Global Lyapunov functions: a long-standing open problem in mathematics, with symbolic transformers

Alberto Alfarano[*]
FAIR, Meta
albealfa@meta.com

François Charton[*]
FAIR, Meta – CERMICS, Ecole des Ponts
fcharton@meta.com

Amaury Hayat[*]
CERMICS, Ecole des Ponts — Institut Polytechnique de Paris
amaury.hayat@enpc.fr

Abstract

Despite their spectacular progress, language models still struggle on complex reasoning tasks, such as advanced mathematics. We consider a long-standing open problem in mathematics: discovering a Lyapunov function that ensures the global stability of a dynamical system. This problem has no known general solution, and algorithmic solvers only exist for some small polynomial systems. We propose a new method for generating synthetic training samples from random solutions, and show that sequence-to-sequence transformers trained on such datasets perform better than algorithmic solvers and humans on polynomial systems, and can discover new Lyapunov functions for non-polynomial systems.

1 Introduction

As large language models achieve human-level performance over a broad set of tasks [4, 35, 45], their capability to *reason* becomes a focus of discussion and research. There is no single definition of reasoning, and work in this area encompasses factuality, real world alignment, compositionality, the discovery and following of rules, &c. Still, mathematics are considered as one of the purest, and most demanding, forms of reasoning [17]. As such, solving research-level mathematical problems is a major milestone in demonstrating the reasoning capabilities of language models. Such an advance in AI would also transform mathematical practice.

There is little research on applying language models to open problems of mathematics. Except a few papers on combinatorial optimization and graph theory [34, 39], most prior works focus on problems with known solutions [37, 23, 30, 8]. We believe this lack of results is due to two main reasons. First, research problems may require specialized work by mathematicians [6] before they can be handed to language models. Second, most math transformers are trained on sets of problems and solutions which are hard to generate in the case of open problems, when no generic method for finding a solution is known.

In this paper, we focus on a long-standing, yet easy to formalize, open problem in mathematics: discovering the Lyapunov functions that control the global stability of dynamical systems – the boundedness of their solutions when time goes to infinity with respect to an equilibrium or an orbit. A famous instance of this problem is the *three-body problem*: the long-term stability of a system of three celestial bodies subjected to gravitation. The stability problem was studied by Newton, Lagrange and Poincaré. Lyapunov discovered that stability is guaranteed if an entropy-like function for the system

---

[*]Equal contribution

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

–the Lyapunov function– can be found. Unfortunately, no method is known for deriving Lyapunov functions in the general case, and Lyapunov functions are only known for a small number of systems.

We propose a new technique for generating training data from randomly sampled Lyapunov functions. Sequence-to-sequence transformers trained on these datasets achieve near perfect accuracy (99%) on held-out test sets, and very high performance (73%) on out-of-distribution test sets. We show that higher accuracies (84%) can be achieved by enriching the training set with a small number (300) of easier examples that can be solved with existing algorithmic methods. These enriched models greatly outperform state-of-the-art techniques and human performance on a variety of benchmarks.

Finally, we test the capability of our models to discover yet unknown Lyapunov functions on randomly generated systems. On polynomial systems, the only ones current methods can solve, our models find Lyapunov function for 10.1% or systems, vs 2.1% for state-of-the-art techniques. On non-polynomial systems, where no algorithm is known, our best models discover new Lyapunov functions for 12.7% of systems. Our research demonstrates that generative models can be used to solve research-level problems in mathematics, by providing mathematicians with guesses of possible solutions. The solutions proposed by the black-box model are explicit and their mathematical correctness can be verified. We believe this research is an AI-driven blueprint for solving open problems in mathematics.

Related works

Most classical methods for finding Lyapunov rely on parameterized families of candidate solutions, and attempt to derive conditions on the parameters [11, 14]. Additional techniques such as backstepping or forwarding [11, Chap. 12], were introduced to leverage the specifics of particular systems (see also physics-based methods [44]). These techniques are limited to specific, or simple, systems. The global Lyapunov functions of polynomial systems that are sums of squares of polynomials of given degree can be found by computational-intensive algorithmic tools, such as SOSTOOLS [32, 33], which leverage the fact that the Lyapunov function belongs to a finite-dimensional space.

Methods involving neural networks have been proposed in recent years [7, 15, 12, 24, 25]. They train feed-forward networks to approximate Lyapunov functions of a given system, and use a Satisfiability Modulo Theories (SMT) solver as a verifier which proposes potential counter-examples. This approach, very different from ours, was shown to be successful for several well-studied high dimensional systems. However, it only finds local or semi-global Lyapunov functions (see Definition A.3). Since the Lyapunov functions that are found are implicit, it would be hard for mathematicians to check whether they are global Lyapunov functions or not. Semi-global Lyapunov functions are useful in many engineering fields such as robotics, where one wants a system to be robust to small perturbations. In other fields, like epidemics, being resilient to large perturbations is central, and global Lyapunov functions are required.

Transformers trained on synthetic datasets have been proposed for many problems of mathematics, including arithmetic [28], linear algebra [10], symbolic integration [22], symbolic regression [5], Shortest Vector Problem [40], Gröbner basis computation [19] and theorem proving [30]. [8] investigate a problem related to ours: the local stability of dynamical systems. Different architectures were used to solve hard problems in combinatorial optimisation [34], and graph theory [39].

2   System stability and Lyapunov functions

The stability of dynamical systems is a hard mathematical question, which intrigued many generations of mathematicians, from Newton and Lagrange in the 18th century, to Poincaré in the 20th in the context of the three-body problem. The main mathematical tool for assessing stability was proposed by Lyapunov, who showed in 1892 that a system is stable if a decreasing entropy-like function –the Lyapunov function– can be found [20, 11, 26]. Later, the existence of a Lyapunov function was shown to be a necessary condition for the stability of large classes of systems [29, 27, 18]. Unfortunately, these very strong results provide no clue on how to find Lyapunov functions, or just proving their existence for a particular system. In fact, 130 years later, systematic derivations of global Lyapunov functions are only known in a few special cases, and their derivation in the general case remains a well-known open problem.

In mathematical terms, we consider the dynamical system

$$\dot{x} = f(x), \tag{1}$$

2

$$\begin{cases} \dot{x}_0 = -7x_0^5 - 4x_0^3x_1^2 - 5x_0^3 \\ \dot{x}_1 = 7x_0^4 - 3x_1 - 2x_2 \\ \dot{x}_2 = -8x_0^2 - 9x_2 \end{cases}$$

$$V(x) = 2x_0^4 + 2x_0^2x_1^2 + 3x_0^2 + 2x_1^2 + x_2^2$$

$$\begin{cases} \dot{x}_0 = -x_0 + x_0x_1 \\ \dot{x}_1 = -x_1 \end{cases}$$

$$V(x) = \ln(1 + 5x_0^2) + x_1^2$$

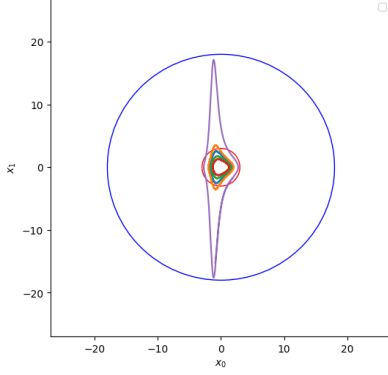



Figure 1: Dynamic of a stable system: trajectories may be complicated but as long as they start in the red ball they remain in the blue ball.

