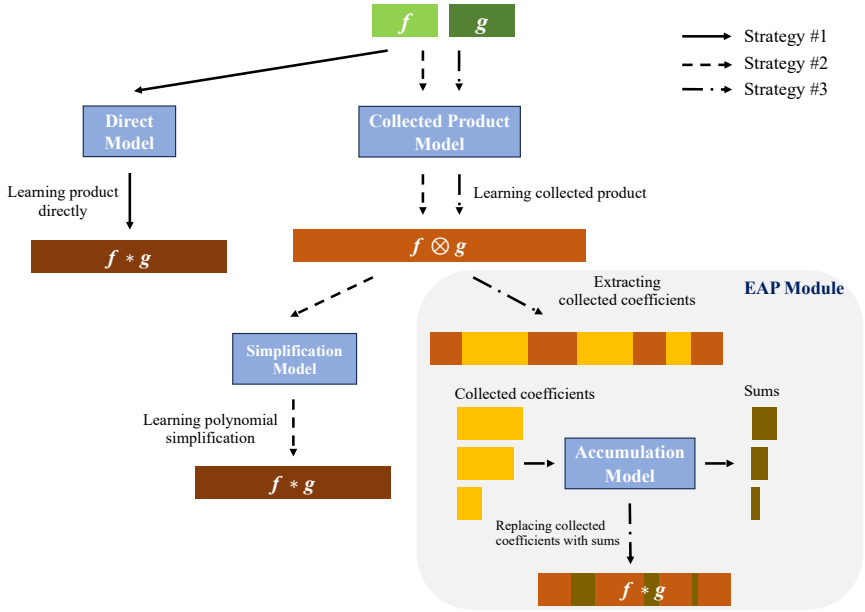

Figure 1: Three different strategies for learning the computation of polynomial products.

INPUT A pair of polynomials	$[5x^3 + 3x^2 - 4x - 8, 2x^3 - 6x^2 - 4x - 5]$ [<s>, +, 5, *, x, ^, 3, +, 3, *, x, ^, 2, −, 4, *, x, ^, 1, −, 8, <op>, +, 2, *, x, ^, 3, −, 6, *, x, ^, 2, −, 4, *, x, ^, 1, −, 5 </s>]
OUTPUT Product of polynomials	$10x^6 - 24x^5 - 46x^4 - 29x^3 + 49x^2 + 52x + 40$ [<s>, +, 1, 0, *, x, ^, 6, −, 2, 4, *, x, ^, 5, −, 4, 6, *, x, ^, 4, −, 2, 9, *, x, ^, 3, +, 4, 9, *, x, ^, 2, +, 5, 2, *, x, ^, 1, +, 4, 0, </s>]
INTERMEDIATE OUTPUT Collected product	$10x^6(-30 + 6)x^5(-20 - 18 - 8)x^4(-25 - 12 + 24 - 16)x^3$ $(-15 + 16 + 48)x^2(+20 + 32)x + 40$ [<s>, +, 1, 0, *, x, ^, 6, (, −, 3, 0, +, 6,), *, x, ^, 5, (, −, 2, 0, −, 1, 8, −, 8,), *, x, ^, 4, (, −, 2, 5, −, 1, 2, +, 2, 4, −, 1, 6,), *, x, ^, 3 (, −, 1, 5, +, 1, 6, +, 4, 8,), *, x, ^, 2, (, +, 2, 0, +, 3, 2,), *, x, ^, 1, +, 4, 0, </s>]
ACCUMULATION SEQUENCES Collected coefficients	[−, 3, 0, +, 6, =, 0, +, >, 0, 3, −, >, 4, 2, −, </s>] [−, 2, 0, −, 1, 8, −, 8, =, 0, +, >, 0, 2, −, >, 8, 3, −, > 6, 4, −, </s>] [−, 2, 5, −, 1, 2, +, 2, 4, −, 1, 6, =, 0, +, >, 5, 2, −, >, 7, 3, −, >, 3, 1, −, >, 9, 2, −, </s>] [−, 1, 5, +, 1, 6, +, 4, 8, =, 0, +, >, 5, 1, −, >, 1, +, >, 9, 4, +, </s>] [+, 2, 0, +, 3, 2, =, 0, +, >, 0, 2, +, >, 2, 5, +, </s>]

Figure 2: An example illustrating the input and output of Strategy 3.

step. Also, Strategy 3 refines the multi-operand addition method to the signed integer case. The learning flowchart of the proposed three strategies is depicted in Figure 1. An example illustrating the input and output of Strategy 3 is shown in Figure 2.

3.3 Addition of multiple signed integers

In the proposed methods, the accumulation model plays a central role, as both Strategy #3 and the strategy for handling long polynomials rely on accurate coefficient accumulation. To support this, a two-layer position coupling mechanism is introduced for signed integer accumulation, as shown in Fig. 3, where a scratchpad is utilized to explicitly record intermediate results, generating digits in reverse in each result.

The first layer (PosID1) performs digit-wise alignment: digits at the same place (units, tens, hundreds, etc.) share the same positional identifiers and thus the same positional embeddings. In this scheme, the sign of an operand is assigned PosID1 = 1, and digit alignment begins from PosID1 = 2 for the units. The second layer (PosID2) couples each digit of the intermediate result (initially zero) with the corresponding digit of the next operand to be added.

The key difference from the unsigned accumulation framework in [10] lies in the handling of plus symbols, as illustrated in Fig. 4. In unsigned setting, plus symbol "+" serves only as a separator, sharing the same PosID1 as delimiter ">", and is coupled with the preceding operand in PosID2. In contrast, the proposed model treats the sign as a meaningful digit in an operand with its own PosID1, allowing sign information to be embedded, enabling accurate signed accumulation crucial for precise coefficient combination in polynomial multiplication.

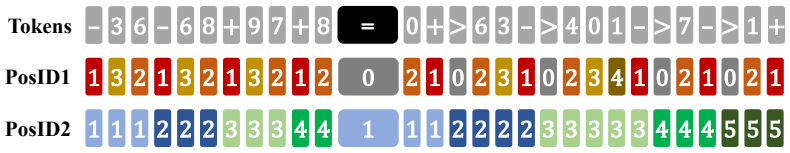

Figure 3: Two-layer position coupling for the accumulation of signed integers.

Cho *et al.*	**Tokens**	28+96+77=0>82>421>102
	PosID1	32132132 121231234 1234
	PosID2	11122233 112223333 4444
Our method	**Tokens**	+28+96+77=0+>82+>421+>102+
	PosID1	132132132 021023102341 02341
	PosID2	111222333 111 222233333 44444

Figure 4: Comparison of our method and the method in [10].

3.4 Divide-and-conquer method

Handling long polynomials incurs evident challenges, particularly due to GPU memory constraints during training. To address this, a structured multi-stage approach is adopted as illustrated in Fig. 5. First, the two input polynomials are sliced into smaller segments. Pairwise slices are then processed to compute collected subproducts. Each collected subproduct is subsequently simplfied using the same procedure as in Strategy #3: identifying coefficients that require accumulation and applying the signed accumulation model to compute their sums. After simplification, subproducts are incrementally merged. Specifically, terms with matching

exponent vectors across subproducts such as (p_{01}, p_{10}) are grouped to form a collected sum $p_{01} \oplus p_{10}$, which collects coefficients in like terms in the same way as computing the collected product. For instance, the collected sum $f \oplus g$ of two polynomials in Eq. 1 is:

$$(+5+2) * x^3 (+3-6) * x^2 (-4-4) * x (-8-5) \tag{3}$$

The collected sum are then handled to the EAP module in Fig. 3 to obtain final sums. This process is repeated until all intermediate results are merged, ultimately yielding the final product.

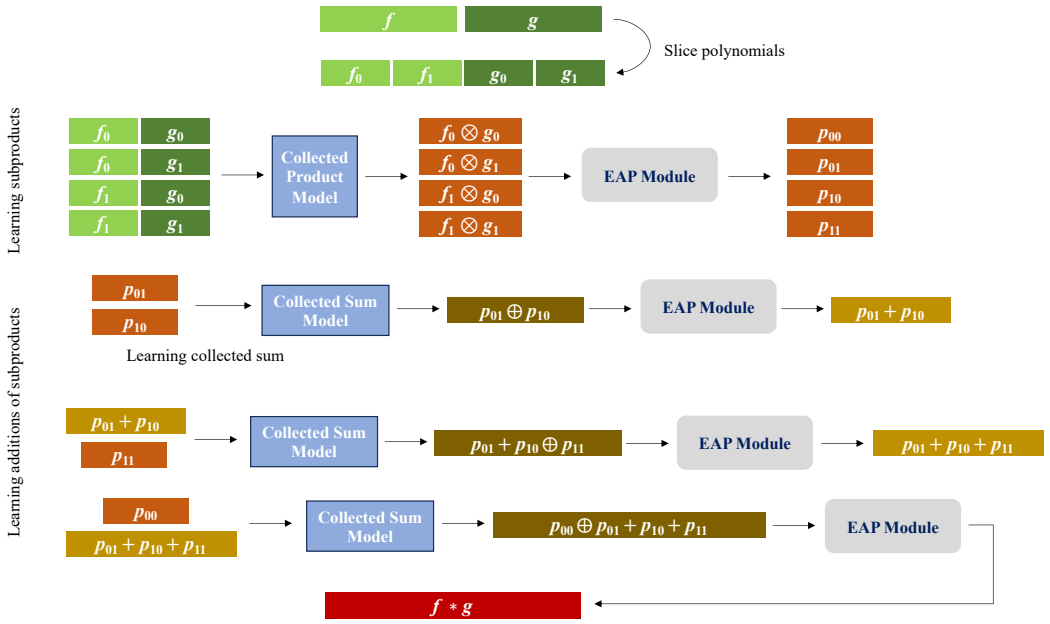

Figure 5: A divide-and-conquer method for learning to compute products of long polynomials.

3.5 Evaluation metrics

We define each metric as a per-example score in [0, 1], and report its accuracy on a dataset as the average of these scores over all examples. Token-level Accuracy (TA) and sequence-level accuracy (SA) are core metrics to evaluate the ability of token prediction and pattern matching of transformer models. Let $\mathbf{y} = (y_1, \ldots, y_t)$ and $\hat{\mathbf{y}} = (\hat{y}_1, \ldots, \hat{y}_s)$ be respectively the target sequence and the predicted sequence.

$$TA = \begin{cases} 0 & \text{if } s \neq t \\ \frac{\sum_{i=1}^{s} \mathbf{1}(y_i = \hat{y}_i)}{s} & \text{if } s = t \end{cases}, \quad SA = \begin{cases} 0 & \text{if } s \neq t \\ \prod_{i=1}^{s} \mathbf{1}(y_i = \hat{y}_i) & \text{if } s = t \end{cases}, \tag{4}$$

where $\mathbf{1}(\cdot)$ is indicator function. While TA and SA are not specifically designed for polynomial data, several additional metrics are proposed to measure how well models learn polynomial structures and arithmetic outcomes.