Figure 2: Two stable systems and associated Lyapunov functions discovered by our model. The second, a polynomial system with a non-polynomial Lyapunov function, was studied in [1].

where $x \in \mathbb{R}^n$, $f \in C^1(\mathbb{R}^n)$ and $\dot{x} = \frac{dx}{dt}$. We want to know if the system has a stable equilibrium around a point x^* such that $f(x^*) = 0$. We assume, without loss of generality, that $x^* = 0$.

Definition 2.1. The system (1) is *stable* when, for any $\varepsilon > 0$, there exists $\eta > 0$ such that, if $\|x(0)\| < \eta$, the system (1) with initial condition $x(0)$ has a unique solution $x \in C^1([0, +\infty))$ and

$$\|x(t)\| \leq \varepsilon, \quad \forall\, t \in [0, +\infty). \tag{2}$$

In other words, a system is stable if a solution that begins close to the origin ($\|x(0)\| < \eta$) stays close to the origin at all time ($\|x(t)\| \leq \varepsilon$). Lyapunov proved that the stability is related to the existence of what is now called a Lyapunov function.

Definition 2.2. The function $V \in C^1(\mathbb{R}^n, \mathbb{R}_+)$ is said to be a *(global) Lyapunov function* for the system (1) if the following condition are satisfied

$$\begin{aligned} V(0) = 0, \quad &\lim_{\|x\| \to +\infty} V(x) = +\infty, \\ V(x) > 0, \quad &\nabla V(x) \cdot f(x) \leq 0 \text{ for } x \neq 0. \end{aligned} \tag{3}$$

Theorem 2.3 (Lyapunov 1892). *If the system* (1) *has a Lyapunov function, then it is stable.*

In fact, the existence of a Lyapunov function is more powerful and provides additional information.

Theorem 2.4 (LaSalle, 1961). *If the system* (1) *has a Lyapunov function V, then all the solutions of* (1) *converge to the largest invariant set of $\{f(x) \cdot \nabla V(x) = 0\}$.*

In many cases this largest invariant set is reduced to $\{x^* = 0\}$ and the system is said *globally asymptotically stable* (all solutions converge to the equilibrium, see Appendix A).

Most dynamical systems are unstable. For instance, the solutions of the simple system $\dot{x}(t) = x(t)$ grow exponentially with time, and the solutions of $\dot{x}(t) = 1 + x(t)^2$ ($x \in \mathbb{R}$) always blow up before $t = \pi$. No Lyapunov functions can be found for these systems.

On the other hand, stable systems can have an infinite number of Lyapunov functions. The system

$$\begin{cases} \dot{x}_0(t) = -x_0(t) \\ \dot{x}_1(t) = -x_1(t) \end{cases}$$

has $V(x) = a_0x_0^2 + a_1x_1^2$ as a Lyapunov function for any choice of $a_0 > 0$ and $a_1 > 0$.

In the general case, there is no systematic way of discovering a Lyapunov function, or even showing that one exist. Tools exist for small polynomial systems with special "sum of squares" (SOS) Lyapunov functions, but they need a lot of resources, do not always find a solution, and fail once the systems involve more than a few variables.

We also consider a related, but easier, problem: finding nontrivial V which are semi-definite positive, i.e. V verifying $V(x) \geq 0$ instead of $V(x) > 0$ in Equation (3). These functions, called *barrier*

3

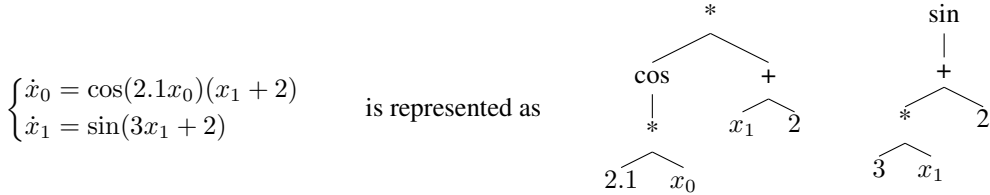
functions, form "barriers" that divide $\mathbb{R}^n$ into two subspaces. A solution starting inside the barrier must remains in the same subspace, which is an invariant set of the system [31, 41]. For polynomial systems, barrier functions are slightly easier to find using SOS solvers.

3 Experimental settings

In this work, we train sequence-to-sequence transformers [38] to predict a Lyapunov function for a given system, when it exists. We frame the problem as a translation task: problems and solutions are represented as sequences of symbolic tokens, and the model is trained from generated pairs of systems and Lyapunov functions to minimize the cross-entropy between the predicted sequence and the correct solution. We train transformers with 8 layers, 10 attention heads and an embedding dimension of 640 (ablation studies on different model sizes can be found in Appendix C), on batches of 16 examples, using the Adam optimizer [21] with a learning rate of 10^{-4}, an initial linear warm-up phase of $10,000$ optimization steps, and inverse square root scheduling. All experiments run on 8 V100 GPU with 32 GB of memory, for 3 or 4 epochs of 2.4 million examples per epoch. Training time is between 12 to 15 hours per GPU.

Tokenization. Model inputs are systems of the form $(\dot{x}_i = f_i(x_1, \ldots, x_n))_{i \in \{1,\ldots,n\}}$, represented by the n functions f_i. Model outputs are single functions $V(x_1, \ldots, x_n)$. As in [22], functions are represented as trees, with operators in their internal nodes, and variables or constants as their leaves. Trees are then enumerated in Polish (pre-order) notation to produce sequences of tokens that can be processed by the transformer.

All operators and variables are tokenized as single symbols (e.g. 'cos' or 'x_1'). Integer constants are tokenized as sequences of "digits" in base 1000 (e.g. 1024 as the sequence [+, 1, 24]), and real constants, in scientific notation, as pairs of two integers (mantissa and exponent, e.g. -3.14 as [-, 314, 10^, -, 2]). For instance:

$$\begin{cases} \dot{x}_0 = \cos(2.1x_0)(x_1 + 2) \\ \dot{x}_1 = \sin(3x_1 + 2) \end{cases} \qquad \text{is represented as}$$

enumerated as the sequences: $[*, \cos, *, 2.1, x_0, +, x_1, 2]$ and $[\sin, +, *, 3, x_1, 2]$, and finally tokenized as [*, cos, *, 21, 10^, -, 1, x_0, +, x_1, 2, SEP, sin, +, *, 3, x_1, 2] (using SEP as a separator).

Evaluation. Trained models are evaluated on sets of stable systems. Since systems have an infinite number of Lyapunov functions, we cannot check the model predictions by comparing them to the solutions from the test set, and need to use an external verifier. For polynomial systems, we verify that there exists a small positive polynomial P such that $-\nabla V \cdot f$ and $V - P$ are sum of squares (SOS) of polynomials (with $P = 0$ for barrier functions), using a Python solver based on SumOfSquares [43]. For non-polynomial systems, we also use a verifier based on `shgo` that checks (3) numerically. To further ensure correctness we also verify the symbolic solutions using Satisfiability Modulo Theories (SMT) solvers, relying on dReal [13] for verification through interval analysis. This guarantees that equations (3) hold, at least in a chosen ball around the origin. The performances of the two verifiers (numerical solver and SMT) are similar, a comparison is provided in Table 1. Both the SOS and SMT verifiers sometimes fail to return an answer. In that case, we classify the solution as wrong, even though it might have been correct. As a result, model accuracies may be underestimated.

Model predictions use beam search with early stopping, normalizing log-likelihood scores by their sequence length. We report results with beam size 1 (greedy decoding) and beam size 50. With beam size 50, we consider the model to be correct if one Lyapunov function is found among the 50 guesses.

4 Data generation

Our models are trained and tested on large datasets of pairs of stable systems and associated Lyapunov functions. Sampling such stable systems raises two difficulties. First, most dynamical systems are unstable, and no general method exists for deciding whether a system is stable. Second, once a stable

4

system is sampled, there is no general technique for finding a Lyapunov function, except in particular cases. In this paper, we rely on **Backward generation** [22], sampling solutions and generating associated problems, for the general case, and **forward generation**, sampling systems and calculating their solutions with a solver, for the tractable polynomial systems of small degree.

4.1 Backward generation

Backward generation methods, sampling problems from their solutions, are only useful if the model can be prevented from learning to reverse the generation procedure, or from "reading" the solutions in the generated problems. For instance, when training a model to solve the hard problem of finding the roots of an integer polynomial [9], one can easily generate a polynomial from its roots, i.e. from the roots $3, 5$ and 7, generate the polynomial:

$$P(X) = 2(X^2 + 1)(X - 3)(X - 5)(X - 7).$$

However, if the model is trained from factorized form of $P(X)$, it will learn to read the roots in the problem, instead of computing them. On the other hand, the developed and simplified form

$$P(X) = 2X^5 - 30X^4 + 144X^3 - 240X^2 + 142X - 210$$

offers no clues. A second difficulty of backward generation is that sampling solutions instead of problems biases the training distribution. A model trained on backward-generated data may not perform well on a forward-generated test set. Finally, prior work [42] observed that, for hard problems, backward generation methods sometimes focus on easier sub-problems (see, for instance, our comment below about choosing $f = -\nabla V$ in step 2).

We propose a procedure for generating a stable system S from a random Lyapunov function V. The rationale is the following. Since V must be positive with a strict minimum in 0, and tend to infinity at infinity ((3)), we first generate $V = V_{\text{proper}} + V_{\text{cross}}$ where V_{proper} belongs to a class of functions with a guaranteed strict minimum in zero and V_{cross} to a larger class of non-negative functions, valued 0 at the origin, but with no guarantee of a strict minimum (step 1 and Appendix B). From V, we need to generate f so that the third condition of (3) is met. A naive solution would be $f = -\nabla V$ since $f \cdot \nabla V \le 0$ would hold. But this would severely limit the systems we create, and turn the Lyapunov function discovery problem (find V from f) into an easier integration problem (find V from $-\nabla V$). Instead, starting from $f_0 = -\nabla V$, we apply the following transformations:

- multiply each coordinate of f_0 by random non-negative functions h_i^2 (step 4) and call it $\tilde{f}_0$.
- generate a random function $\phi = \sum_{i=1}^{p} g_i(x)e^i(x)$ (steps 2 and 3), where e^i are orthogonal to $\nabla V(x)$, and set $f = \varphi + \tilde{f}_0$. We have $\phi \cdot \nabla V = 0$ and $(\phi + \tilde{f}_0) \cdot \nabla V \le 0$.

These transformations guarantee that all conditions in (3) are met. On the other hand, they allow f to span a very large set of systems, since any f satisfying $\nabla V(x) \cdot f(x) \le 0$ can be written as the sum of a function collinear to $\nabla V(x)$ and a function orthogonal to $\nabla V(x)$.

Specifically, the procedure can be summarized as follows (see Appendix B for more details).

> **Step 1** Generate a random function V, satisfying $V(x) > V(0), \quad \forall x \in \mathbb{R}^n \setminus \{0\}$, and $V(x) \to +\infty$ when $\|x\| \to +\infty$.
> **Step 2** Compute the gradient $\nabla V(x)$ and denote $\mathcal{H}_x = \{z \in \mathbb{R}^n \mid z \cdot \nabla V(x) = 0\}$ the hyperplane[2] orthogonal to $\nabla V(x)$, for any $x \in \mathbb{R}^n$.
> **Step 3** Select $1 \le p \le n$ at random and sample p vectors $\{e^i(x)\}_{i \in \{1,\dots,p\}}$ from hyperplane $\mathcal{H}_x$. Generate p real-valued functions $(g_i)_{i \in \{1,\dots,p\}}$.
> **Step 4** Select $1 < k_1 \le n$ at random, generate k_1 random real-valued functions $(h_i)_{i \in \{1,\dots,k_1\}}$, set $h_i = 0$ for $k_1 + 1 \le i \le n$.
> **Step 5** Build the n functions
>
> $$f(x) = -(h_{\pi(i)}^2(x)(\nabla V)_i(x))_{i \in \{1,\dots,n\}} + \sum_{i=1}^{p} g_i(x)e^i(x),$$
>
> with π a random permutation of $\{1, \dots, n\}$.
> **Step 6** Simplify the functions f_i, obscuring patterns from the generative process.

---

[2] if $\nabla V(x) = 0$ this is the whole space instead, but this does not change the method.

5

	SMT Solver		SOS Solver	
Timeout	10 minutes	60 minutes	10 minutes	60 minutes
Correct Lyap function	82.6	94.1	89.6	95.3
Solver Timeouts	17.4	5.9	10.4	4.7
Incorrect Lyap function	0	0	0	0

Table 1: **SMT and SOS timeout and error rates,** benchmarked on correct Lyapunov functions.

This method produces a stable system $S : \dot{x} = f(x)$, with V as its Lyapunov function. The difficulty of inferring V from S hinges on a careful choice of the vectors e^i. For instance, if we naively select e^i as an orthonormal basis of $\mathcal{H}_x$, computed from $\nabla V(x)$ by Gram-Schmidt orthogonalization, prefactors like $1/\|\nabla V(x)\|$ appear at step 3, and are unlikely to simplify away at step 6. This provides the model with a shortcut: reading $\|\nabla V(x)\|$ in S, and using it to recover ∇V and then V, not a trivial task, but an easier one than discovering Lyapunov functions. To counter this, we relax the orthonormality condition on $e^i(x)$, so that $1/\|\nabla V(x)\|$ never appears, yet keep the $e^i(x)$ simple enough for ∇V-specific patterns in $\sum_i g_i(x)e^i(x)$ to simplify away at step 6. We also want to ensure that the e^i span all of $\mathcal{H}_x$, or the systems generated will not be diverse enough.

In our experiments, we slightly modify this procedure, by running steps 2 to 6 five times for each Lyapunov function V created at step 1. As a result, 5 systems are generated that share the same Lyapunov function (a discussion of this choice can be found in Appendix C.1). From a mathematical point of view, a Lyapunov function describes a hidden quantity in a system, and we believe that providing the model with several systems that share this hidden quantity should help it learn the parts of the system that contribute to this hidden quantity, and therefore learn a Lyapunov function.