We use extended Backus-Naur Form (EBNF) to formally define a legal polynomial. Recall that $\{X\}$ means zero or more occurrences of X, and $[X]$ means zero or one occurrence of X.

$$
\begin{aligned}
\langle \text{digit} \rangle &\to \texttt{0} \mid \texttt{1} \mid \cdots \mid \texttt{9} \\
\langle \text{non-zero-digit} \rangle &\to \texttt{1} \mid \texttt{2} \mid \cdots \mid \texttt{9} \\
\langle \text{var} \rangle &\to \texttt{x}\ \{\langle \text{digit} \rangle\} \\
\langle \text{unsigned} \rangle &\to \texttt{0} \mid \langle \text{non-zero-digit} \rangle\ \{\langle \text{digit} \rangle\} \\
\langle \text{signed} \rangle &\to (\texttt{+} \mid \texttt{-})\ \langle \text{unsigned} \rangle \\
\langle \text{power} \rangle &\to \langle \text{var} \rangle\ [\texttt{\^{}}\ \langle \text{unsigned} \rangle] \\
\langle \text{collect} \rangle &\to \texttt{(}\ \langle \text{signed} \rangle\ \langle \text{signed} \rangle\ \{\langle \text{signed} \rangle\}\ \texttt{)} \\
\langle \text{powclct} \rangle &\to \langle \text{power} \rangle\ \langle \text{collect} \rangle \\
\langle \text{factor} \rangle &\to \langle \text{unsigned} \rangle \mid \langle \text{power} \rangle \mid \langle \text{collect} \rangle \mid \langle \text{powclct} \rangle \\
\langle \text{term} \rangle &\to \langle \text{factor} \rangle\ \{\texttt{*}\ \langle \text{factor} \rangle\} \\
\langle \text{poly} \rangle &\to [\texttt{+} \mid \texttt{-}]\ \langle \text{term} \rangle\ \{(\texttt{+} \mid \texttt{-})\ \langle \text{term} \rangle\}
\end{aligned}
$$

Legal accuracy (LA) is 1 if the expression is a legal $\langle \text{poly} \rangle$, and 0 otherwise. LA is the minimal requirement: if a prediction fails the grammar check, all subsequent expression-level metrics for that sample are immediately false. Given a predicted legal expression f and a target polynomial $\hat{f}$ in canonical form, we say that *mathematical accuracy* (MA) is 1 if f defines the same polynomial as $\hat{f}$ after simplification, and 0 otherwise. We say that *canonical accuracy* (CA) is 1 if f defines the same polynomial as $\hat{f}$ after only applying the sorting simplification, and 0 otherwise. At the structural level, metrics must assess the model's ability to learn polynomial structures. At this level, we introduce Degree Accuracy (DA), Number of terms Accuracy (NA), Support Accuracy (SPA), Signed Support Accuracy (SSA), Accuracy of the 1st digits of Coefficients (C1A) and Accuracy of the first 2 digits of Coefficients (C2A), evaluated on mathematically simplified predicted expressions.

These structural metrics are evaluated in a strict hierarchical order on the simplified prediction S_p and simplified target S_t: any failure terminates the chain and leaves all remaining metrics as false for that sample. The strictness is

$$\{\text{DA}, \text{NA}\} > \text{SPA} > \text{SSA} > \text{C1A} > \text{C2A}. \tag{5}$$

- **Degree Accuracy (DA).** Let $D_p = \max_{t \in S_p} \deg(t)$ and $D_t = \max_{t \in S_t} \deg(t)$ be the maximum total degrees of the simplified prediction and target, respectively. DA checks whether $D_p = D_t$.
- **Number-of-Terms Accuracy (NA).** Let $|S_p|$ and $|S_t|$ denote the numbers of terms after simplification. NA checks whether $|S_p| = |S_t|$. Both DA and NA must be true before support-level checks are performed.
- **Support Accuracy (SPA).** SPA checks whether the two polynomials contain exactly the same collection of monomials, ignoring coefficient values.
- **Signed Support Accuracy (SSA).** If SPA is 1, the sign of each monomial's coefficient is compared across prediction and target.
- **Leading-Digit Accuracies (C1A, C2A).** If SSA is 1, each coefficient's absolute value is examined. C1A checks whether the most significant non-zero digit of every coefficient matches; C2A further requires the first two significant digits to match.

Fig. 6 illustrates the hierarchical relationship among these metrics. Sequence Accuracy (SA) is the strictest, while among the metrics designed specifically for polynomials, Canonical Accuracy (CA) is the strictest, and Legal Accuracy (LA) is the most lenient. Table 2 further shows the compliance and violations of different evaluation indicators.

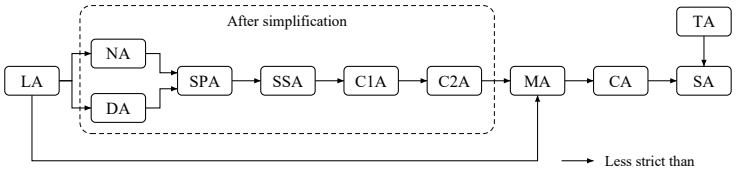

Figure 6: Evaluation metrics.

Table 2: Example of satisfactions and violations in different metrics.

Label	$116 * x_1^9 * x_2^3 - 7 * x_1^2$	
Metric	Satisfaction	Violation
SA	$116 * x_1^9 * x_2^3 - 7 * x_1^2$	$-7 * x_1^2 + 116 * x_1^9 * x_2^3$
CA	$-7 * x_1^2 + 116 * x_1^9 * x_2^3$	$26 * x_1^9 * x_2^3 - 7 * x_1^2 + 90 * x_1^9 * x_2^3$
MA	$26 * x_1^9 * x_2^3 - 7 * x_1^2 + 90 * x_1^9 * x_2^3$	$113 * x_1^9 * x_2^3 - 7 * x_1^2$
C2A	$113 * x_1^9 * x_2^3 - 7 * x_1^2$	$143 * x_1^9 * x_2^3 - 7 * x_1^2$
C1A	$143 * x_1^9 * x_2^3 - 7 * x_1^2$	$14 * x_1^9 * x_2^3 - 7 * x_1^2$
SSA	$14 * x_1^9 * x_2^3 - 7 * x_1^2$	$14 * x_1^9 * x_2^3 + 7 * x_1^2$
SPA	$14 * x_1^9 * x_2^3 + 7 * x_1^2$	$14 * x_1^8 * x_2^3 + 7 * x_1$
NA	$14 * x_1^8 * x_2^3 + 7 * x_1$	$-14 * x_1^8 * x_2^3 + 7 * x_1 + x_2$
DA	$14 * x_1^8 * x_2^4 + 7 * x_1^2$	$4 * x_1^3 * x_2^3 + 7 * x_1$
LA	$4 * x_1^3 * x_2^3 + 7 * x_1$	$4 * x_1^{x_2} * x_2^3 + 8*$

4 Experimentation

In this section, we report on the experimental results for learning to compute the product of two polynomials. Performance is measured in different metrics introduced in Section 3.5, covering both syntactic accuracy (TA and SA) and mathematical correctness (MA and CA, etc.).