This procedure can be tuned to generate specific classes of systems. By choosing V, g_i and h_i in particular classes, we can constrain the system functions f_i to be polynomials, polynomials of functions (e.g. trigonometric polynomials), or more general functions (see Appendix B.4 for more).

The Lyapunov functions obtained here are correct by design. Nevertheless, we still performed an evaluation of the solutions both as a safeguard and to benchmark the failure and timeout rates of the SMT and SOS solvers on correct solutions, which we report in Table 1.

4.2 Forward generation

Whereas the stability problem is unsolved in the general case, methods exist to calculate Lyapunov functions of polynomial systems, when they exist and can be written as a sum of squares of polynomials (see Section 1). These algorithms, of polynomial complexity, are very efficient for small systems, but their CPU and memory requirements explode as the size of the systems grows. We leverage them to generate forward datasets, as follows.

> **Step 1** Generate a polynomial system at random
> **Step 2** Use a routine to find a polynomial sum-of-squares (SOS) Lyapunov function.
> **Step 3** Keep the system if such function exists, restart from step 1 otherwise.

This approach has several limitations. First, since most polynomial systems are not stable, and the computation of SOS Lyapunov function involves a complicated search [33], it is slow and limited to small systems of polynomials with small degree. Second, because not all stable polynomial systems have polynomial SOS Lyapunov functions [1], it can only generate a subset of stable polynomial systems.

Finally, SOS routines process the constraints in Equation (3) by solving semi-definite programming (SDP) problems. This guarantees that V is a sum-of-squares, hence we have $V(x) \geq 0$, but not necessarily $V(x) > 0$, for $x \neq 0$. As a result, these methods can only discover *barrier functions*. State-of-the-art methods circumvent this by introducing the stronger constraint $V(x) \geq \sum_{i=1}^{n} \varepsilon_i x_i^2$, with ε_i small [32]. V then has a unique minimum in $x = 0$, which makes it a Lyapunov function, but this further restricts the class of polynomial systems that the method can solve.

4.3 Datasets

We generate 2 backward and 2 forward datasets for training and evaluation purpose, and one smaller forward dataset for evaluation purposes (see Table 8 in Appendix B.6 for a list).

6

Backward datasets Our main backward set, **BPoly**, features 1 million non-degenerate polynomial systems S with integer coefficients, and 2 to 5 equations (in equal proportions). We also create **BNonPoly**, a dataset of 1 million non-degenerate non-polynomial systems with 2 to 5 equations. In this dataset, the coordinates of f are polynomials of general functions, e.g. trigonometric polynomials, or functions such as $3\cos(x_1) + 2x_1 e^{x_2}$. For such general systems, no method for discovering a Lyapunov function is known.

Forward datasets All 2 forward datasets are generated using a solver derived from the SumOf-Squares package in Python, and implementing techniques similar to those used in SOSTOOLS (see Appendix B.5). All systems in these datasets are non-zero integer polynomials with 2 to 3 equations, and integer polynomial Lyapunov functions – the only systems these methods can solve. We create **FLyap**, a dataset of 100,000 systems having a non-homogeneous polynomial as a Lyapunov function. We also have a dataset focusing on barrier functions (see the end of section 4.2): **FBarr** features 300,000 systems having a non-homogeneous polynomial as a barrier function. The small size of these datasets is due to the computational cost of SOS methods, and the difficulty of discovering Lyapunov or barrier functions.

To allow for comparison with SOSTOOL, the state-of-the-art method for discovering Lyapunov functions of polynomial systems, we also generated a test set of 1,500 polynomial systems with integer coefficients that SOSTOOLS can solve (**FSOSTOOLS**).

5   Results

Our models trained on different datasets achieve near perfect accuracy on held-out test sets, and very high performances on out-of-distribution test sets, especially when enriching the training set with a small number of forward examples. They greatly outperform state-of-the-art techniques and also allow to discover Lyapunov functions for new systems. These results are detailed below.

5.1   In and out-of-distribution accuracy

In this section, we present the performance of models trained on the 4 datasets. All models achieve high in-domain accuracy – when tested on held-out test sets from the datasets they were trained on (Table 2). On the forward datasets, barrier functions are predicted with more than 90% accuracy, and Lyapunov functions with more than 80%. On backward datasets, models trained on BPoly achieve close to 100% accuracy. We note that beam search, i.e. allowing several guesses at the solution, brings a significant increase in performance (7 to 10% with beam size 50, for the low-performing models). We use beam size 50 in all further experiments.

Backward datasets	Accuracy		Forward datasets	Accuracy	
	bs=1	bs=50		bs=1	bs=50
BPoly (polynomial)	99	100	FBarr (barrier)	93	98
BNonPoly (non-poly)	77	87	FLyap (Lyapunov)	81	88

Table 2: **In-domain accuracy of models**. Beam size (bs) 1 and 50.

The litmus test for models trained on generated data is their ability to generalize out-of-distribution (OOD). Table 3 presents evaluations of backward models on forward-generated sets (and the other way around). All backward models achieve high accuracy (73 to 75%) when tested on forward-generated random polynomial systems with a sum-of-squares Lyapunov functions (FLyap). The best performances are achieved by non-polynomial systems (BNonPoly), the most diverse training set. The lower accuracy of backward models on forward-generated sets of systems with barrier functions (FBarr) may be due to the fact that many barrier functions are not necessarily Lyapunov functions. On those test sets, backward models must cope with a different distribution and a (slightly) different task. Forward models, on the other hand, achieve low performance on backward test sets. This is possibly due to the small size of these training set.

Overall, these results seem to confirm that backward-trained models are not learning to invert their generative procedure. If it were the case, their performance on the forward test sets would be close to zero. They also display good OOD accuracy.

7

Backward datasets	FLyap	FBarr ‖	Forward datasets	BPoly
BPoly (polynomial)	**73**	35 ‖	FBarr (barrier)	15
BNonPoly (non-poly)	**75**	24 ‖	FLyap (Lyapunov)	10

Table 3: **Out-of-domain accuracy of models**. Beam size 50. Columns are the test sets.

5.2 Enriching training distributions for improved performance

To improve the OOD performance of backward models, we add to their training set a tiny number of forward-generated examples, as in [16]. Interestingly, this brings a significant increase in performance (Table 4). Adding 300 examples from FBarr to BPoly brings accuracy on FBarr from 35 to 89% (even though the proportion of forward examples in the training set is only 0.03%) and increases OOD accuracy on FLyap by more than 10 points. Adding examples from FLyap brings less improvement.

These results indicate that the OOD performance of models trained on backward-generated data can be greatly improved by adding to the training set a small number of examples (tens or hundreds) that we know how to solve. Here, the additional examples solve a weaker but related problem: discovering barrier functions. The small number of examples needed to boost performance makes this technique especially cost-effective.