Table 3 describes the datasets used in our experiments. Five control parameters v, d, t, e, and c are chosen to randomly generate pairs of polynomials and their products. The five parameters denote respectively the number of variables, the degree of a univariate polynomial, the number of terms, the exponent, and the coefficient. In the dataset names, v, d, and t follow the same definitions as above, while E and C indicate the maximum number of digits allowed in exponents and coefficients, respectively. More precisely, $E1$ indicates that e takes values in 0..9 and $C1$ means that c takes values in $-9..9$. For the accumulation dataset, o is utilized to denote the number of operands in addition. The columns $L^{\text{in}}_{\text{max}}$ and $L^{\text{out}}_{\text{max}}$ denote respectively the maximum number of tokens in the input and output sequence of an example for each dataset. The *Task* column indicates three different tasks in Fig. 1 for learning polynomial multiplication: *Multiply* refers to learning the final product; *Collect* refers to learning the collected product; *Combine* refers to learning to combine the collected product into the final one.

By default, except for the accumulation model, a transformer architecture with standard settings was adopted: 6 encoder/decoder layers, 8 attention heads, embedding dimension 512, hidden dimension 2048 with ReLU activation function for the feed-forward module. A learned absolute position embedding is adopted and the dropout rate during training is set to 0.1. Ranger optimizer [33] is applied to updating weights during training, with initial learning rate of 0.0001. Each training run is allotted up to 400 epochs, with an early stopping strategy that terminates training if the validation sequence accuracy does not improve within a window of 10 consecutive epochs. By default, the dataset is split into training/validation/test in ratio 7:1:2.

Table 3: Description of datasets.

Dataset	Size	v	d	t / o	e	c	Task	$L^{in}_{\max}$	$L^{out}_{\max}$
v1d1C1	100,000	1	1	N/A	N/A	−9..9	Multiply Collect Combine	19 19 24	20 24 20
v1d3C1	100,000	1	3	N/A	N/A	−9..9	Multiply Collect Combine	43 43 84	52 84 52
v1d7C1	100,000	1	7	N/A	N/A	−9..9	Multiply Collect Combine	91 91 281	121 281 121
v1d15C1	100,000	1	15	N/A	N/A	−9..9	Multiply Collect Combine	199 199 969	272 969 272
v2d2C1	100,000	2	2	N/A	N/A	−9..9	Multiply Collect Combine	75 75 214	139 214 139
v3d2C1	100,000	3	2	N/A	N/A	−9..9	Multiply Collect Combine	139 139 604	378 604 378
v2t2E1C1	100,000	2	N/A	2	0..9	−9..9	Multiply Collect Combine	43 43 54	54 54 54
v3t3E1C1	100,000	3	N/A	3	0..9	−9..9	Multiply Collect Combine	87 87 164	164 164 164
C4o9	1,000,000	N/A	N/A	2..9	N/A	−9999.. 9999	Add	112 (Decoder-only)	

For the accumulation model, a decoder-only transformer architecture was employed with 6 decoder layers. A multilevel position coupling scheme serves as the position embedding: PosID1 encodes the digit position within each operand (1 for signs, 2 for units place, 3 for tens place, etc.), while PosID2 encodes the operand position. The model is trained on the signed integer accumulation dataset C4o9 comprising 1,000,000 samples of summing 2–9 signed integers of up to 4 digits each, with intermediate summation results stored in a scratchpad format.

4.1 Comparison between different strategies and the method in [10]

Table 4 compares the performance of three different strategies depicted in Fig. 1 on different types of validation datasets. The tokenization scheme used is ExDs. For multiplying univariate dense polynomials, Direct Strategy only performs well for linear polynomials. As the degree increases, its performance drops dramatically. The two-step Collect-Combine Strategy demonstrates a better performance for trivariate polynomials, but still fails to learn the product for degree seven polynomials. On the contrast, the proposed Collect-Accumulation Strategy performs well on all three datasets.

To compare with [10], we create unsigned versions of the dataset v1d7C1 and C4o9, train an integer accumulation model following their method and apply it on the dataset v1d7C1 (unsigned). The experimental results in Table 5 show that the performance of two methods are

Table 4: Experimental results of three strategies for learning polynomial product.

Type	Dataset	Strategy	TA	SA	MA	CA
Univariate	v1d1C1	1	**100**	**100**	**100**	**100**
		2	99.99	99.97	99.97	99.97
		3	98.84	98.84	**100**	**100**
	v1d3C1	1	85.46	29.54	29.54	29.54
		2	**97.70**	90.20	90.20	90.20
		3	95.09	**95.06**	**99.91**	**99.91**
	v1d7C1	1	6.94	0	0	0
		2	5.15	0	0	0
		3	**90.24**	**90.14**	**99.57**	**99.57**
Multivariate Dense	v2d2C1	1	17.69	0.03	0.03	0.03
		2	97.07	90.62	90.62	90.62
		3	**99.77**	**99.56**	**99.56**	**99.56**
	v3d2C1	1	3.92	0	0	0
		2	0.61	0	0	0
		3	**99.12**	**98.87**	**98.87**	**98.87**
Multivariate Sparse	v2t2E1C1	1	97.33	93.88	96.86	96.05
		2	97.50	94.48	97.92	97.07
		3	**98.42**	**95.35**	**97.95**	**97.95**
	v3t3E1C1	1	95.85	77.07	97.23	96.83
		2	96.39	80.17	98.92	98.51
		3	**96.86**	**80.75**	**99.27**	**99.27**

Table 5: Comparison with [10] on addition of unsigned integers and polynomial multiplication.

Dataset	Method	TA	SA	MA	Dataset	Method	TA	SA	MA
C4o9	[10]	100	100	100	v1d7C1	[10]	98.82	97.92	97.92
(unsigned)	Ours	100	100	100	(unsigned)	Ours	98.94	98.00	98.00

comparable. However, our method can handle both the unsigned and signed case.

4.2 Comparison with LLM

We compare our proposed schemes against three general-purpose LLM configurations on polynomial multiplication: **Qwen-3.5-4B** [36] without deep thinking (Qwen), **Qwen-3.5-4B** with deep thinking (Qwen-think), and **DeepSeek-Math-7B-Base** [30] without deep thinking (DeepSeek). For each dataset, we take the last 100 test instances from the validation dataset. For each test instance, the token limit for Qwen/DeepSeek and Qwen-think is respectively set as 1024 and 35000.

Table 6 reports the experimental results. We have the following observations: (1) Qwen and DeepSeek (without deep thinking) consistently fail on all but the simplest task (v1d1C1); (2) Qwen-think significantly improves accuracy, reaching 91% on v1d3C1 and 96% on v3t3E1C1, but at prohibitive computational cost, taking more than $10,000$ seconds for each of them; (3) Qwen-think still fails entirely on v3d2C1 even taking more than 60,000 seconds for inference; (4) Regarding the running time, all transformer models are orders of magnitude slower than a dedicated polynomial multiplication implementation in the computer algebra system Maple; there is not much difference among dedicated small transformer models; the model with deep thinking is at least an order of magnitude slower than the models without thinking.

These results show that general-purpose LLMs barely perform polynomial multiplication without explicit reasoning chains. However, even with reasoning chains, general-purpose LLMs still perform poorly on multiplication of more complex polynomials in spite of the huge extra

time cost. In contrast, Collect-Accumulate achieves near-perfect accuracy across all datasets (98–100% MA) while requiring only 0.6–3.8 seconds per dataset, three to four orders of magnitude faster than Qwen-think. This dramatic improvement stems from the task-specific architectural design. Rather than relying on general-purpose reasoning, the pipeline decomposes polynomial multiplication into learned subtasks (coefficient collection and signed integer accumulation), each solved by a dedicated transformer model optimized for its specific role.

Table 6: Accuracy and runtime comparison.

Type	Dataset	Model / Method	MA	CA	Total Time (s)
Univariate	v1d1C1	Qwen	33	33	39.44
		Qwen-think	100	100	1998.01
		DeepSeek	27	27	24.91
		Direct	100	100	0.42
		Collect-Combine	100	100	0.77
		Collect-Accumulate	100	100	0.65
		Maple	100	100	0.003
	v1d3C1	Qwen	0	0	107.33
		Qwen-think	91	91	13809.70
		DeepSeek	0	0	93.27
		Direct	35	35	0.37
		Collect-Combine	79	79	1.10
		Collect-Accumulate	100	100	1.29
		Maple	100	100	0.003
	v1d7C1	Qwen	0	0	197.14
		Qwen-think	68	68	34610.37
		DeepSeek	0	0	439.15
		Direct	0	0	0.69
		Collect-Combine	0	0	2.44
		Collect-Accumulate	100	100	3.80
		Maple	100	100	0.003
Multi-Dense	v2d2C1	Qwen	0	0	441.52
		Qwen-think	46	46	43596.81
		DeepSeek	0	0	696.62
		Direct	0	0	0.75
		Collect-Combine	81	81	1.89
		Collect-Accumulate	100	100	1.18
		Maple	100	100	0.004
	v3d2C1	Qwen	0	0	612.20
		Qwen-think	0	0	60244.55
		DeepSeek	0	0	1082.19
		Direct	0	0	1.82
		Collect-Combine	0	0	4.92
		Collect-Accumulate	99	99	2.79
		Maple	100	100	0.009
Multi-Sparse	v2t2E1C1	Qwen	0	0	97.03
		Qwen-think	95	95	15398.13
		DeepSeek	2	2	175.49
		Direct	97	96	0.40
		Collect-Combine	100	99	0.78
		Collect-Accumulate	98	98	0.57
		Maple	100	100	0.002
	v3t3E1C1	Qwen	0	0	565.28
		Qwen-think	96	96	18697.16
		DeepSeek	0	0	1393.77
		Direct	97	96	0.93
		Collect-Combine	99	98	1.65
		Collect-Accumulate	99	99	1.08
		Maple	100	100	0.004

Table 7 further demonstrates the wrongly predicted output of different LLMs on a specific example. For this example, we also include a large-scale frontier model GPT-5.5 (with the prompt "Please directly yield the final polynomial product without thinking."). In the table, the incorrectly predicted tokens are marked in bold.

Table 7: Model comparison on a test case from v1d7C1.