Forward datasets	Examples added (1M in training set)	FLyap	FBarr
No mixture	0	73	35
FBarr	30	75	61
	300	83	89
	3,000	85	93
	30,000	89	95
FLyap	10	75	25
	100	82	29
	1,000	83	37
	10,000	86	38

Table 4: **Mixing backward data (BPoly) with a small number of forward examples**. Beam size 50.

5.3 Comparing with state-of-the-art baselines

To provide a baseline for our models, we developed `findlyap`, a Python counterpart to the MATLAB Lyapunov function finder from SOSTOOLS (see Appendix B.5). We also introduce FSOSTOOLS, a test set of 1,500 polynomial systems with integer coefficients that SOSTOOLS can solve. We also tested AI-based tools (see Appendix E for the full list of parameters sweeps we used for each of these methods), such as Fossil 2 [12], ANLC v2 [15] and LyzNet [25]. These methods achieve low accuracies on our test sets. This might be due to the fact that these tools are designed to solve a different problem: discovering local or semi-global Lyapunov function (and potentially finding a control function), while we target global Lyapunov functions.

	SOSTOOL	Existing AI methods			Models			
Test sets	`findlyap`	Fossil 2	ANLC	LyzNet	PolyMixture	FBarr	FLyap	BPoly
FSOSTOOLS	-	32	30	46	**84**	80	53	54
FBarr	-	12	18	28	**89**	-	28	35
FLyap	-	42	32	66	83	**93**	-	73
BPoly	15	10	6	24	**99**	15	10	-

Table 5: **Performance comparison on different test sets**. Beam size 50. PolyMixture is BPoly + 300 FBarr.

Table 5 compares `findlyap` and AI-based tools to our models on all available test sets. A model trained on BPoly complemented with 500 systems from FBarr (PolyMixture) achieves 84% on FSOS-TOOLS, confirming the high OOD accuracy of mixture models. On all generated test sets, PolyMixture achieves accuracies over 84% whereas `findlyap` achieves 15% on the backward generated test

8

set. This demonstrates that, on polynomial systems, transformers trained from backward-generated data achieve very strong results compared to the previous state of the art.

On average Transformer-based models are also much faster than SOS methods. When trying to solve a random polynomial system with 2 to 5 equations (as used in Section 5.4), `findlyap` takes an average of 935.2s (with a timeout of 2400s). For our models, inference and verification of one system takes 2.6s on average with greedy decoding, and 13.9s with beam size 50.

5.4 Into the wild - discovering new mathematics

Our ultimate goal is to discover new Lyapunov functions. To test our models' ability to do so, we generate three datasets of random systems: polynomials systems with 2 or 3 equations (**Poly3**), polynomial systems with 2 to 5 equations (**Poly5**), and non-polynomial systems with 2 or 3 equations (**NonPoly**). For each dataset, we generate 100,000 random systems and eliminate those that are trivially locally exponentially unstable in $x^* = 0$, because the Jacobian of the system has an eigenvalue with strictly positive real part [20]. We compare `findlyap` and AI based methods with two models trained on polynomial systems, FBarr, and PolyM(ixture) – a mixture of BPoly and 300 examples from FBarr– and one model trained on a mixture of BPoly, BNonPoly and 300 examples from FBarr (NonPolyM).

Table 6 presents the percentage of correct solutions found by our models. On the polynomial datasets, our best model (PolyM) discover Lyapunov functions for 11.8 and 10.1% of the (degree 3 and degree 5) systems, ten times more than `findlyap`. For non-polynomial systems, Lyapunov functions are found for 12.7% of examples. These results demonstrate that language model trained from generated datasets of systems and Lyapunov function can indeed discover yet unknown Lyapunov functions and perform at a much higher level that state-of-the-art SOS solvers.

Test set	Sample size	SOSTOOL `findlyap`	Fossil 2	ANLC	LyzNet	Forward FBarr	PolyM	NonPolyM
Poly3	65,215	1.1	0.9	0.6	4.3	11.7	**11.8**	11.2
Poly5	60,412	0.7	0.3	0.2	2.1	8.0	**10.1**	9.9
NonPoly	19,746	-	1.0	0.6	3.5	-	-	**12.7**

Table 6: **Discovering Lyapunov comparison for random systems**. Beam size 50. PolyM is BPoly + 300 FBarr. NonPolyM is BNonPoly + BPoly + 300 FBarr.

5.5 Expert iteration

Given the performance on our model in Table 6, we can use the newly solved problems to further fine-tune the model. Specifically, we create a sample of verified model predictions for polynomial systems, **FIntoTheWild**, we add it to the original training sample and we continue training the model.

We test different strategy to finetune the model and we report performance on forward benchmarks and "into the wild" in Table 7.

> *n1*: Add 20,600 samples from BPoly (20,000), FBarr (50), FLyap (50) and FIntoTheWild (500) to keep similar proportion used during pretraining
> *n2*: Add 2,000 samples from FLyap (1,000) and FIntoTheWild (1,000) to improve on both forward benchmark and in the wild
> *n3*: Add 50 samples from FIntoTheWild to show that this indeed helps
> *n4*: Add 1,000 samples from FIntoTheWild
> *n5*: Add 2,000 samples from FIntoTheWild
> *n6*: Add 5,000 samples from FIntoTheWild to see if there are benefits to add more samples

We also retrain a model (*n7*) from scratch using a mixture of BPoly (1M), FBarr (500), FLyap (500) and FIntoTheWild (2,000).

We notice that the addition of 1,000 verified predictions to our training set of 1 million improves performance on the "into to wild" test sets by about 15%, while not affecting the other test sets (*n4*). Adding more examples seems to be detrimental, as it decreases the performance on other benchmarks (*n5* and *n6*). We also notice that finetuning with mixed data from other distributions is not efficient (*n1* and *n2*) and a small contribution already help to get some improvements (result *n3*). Finally, it's not efficient to pretrain the model from scratch using data from FIntoTheWild (*n7*).

9

	Forward benchmark		Regenerated IntoTheWild	
Strategy	FBarr	FLyap	Poly3	Poly5
Baseline	93	84	11.7	9.6
n1	94	84	10.3	9.6
n2	90	85	12.2	11.3
n3	92	84	12.4	10.1
n4	92	84	**13.5**	**11.9**
n5	89	79	13.5	11.9
n6	85	72	13.5	11.9
n7	93	81	12.1	10.0

Table 7: **Expert iteration using IntoTheWild correct guesses**. The Poly3 and Poly5 test sets are regenerated, to prevent data contamination.

6 Discussion

We have shown that models can be trained from generated datasets to solve a long-standing open problem in mathematics: discovering the Lyapunov functions of stable dynamical systems. For random polynomial systems, our best models can discover Lyapunov functions in five times more cases than state-of-the-art methods. They can also discover Lyapunov functions of non-polynomial systems, for which no algorithm is yet known, and were able to re-discover a non-polynomial Lyapunov function of a polynomial systems discovered by [1] (Appendix F).

The backward generation method introduced in section 4.1 is the key innovation in this paper. The main problem with such approaches is their tendency to generate training sets with very specific distributions, which prevent models from generalizing to general instances of the problem. Our models can generalize out of their training distributions (Table 3), and we can improve their performance by adding to their training set a tiny number of systems that we know how to solve (Table 5).