Input	$(-8x^7 - 8x^6 + 3x^5 - x^4 - 2x^3 - 4x^2 + 2x - 8)$ $\times (9x^7 - 8x^5 - 5x^4 + 2x^3 - 2x^2 - 4x + 5)$
Correct Answer	$-72x^{14} - 72x^{13} + 91x^{12} + 95x^{11} - 18x^{10} - 43x^9 + 93x^8$ $- 46x^7 - 50x^6 + 69x^5 + 55x^4 - 14x^3 - 12x^2 + 42x - 40$
Qwen-3.5-4B (without deep thinking)	$-72x^{14}$**$+64x^{13} + 104x^{12} - 128x^{11} + 104x^{10} + 128x^9$** **$-128x^8 - 128x^7 + 128x^6 + 128x^5 - 128x^4 - 128x^3$**$+$**$128x^2 + 128x$**$- 40$
Qwen-3.5-4B (with deep thinking)	$-72x^{14} - 72x^{13} + 91x^{12} + 95x^{11} - 18x^{10} - 43x^9 +$ **71**x^8 **-50**$x^7 - 50x^6 + 69x^5 + 55x^4 - 14x^3 +$**$12$**$x^2 + 42x - 40$
DeepSeek-Math-7B-Base	$-72x^{14}$**$+112x^{13} + 144x^{12} + 112x^{11} + 112x^{10} + 112x^9 + 112x^8 + 112x^7$** $+ 112x^6 + 112x^5$**$+112x^3 + 112x^2 + 112x$**$- 40$
GPT-5.5	$-72x^{14} - 72x^{13} + 91x^{12} +$ **88**$x^{11} -$**11**$x^{10} +$**11**$x^9 +$**17**x^8 **-31**$x^7 -$**24**$x^6 +$**74**$x^5 +$**34**$x^4 -$**18**$x^3 +$**46**$x^2 + 42x - 40$**

4.3 Application on learning pseudo remainders and characteristic sets

In this section, we demonstrate that learning polynomial multiplication helps on learning pseudo division and characteristic set computation, which are higher-level symbolic tasks that internally depend on polynomial multiplication.

Pseudo division. We formulate pseudo division as a sequence-to-sequence task: given two polynomials f and g, the model predicts the pseudo remainder. Method 1 trains a transformer directly to learn this mapping. Method 2 fine-tunes a model pretrained on polynomial multiplication to learn this mapping. The fine-tuning strategy is full-parameter tuning. Two random univariate datasets (v1d3C1, v1d7C1) and one random multivariate dataset (v3t3E1C1L3) are used for the evaluation. The creation of the datasets use several control parameters, where the meaning of v, d, t, C, E is exactly the same as for polynomial multiplication, explained in detail at the beginning of this section. Here, for the dataset v3t3E1C1L3, "L3" indicates that for any example in the dataset, the number of digits appearing in the coefficient of the pseudo remainder is at most 3. For Method 2, the pretrained models for two univariate datasets are the collected product models trained on their corresponding polynomial multiplication datasets v1d3C1 and v1d7C1. The pretrained model for the multivariate dataset v3t3E1C1L3 is the direct model trained on the polynomial multiplication dataset v3t3E1C1. Table 8 reports the experimental results. For all tested three datasets, the pretraining & fine-tuning method outperforms the direct approach, which shows that learning polynomial multiplication do help learning pseudo division.

Characteristic set. The computation of characteristic set can also be formulated directly as a sequence-to-sequence learning task, where the input is set of polynomials and the output is another set of polynomials forming a characteristic set. We also compare with two methods:

Table 8: Evaluation on learning polynomial pseudo division.

Dataset	M	TA	SA	LA	DA	NA	SPA	SSA	C1A	MA	CA
v1d3C1	1	47.43	2.26	**100**	98.49	**100**	97.42	24.95	2.95	2.26	2.26
	2	**98.90**	**97.30**	**100**	**99.96**	**100**	**99.96**	**99.60**	**97.68**	**97.30**	**97.30**
v1d7C1	1	11.87	0	**100**	98.52	**100**	91.66	1.74	0	0	0
	2	**32.47**	**1.62**	**100**	**98.76**	**100**	**92.97**	**90.82**	**2.75**	**1.62**	**1.62**
v3t3E1C1L3	1	84.74	65.85	**100**	98.04	**99.99**	90.18	85.55	81.29	80.71	80.49
	2	**90.28**	**77.64**	99.98	**98.92**	99.97	**93.35**	**90.44**	**87.83**	**87.50**	**87.25**

Method 1 directly trains a transformer model for this task; Method 2 fine-tunes a model pretrained for learning pseudo remainder to predict the characteristic set, which is a natural idea as the computation of characteristic set relies mainly on repeated pseudo division. In practice, to learn characteristic set effectively, we adopt a multi-level pretraining & fine-tuning for Method 2, which will be detailed for each dataset used in our experiments.

Table 9: Evaluation on learning characteristic set.

Dataset	M	pretraining	TA	SA	LA	DA	NA	SPA	C1A	MA	CA
v3t3e2C1L3	1	N/A	60.47	42.12	100	96.54	85.73	78.83	45.36	45.31	42.12
	2	v3t3E1C1L3	**77.50**	**62.32**	**100**	**97.63**	**92.69**	**89.28**	**68.30**	**67.21**	**62.32**
v3t3e2F7	1	N/A	82.05	53.45	100	96.27	84.65	79.30	54.09	54.09	53.45
	2	v3t3E1F7	**84.55**	**59.31**	**100**	**97.23**	**86.85**	**83.93**	**59.99**	**59.99**	**59.31**

Table 9 reports the experimental results on two different randomly generated multivariate datasets. For both datasets, the input consists of two random polynomials. The meaning of the control parameters v, d, t, C, E is exactly the same as for polynomial multiplication and pseudo division. One new control parameter F is introduced. In the table, $F7$ tells that the base ring is the finite field $\mathbb{Z}_7$. In Table 9, the column "pretraining" shows the name of the pretraining pseudo division dataset. For the first characteristic set dataset v3t3e2C1L3, one fine-tunes the fine-tuned pseudo division model on the dataset v3t3E1C1L3 in Table 8 to predict the characteristic set. For the second characteristic set dataset v3t3e2F7, one firstly further fine-tunes the fine-tuned pseudo division model (trained on the dataset v3t3E1C1L3 in Table 8) w.r.t. the pseudo division dataset v3t3E1F7 to obtain a better pseudo division model. This model is then fine-tuned again on the dataset v3t3e2F7 to predict the characteristic set. Note that the range of exponents is randomly taken from 0..2 for both characteristic datasets, whereas this range is enlarged to 0..9 for their corresponding pseudo division datasets. This is done in purpose as multiple pseudo division may be called to compute a characteristic set. As result, the input for intermediate pseudo division may involve polynomials of higher degree. From the table, we observe that multi-level pretraining and fine-tuning starting with multiplication models do help improve the learning of characteristic sets.