While our models exceed the algorithmic state of the art, one might wonder **how they compare to human mathematicians**. To this effect, we proposed 75 problems from the FSOSTOOLS dataset (polynomial systems with 2 or 3 equations) as an examination for 25 first year Masters students in mathematics, following a course on the subject. Each student was given 3 systems chosen at random and had a total of 30 min. Their performance was 9.33%, significantly lower than our models (84%).

Our work has a number of limitations. Because there is no known way to tell whether a random system is stable, we lack a good benchmark on non-polynomial systems. Also, all the systems studied in this paper are relatively small, at most 5 equations for polynomial systems and 3 for non-polynomial. We believe that scaling to larger models should help tackle larger, and more complex, systems. Finally, this work could be extended to take into account the domain of definition of non-polynomial systems.

The broader implications of our work extend into two directions: the capability of transformers to reason, and the potential role of AI in scientific discovery. While large language models perform at human level on a broad set of tasks, they are still embarrassingly clumsy on many simple problems of logic and reasoning, to the point that it was suggested that planning and high level reasoning may be an inherent limitation of auto-regressive transformer architectures. Our results suggest that transformers can indeed be trained to discover solutions to a hard problem of symbolic mathematics that humans solve through reasoning, and that this is enabled by a careful selection of training examples, instead of a change of architecture. We do not claim that the Transformer is reasoning but it may instead solve the problem by a kind of "super-intuition" that stems from a deep understanding of a mathematical problem.

From a mathematical point of view, we propose a new, AI-based, procedure for finding Lyapunov functions, for a broader class of systems than were previously solvable using current mathematical theories. While this systematic procedure remains a black box, and the "thought process" of the transformer cannot be elucidated, the solutions are explicit and their mathematical correctness can be verified. This suggests that generative models can already be used to solve research-level problems in mathematics, by providing mathematicians with guesses of possible solutions. While a small minority of mathematicians is currently using deep-learning tools, we believe generative models have the potential to foster tremendous progress on a number of research subjects, and may eventually become a central component in the future landscape of mathematical practice.

10

Acknowledgements

This work was performed in part using HPC resources from GENCI–IDRIS (Grant 2023-AD011014527). The authors also acknowledge the Office of Advanced Research Computing (OARC) at Rutgers, The State University of New Jersey. The authors would also like to thank the Master students of the Mathematics and Computer Science Department of the Ecole des Ponts - IP Paris from the year 2023-2024 who attended the course Control of Dynamical Systems. The authors also wish to thank Guillaume Lample for fruitful discussion.

References

[1] Amir Ali Ahmadi, Miroslav Krstic, and Pablo A. Parrilo. A globally asymptotically stable polynomial vector field with no polynomial lyapunov function. In *2011 50th IEEE Conference on Decision and Control and European Control Conference*, pages 7579–7580, 2011.

[2] Amir Ali Ahmadi, Miroslav Krstic, and Pablo A. Parrilo. A globally asymptotically stable polynomial vector field with no polynomial lyapunov function. In *2011 50th IEEE Conference on Decision and Control and European Control Conference*, pages 7579–7580, 2011.

[3] Martin S. Andersen, Joachim Dahl, and Lieven Vandenberghe. Cvxopt. `https://cvxopt.org/`, 2023. Version 1.3.2; Free software package for convex optimization based on Python. Last updated August 9, 2023.

[4] Anton Bakhtin, Noam Brown, Emily Dinan, Gabriele Farina, Colin Flaherty, Daniel Fried, Andrew Goff, Jonathan Gray, Hengyuan Hu, Athul Paul Jacob, Mojtaba Komeili, Karthik Konath, Minae Kwon, Adam Lerer, Mike Lewis, Alexander H. Miller, Sasha Mitts, Adithya Renduchintala, Stephen Roller, Dirk Rowe, Weiyan Shi, Joe Spisak, Alexander Wei, David Wu, Hugh Zhang, and Markus Zijlstra. Human-level play in the game of <i>diplomacy</i> by combining language models with strategic reasoning. *Science*, 378(6624):1067–1074, 2022.

[5] Luca Biggio, Tommaso Bendinelli, Alexander Neitz, Aurelien Lucchi, and Giambattista Parascandolo. Neural symbolic regression that scales, 2021.

[6] Kevin Buzzard, Johan Commelin, and Patrick Massot. Formalising perfectoid spaces. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*. ACM, jan 2020.

[7] Ya-Chien Chang, Nima Roohi, and Sicun Gao. Neural Lyapunov Control. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[8] François Charton, Amaury Hayat, and Guillaume Lample. Learning advanced mathematical computations from examples. *arXiv preprint arXiv:2006.06462*, 2020.

[9] François Charton. Computing the roots of polynomials, 2022. `https://f-charton.github.io/polynomial-roots`.

[10] François Charton. Linear algebra with transformers, 2022.

[11] Jean-Michel Coron. *Control and nonlinearity*. American Mathematical Soc., 2007.

[12] Alec Edwards, Andrea Peruffo, and Alessandro Abate. Fossil 2.0: Formal certificate synthesis for the verification and control of dynamical models. In *Proceedings of the 27th ACM International Conference on Hybrid Systems: Computation and Control*, HSCC '24, New York, NY, USA, 2024. Association for Computing Machinery.

[13] Sicun Gao, Soonho Kong, and Edmund M. Clarke. dreal: An smt solver for nonlinear theories over the reals. In Maria Paola Bonacina, editor, *Automated Deduction – CADE-24*, pages 208–214, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[14] Peter Giesl. *Construction of global Lyapunov functions using radial basis functions*, volume 1904. Springer, 2007.

[15] Davide Grande, Andrea Peruffo, Enrico Anderlini, and Georgios Salavasidis. Augmented Neural Lyapunov Control. *IEEE Access*, 11:67979–67986, 2023.

11

[16] 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.

[17] Immanuel Kant. *Critique of pure reason*. 1787.

[18] Christopher M Kellett. Classical converse theorems in lyapunov's second method. *Discrete & Continuous Dynamical Systems-Series B*, 20(8), 2015.

[19] Hiroshi Kera, Yuki Ishihara, Yuta Kambe, Tristan Vaccon, and Kazuhiro Yokoyama. Learning to compute gr\" obner bases. *arXiv preprint arXiv:2311.12904*, 2023.

[20] Hassan K. Khalil. *Nonlinear systems*. Macmillan Publishing Company, New York, 1992.

[21] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[22] Guillaume Lample and François Charton. Deep learning for symbolic mathematics. *arXiv preprint arXiv:1912.01412*, 2019.

[23] Guillaume Lample, Marie-Anne Lachaux, Thibaut Lavril, Xavier Martinet, Amaury Hayat, Gabriel Ebner, Aurélien Rodriguez, and Timothée Lacroix. HyperTree Proof Search for Neural Theorem Proving. *Advances in neural information processing systems*, 2022.

[24] Jun Liu, Maxwell Fitzsimmons, Ruikun Zhou, and Yiming Meng. Formally verified physics-informed neural control lyapunov functions. *arXiv preprint arXiv:2409.20528*, 2024.

[25] Jun Liu, Yiming Meng, Maxwell Fitzsimmons, and Ruikun Zhou. Lyznet: A lightweight python tool for learning and verifying neural lyapunov functions and regions of attraction, 2024.