4.4 Analyzing the failure/success of three strategies

Although Strategy 1 (S_1) and Strategy 2 (S_2) both yield SA = 0% on v1d7C1 (Table 4), their support accuracies (SPA) under ExDs remain relatively high: 90.20% and 80.81%, respectively. This indicates that both strategies largely identify the correct set of monomials, and their failures stem primarily from incorrect coefficient prediction. To identify the cause, Table 10 reports the per-term prediction accuracy for each monomial in the expanded product, along with its NLT (the maximum number of like terms that must be combined to produce that monomial's coefficient). The product of two degree-7 univariate polynomials always has exactly 15 mono-

Table 10: Per-term accuracies of S_1 and S_2 on dataset $v1d7C1$.

Monomial	NLT[a]	S_1	S_2	Monomial	NLT	S_1	S_2	Monomial	NLT	S_1	S_2
x^{14}	1	45.61	3.01	x^9	6	0.97	0.38	x^4	5	1.02	0.42
x^{13}	2	28.20	0.99	x^8	7	0.90	0.35	x^3	4	3.29	0.52
x^{12}	3	10.59	0.68	x^7	8	0.78	0.39	x^2	3	10.61	0.68
x^{11}	4	3.54	0.55	x^6	7	0.90	0.29	x^1	2	28.04	1.58
x^{10}	5	1.53	0.36	x^5	6	0.85	0.46	x^0	1	47.97	44.41

[a] Maximum number of like terms requiring combining.

mials of degrees $x^{14}, \ldots, x^0$; each monomial (a *term*) combines $n(d)$ like-term contributions, where $n(d) = d + 1$ for $d \leq 7$ and $15 - d$ for $d > 7$. For each degree d we keep two counters over the validation set: TC[d], the number of degree-d terms in the *target* polynomial product, and CC[d], how many of these have the correctly predicted coefficients. Per-term accuracy is defined as CC[d]/TC[d].

For both S_1 and S_2, per-term accuracy is lower than 50% and exhibits a strong negative correlation with NLT. Note that combining k like terms requires both identifying the same monomial and summing k signed integers correctly. Given the fact that the support accuracies for both stategies are higher than 80%, these results reveal that the fundamental bottleneck is neither support identification nor polynomial structure learning, but rather *signed integer accumulation*, that is learning to add exactly multiple signed integers to form each coefficient. This motivates Strategy 3 (Collect-Accumulate), which isolates accumulation into a dedicated transformer model trained specifically for this arithmetic subtask. Table 11 confirms that the accumulation model achieves near-perfect accuracy.

Table 11: Accuracy of the integer accumulation model on signed integers.

Dataset	Tokenization	TA	SA	LA	DA	NA	SPA	SSA	C1A	C2A	MA	CA
C4o9	N/A	99.98	99.99	N/A	N/A	N/A	N/A	N/A	N/A	N/A	99.99	N/A

5 Conclusion

In this work, we propose a multi-stage transformer-based approach for learning the multiplication of two integer polynomials. Instead of directly modelling the polynomial multiplication as a one-step sequence-to-sequence task, the method introduces an intermediate product to learn and extends state-of-the-art work on learning to add multiple unsigned integers to the case of signed integers. Its effectiveness are demonstrated by running experiments on various dense univariate and sparse multivariate datasets against the direct and the two-step baseline approaches. More importantly, we demonstrate that learning polynomial multiplication helps on learning a more advanced operation polynomial pseudo division through a pretraining & fine-tuning scheme. As pseudo division is the core operation in Ritt-Wu's characteristic set method for solving polynomial equations, we further fine-tune the obtained models for performing pseudo division to generate characteristic sets. Experiments show that this multi-level pretraining and fine-tuning strategy substantially outperforms a direct transformer based approach. To date, we have trained transformer models only to learn polynomial multiplication over the ring of integers. In future work, we plan to extend this approach to finite fields.

References

[1] Janice Ahn, Rishu Verma, Renze Lou, Di Liu, Rui Zhang, and Wenpeng Yin. Large language models for mathematical reasoning: Progresses and challenges. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics: Student Research Workshop*, pages 225–237. Association for Computational Linguistics, 2024.

[2] Christopher Brown and James Davenport. The complexity of quantifier elimination and cylindrical algebraic decomposition. In *Proceedings of the 2007 International Symposium on Symbolic and Algebraic Computation*, pages 54–60, 2007.

[3] Christopher Brown and Glenn Christopher Daves. Applying machine learning to heuristics for real polynomial constraint solving. In *International Congress on Mathematical Software*, pages 292–301. Springer, 2020.

[4] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 33:1877–1901, 2020.

[5] François Charton. Learning the greatest common divisor: explaining transformer predictions. In *Proceedings of the 12th International Conference on Learning Representations (ICLR 2024)*. Curran Associates, Inc., 2024.

[6] Changbo Chen, Rui-Juan Jing, Chengrong Qian, Yaru Yuan, and Yuegang Zhao. A dataset for suggesting variable orderings for cylindrical algebraic decompositions. In *Computer Algebra in Scientific Computing: 26th International Workshop*, page 100–119. Springer, 2024.

[7] Changbo Chen and Marc Moreno Maza. Algorithms for computing triangular decomposition of polynomial systems. *Journal of Symbolic Computation*, 47(6):610–642, 2012.

[8] Changbo Chen, Zhangpeng Zhu, and Haoyu Chi. Variable ordering selection for cylindrical algebraic decomposition with artificial neural networks. In *International Congress on Mathematical Software*, pages 281–291. Springer, 2020.

[9] Hanseul Cho, Jaeyoung Cha, Pranjal Awasthi, Srinadh Bhojanapalli, Anupam Gupta, and Chulhee Yun. Position coupling: Improving length generalization of arithmetic transformers using task structure. *Advances in Neural Information Processing Systems*, 37:22233–22315, 2024.