[26] Aleksandr Mikhaĭlovich Lyapunov. *The general problem of the stability of motion*. PhD thesis, Kharkov University, 1892.

[27] Jose Luis Massera. On liapounoff's conditions of stability. *Annals of Mathematics*, pages 705–721, 1949.

[28] Rodrigo Frassetto Nogueira, Zhiying Jiang, and Jimmy Lin. Investigating the limitations of the transformers with simple arithmetic tasks. *CoRR*, abs/2102.13019, 2021.

[29] KP Persidskii. On a theorem of liapunov. In *CR (Dokl.) Acad. Sci. URSS*, volume 14, pages 541–543, 1937.

[30] Stanislas Polu and Ilya Sutskever. Generative language modeling for automated theorem proving, 2020.

[31] Stephen Prajna, Ali Jadbabaie, and George J Pappas. A framework for worst-case and stochastic safety verification using barrier certificates. *IEEE Transactions on Automatic Control*, 52(8):1415–1428, 2007.

[32] Stephen Prajna, Antonis Papachristodoulou, and Pablo A Parrilo. Introducing sostools: A general purpose sum of squares programming solver. In *Proceedings of the 41st IEEE Conference on Decision and Control, 2002.*, volume 1, pages 741–746. IEEE, 2002.

[33] Stephen Prajna, Antonis Papachristodoulou, Peter Seiler, and Pablo A Parrilo. Sostools and its control applications. *Positive polynomials in control*, pages 273–292, 2005.

[34] Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024.

12

[35] Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. Code llama: Open foundation models for code, 2024.

[36] Guillaume Sagnol and Maximilian Stahlberg. PICOS: A Python interface to conic optimization solvers. *Journal of Open Source Software*, 7(70):3915, February 2022.

[37] Trieu H. Trinh, Yuhuai Wu, Quoc V. Le, He He, and Thang Luong. Solving olympiad geometry without human demonstrations. *Nature*, 625(7995):476–482, 2024.

[38] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[39] Adam Zsolt Wagner. Constructions in combinatorics via neural networks. *arXiv preprint arXiv:2104.14516*, 2021.

[40] Emily Wenger, Mingjie Chen, Francois Charton, and Kristin E Lauter. Salsa: Attacking lattice cryptography with transformers. *Advances in Neural Information Processing Systems*, 35:34981–34994, 2022.

[41] Xiangru Xu, Paulo Tabuada, Jessy W Grizzle, and Aaron D Ames. Robustness of control barrier functions for safety critical control. *IFAC-PapersOnLine*, 48(27):54–61, 2015.

[42] Gal Yehuda, Moshe Gabel, and Assaf Schuster. It's not what machines can learn, it's what we cannot teach. *arXiv preprint arXiv:2002.09398*, 2020.

[43] Chenyang Yuan. Sumofsquares.py, 2024. Software version 1.2.0; Available at https://github.com/yuanchenyang/SumOfSquares.py.

[44] Ruo-Shi Yuan, Yi-An Ma, Bo Yuan, and Ping Ao. Lyapunov function as potential function: A dynamical equivalence. *Chinese Physics B*, 23(1):010505, 2013.

[45] Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. Large language models are human-level prompt engineers, 2023.

13

Appendix

A Mathematical definitions

In this Appendix, we recall several mathematical definitions and theorems related to the Lyapunov function problem. We first introduce the notion of global asymptotic stability (GAS).

Definition A.1. We say that the (equilibrium $x^* = 0$ of the) system (1) is *globally asymptotically stable* if it is stable and for any $x_0 \in \mathbb{R}^n$ there exists a unique solution $x \in C^1([0, +\infty); \mathbb{R}^n)$ to (1) which satisfies in addition

$$\lim_{t \to +\infty} x(t) = 0. \tag{4}$$

This notion translates the fact that the equilibrium $x^* = 0$ is robust even to large perturbations. This notion is related to the existence of a Lyapunov function thanks, for instance, to LaSalle Invariance Principle:

Theorem A.2 (LaSalle Invariance Principle (global))**.** *Assume there exists a Lyapunov function for the system* (1) *and let S be the largest subset of $\{\nabla V(x) \cdot f(x) = 0\}$ that is invariant by the dynamics of* (1)*. If $S = \{0\}$, then the system* (1) *is globally asymptotically stable.*

Note that if $\nabla V(x) \cdot f(x) < 0$ for any $x \neq 0$ then necessarily $S = \{0\}$. Because finding a (global) Lyapunov function is a challenging mathematical problem, and still an open problem in general, weaker notions exists.

Definition A.3. The function $V \in C^1(\mathbb{R}^n, \mathbb{R}_+)$ is said to be a *semi-global Lyapunov function* for the system (1) if there exists $r > 0$ such that the following condition are satisfied

$$\begin{aligned} V(0) = 0, \quad V(x) > 0, \\ \nabla V(x) \cdot f(x) \leq 0 \text{ for } \|x\| \leq r. \end{aligned} \tag{5}$$

Finding a semi-global Lyapunov function is usually easier than finding a global Lyapunov function. A semi-global Lyapunov function is enough to show that the equilibrium $x^* = 0$ is robust to small perturbations which, for several engineering applications, is enough. More specifically,

Definition A.4. We say that the (equilibrium $x^* = 0$ of the) system (1) is *locally asymptotically stable* if it is stable and if there exists $r > 0$ such that for any $\|x_0\| \leq r$ there exists a unique solution $x \in C^1([0, +\infty); \mathbb{R}^n)$ to (1) which satisfies in addition

$$\lim_{t \to +\infty} x(t) = 0. \tag{6}$$

Similarly to global Lyapunov function, the existence of a semi-global Lyapunov function is useful to ensure local asymptotic stability

Theorem A.5 (LaSalle Invariance Principle (local))**.** *Assume there exists a semi-global Lyapunov function V, and let S be the largest subset of $\{\nabla V(x) \cdot f(x) = 0\}$ invariant by the dynamics of* (1)*. If $S = \{0\}$ then the system* (1) *is locally asymptotically stable.*

B Generation procedure

B.1 Function generation

To generate random functions we sample random trees with unary and binary internal nodes, and then randomly select operators for these nodes, and variables and integers for leaves (as in [22, 8]). Our binary operators are the four operations and the power function. Unary operators are $\exp, \log, \text{sqrt}, \sin, \cos, \tan$.

To generate polynomials, we randomly sample a given number of monomials, with integer or real coefficients. The number of monomials, range of the coefficients, and the powers and number of terms of each monomial, are randomly selected between bounds, provided as hyperparameters.