[10] Hanseul Cho, Jaeyoung Cha, Srinadh Bhojanapalli, and Chulhee Yun. Arithmetic transformers can length-generalize in both operand length and count. In *Proceedings of the 13th International Conference on Learning Representations*. Curran Associates, Inc., 2025.

[11] David Cox, John Little, and Donal O'Shea. *Ideals, varieties, and algorithms*. Springer, 1997.

[12] Stéphane d'Ascoli, Pierre-Alexandre Kamienny, Guillaume Lample, and François Charton. Deep symbolic regression for recurrence prediction. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162, pages 4520–4536. PMLR, 2022.

[13] Dorian Florescu and Matthew England. A machine learning based software pipeline to pick the variable ordering for algorithms with polynomial inputs. In *International Congress on Mathematical Software*, pages 302–311. Springer, 2020.

[14] John Harrison. A short survey of automated reasoning. In *Proceedings of the International Conference on Algebraic Biology*, pages 334–349. Springer, 2007.

[15] Samy Jelassi, Stéphane d'Ascoli, Carles Domingo-Enrich, Yuhuai Wu, Yuanzhi Li, and François Charton. Length generalization in arithmetic transformers. *arXiv preprint arXiv:2306.15400*, 2023.

[16] Fuqi Jia, Yuhang Dong, Minghao Liu, Pei Huang, Feifei Ma, and Jian Zhang. Suggesting variable order for cylindrical algebraic decomposition via reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, pages 76098–76119. Curran Associates, Inc., 2023.

[17] Rui-Juan Jing, Yuegang Zhao, and Changbo Chen. Breaking the data barrier in learning symbolic computation: A case study on variable ordering suggestion for cylindrical algebraic decomposition. *Journal of Symbolic Computation*, 138:102607, 2027.

[18] Ruijuan Jing, Chengrong Qian, and Changbo Chen. Variable ordering selection for cylindrical algebraic decomposition via reinforcement learning. *Journal of Systems Science and Mathematical Sciences*, 44(9):2826–2849, 2024.

[19] Dejan Jovanović and Leonardo de Moura. Solving non-linear arithmetic. In *Proceedings of the 6th International Joint Conference on Automated Reasoning*, pages 339–354. Springer, 2012.

[20] Amirhossein Kazemnejad, Inkit Padhi, Karthikeyan Natesan Ramamurthy, Payel Das, and Siva Reddy. The impact of positional encoding on length generalization in transformers. *Advances in Neural Information Processing Systems*, 36:24892–24928, 2023.

[21] Hiroshi Kera, Shun Arakawa, and Yuta Sato. CALT: A library for computer algebra with transformer. *ACM Communications in Computer Algebra*, 59(4):121–128, 2025.

[22] Hiroshi Kera, Yuki Ishihara, Yuta Kambe, Tristan Vaccon, and Kazuhiro Yokoyama. Learning to compute Gröbner bases. In *Advances in Neural Information Processing Systems*, pages 33141–33187. Curran Associates, Inc., 2024.

[23] Guillaume Lample and François Charton. Deep learning for symbolic mathematics. In *Proceedings of the 8th International Conference on Learning Representations (ICLR 2020)*, volume 14, pages 10442–10465. Curran Associates, Inc., 2020.

[24] Mohamed Malhou, Ludovic Perret, and Kristin Lauter. HATSolver: Learning Gröbner bases with hierarchical attention transformers. In *Proceedings of the 14th International Conference on Learning Representations*, 2026.

[25] Ernst W Mayr and Albert R Meyer. The complexity of the word problems for commutative semigroups and polynomial ideals. *Advances in Mathematics*, 46(3):305–329, 1982.

[26] Sean McLeish, Arpit Bansal, Alex Stein, Neel Jain, John Kirchenbauer, Brian Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, Jonas Geiping, Avi Schwarzschild, and Tom Goldstein. Transformers can do arithmetic with the right embeddings. *Advances in Neural Information Processing Systems*, 37:108012–108041, 2024.

[27] Rodrigo Nogueira, Zhiying Jiang, and Jimmy Lin. Investigating the limitations of transformers with simple arithmetic tasks. In *The 1st Mathematical Reasoning in General Artificial Intelligence Workshop at ICLR*, 2021.

[28] Dylan Peifer, Michael Stillman, and Daniel Halpern-Leistner. Learning selection strategies in Buchberger's algorithm. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119, pages 7575–7585, 2020.

[29] Joseph Fels Ritt. *Differential equations from the algebraic standpoint*, volume 14. American Mathematical Society, 1932.

[30] Zhihong Shao, Peilin Wang, Qihao Zhu, Runxin Xu, et al. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

[31] Joachim Von Zur Gathen and Jürgen Gerhard. *Modern computer algebra*. Cambridge University Press, 2003.

[32] Dongming Wang. *Elimination Methods*. Springer, New York, 2000.

[33] Less Wright and Nestor Demeure. Ranger21: a synergistic deep learning optimizer. *arXiv preprint arXiv:2106.13731*, 2021.

[34] Wen-Tsun Wu. Basic principles of mechanical theorem proving in elementary geometries. *Journal of Automated Reasoning*, 2(3):221–252, 1986.

[35] Wen-Tsun Wu and Xiao-Shan Gao. Mathematics mechanization and applications after thirty years. *Frontiers of Computer Science in China*, 1(1):1–8, 2007.

[36] An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, et al. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

[37] Doron Zeilberger. A fast algorithm for proving terminating hypergeometric identities. *Discrete Mathematics*, 80(2):207–211, 1990.

[38] Yongchao Zhou, Uri Alon, Xinyun Chen, Xuezhi Wang, Rishabh Agarwal, and Denny Zhou. Trans-

formers can achieve length generalization but not robustly. In *The 2nd Workshop on Mathematical and Empirical Understanding of Foundation Models at ICLR*, 2024.