14

B.2 Backward generation

We build globally stable systems by first generating a Lyapunov function V at random, and then building a dynamic system which has V as a Lyapunov function. The procedure is:

Step 1a: We generate V as $V = V_{\text{cross}} + V_{\text{proper}}$ where V_{proper} belongs to a given class of positive definite function and V_{cross} belongs to a larger class, but of non-negative functions only with $V_{\text{cross}}(0) = 0$. More specifically, we generate

$$V_{\text{cross}}(x) = \sum_{i=1}^{m} p_i^2(x), \tag{7}$$

with m a random integer, and p_i random functions verifying $p_i(0) = 0$. The nature of functions p_i depends on the systems we want to generate (polynomial or not). Clearly $V_{\text{cross}}(x) \geq 0$ and $V_{\text{cross}}(0) = 0$. We similarly generate

$$V_{\text{proper}}(x) = \sum_{i=1}^{n} \alpha_{i,j} x_i^{\beta_i} x_j^{\beta_j}, \tag{8}$$

with n a random integer, β_i random positive integers and $A = (\alpha_{i,j})_{(i,j) \in \{1,\dots,n\}^2}$ a random positive definite matrix, with a given probability of being diagonal. As a consequence, V_{proper} is strictly minimal in $x = 0$. When generating barrier functions, we can optionally set $V_{\text{proper}} = 0$.

Step 1b: In this step, we increase the class of functions that can be sampled for V_{cross} and V_{proper} by several transformations:

1. **Composition of V_{proper}** with probability $p_{1,c}$, replace
$$V_{\text{proper}}(x) \leftarrow I(V_{\text{proper}}(x)) \tag{9}$$
with I selected at random from a pre-defined set of *increasing-functions* (Appendix B.4),

2. **Product V_{proper}** with probability $p_{1,m}$, replace
$$V_{\text{proper}}(x) \leftarrow (V_{\text{proper}}(x) - V_{\text{proper}}(0))g(h(x)), \tag{10}$$
with g selected at random from a pre-defined set of *positive-functions* (Appendix B.4), h a sub-expression of V_{proper}, i.e. $h(x) = \sum_{i=1}^{q} \alpha_{\sigma(i),\sigma(j)} x_{\sigma(i)}^{\beta_{\sigma(i)}} x_{\sigma(j)}^{\beta_{\sigma(j)}}$, for $q \leq n$ and σ a permutation of $\{1, \dots, n\}$.

3. **Composition of V_{cross}:** for every $i \in \{1, \dots, m\}$, with probability p_2, replace
$$p_i^2(x) \leftarrow b_i(\xi_i + p_i(x)), \tag{11}$$
with b_i a real function that is bounded from below with a minimum (not necessarily unique) in ξ_i and chosen at random from a pre-defined set of *bounded-functions* (Appendix B.4). Recall that p_i are the functions appearing in V_{cross}.

Step 1c: Gathering the functions V_{proper} and V_{cross} together, we define the Lyapunov function (candidate) $V(x) = V_{\text{cross}}(x) + V_{\text{proper}}(x)$. Overall, we have

$$V(x) = \left[I\left(\sum_{i=1}^{n} \alpha_{i,j} x_i^{\beta_i} x_j^{\beta_j} \right) - I(0) \right] g\left(\sum_{i=1}^{q} \alpha_{\sigma(i),\sigma(j)} x_{\sigma(i)}^{\beta_{\sigma(i)}} x_{\sigma(j)}^{\beta_{\sigma(j)}} \right) + \sum_{i=1}^{m} b_k(\xi_k + p_k(x)),$$

where I is the identity with probability $1 - p_{1,c}$, g is the constant function 1 with probability $1 - p_{1,m}$ and $b_k(x) = x^2$ with probability $1 - p_2$. Such a Lyapunov function satisfies

$$V(x) > V(0), \quad \forall x \in \mathbb{R}^n \setminus \{0\}. \tag{12}$$

Indeed,

$$V(0) = \sum_{i=1}^{m} b_k(\xi_k)$$

and $V(x) > b_k(\xi_k)$ for any $x \in \mathbb{R}^n \setminus \{0\}$, since g is a positive function, I is increasing and $(\alpha_{i,j})_{i,j \in \{1,\dots,n\}}$ is positive definite.

15

Step 2: In this step we create the random vectors orthogonal to ∇V that will be useful in the generation of the system f (see Section 4.1). Taking advantage of the form of the condition (3), for any $x \in \mathbb{R}$, denote

$$\mathcal{H}_x = \{z \in \mathbb{R}^n \mid z \cdot \nabla V(x) = 0\}$$

the hyperplane orthogonal to $\nabla V(x)$. Then, for a random $p \in \{1, ..., n\}$, generate p random real-valued functions $(g_i)_{i \in \{1,...,p\}}$, and p vectors $\{e^i\}_{i \in \{1,...,p\}}$ from this hyperplane as follows:

$$e^i_j = \begin{cases} A_{\tau_2(i)} & \text{if } j = \tau_1(i) \\ -A_{\tau_1(i)} & \text{if } j = \tau_2(i) \\ 0 & \text{otherwise,} \end{cases} \tag{13}$$

where $A = \nabla V(x)$ and A_j refers to the $j-th$ component of the vector and τ_1 and τ_2 random functions from $\mathbb{N} \setminus \{0\}$ into $\{1, ..., n\}$, such that $\tau_1(i) \neq \tau_2(i)$. This implies that for any $i \in \{1, ..., n\}$

$$\nabla V(x) \cdot e^i = (\nabla V(x))_{\tau_1(i)} (\nabla V(x))_{\tau_2(i)} - (\nabla V(x))_{\tau_2(i)} (\nabla V(x))_{\tau_1(i)} = 0. \tag{14}$$

Note that, so long $\nabla V(x) \neq 0$, one can use this process to construct a generative family of $\mathcal{H}_x$, and the e^i span the whole $\mathcal{H}_x$. If $\nabla V(x) = 0$ then $\mathcal{H}_x = \mathbb{R}^n$.

Step 3: Generate at random k_1 real-valued functions $(h_i)_{i \in \{1,...,k_1\}}$, where $1 \leq k_1 \leq n$ is chosen at random. Set $h_i = 0$ for $k_1 < i \leq n$.

Step 4: Build the system

$$f(x) = -\left(h^2_{\pi(i)}(x)(\nabla V(x))_i\right)_{i \in \{1,...,n\}} + \sum_{i=1}^{p} g_i(x) e^i(x), \tag{15}$$

with π a random permutation of $\{1, ..., n\}$.

Overall, the function f satisfies

$$\nabla V(x) \cdot f(x) = -\left(\sum_{i=1}^{n} h^2_{\pi(i)}(x)(\nabla V(x))^2_i\right) \leq 0, \tag{16}$$

hence V is a Lyapunov function of the system $\dot{x}(t) = f(x(t))$.

Step 5: Expand and simplify the equations of f (using Sympy), in order to eliminate obvious patterns due to the generation steps (that the model could recognize and leverage), eliminate duplicated systems in the training set, and limit the length of training sequences. All polynomial systems are expanded into normal form.

B.3 Backward generation modes

Polynomial generation: we generate polynomial systems with sum-of-square Lyapunov functions to allow for easy comparison with existing methods such as SOSTOOLS [32, 33]. In this case, all P_i are polynomials with no zero-order term and $p_{1,c} = p_{1,m} = p_2 = 0$. Also, f_i and g_i are polynomials (Appendix B.1). We generate f_i with a degree lower or equal to half the maximal degree of g_i and a maximal value of coefficients of the order of the square root of the maximal value of g_i. Since the f_i are squared in the final system, this allows f_i^2 and g_i to have the same order, and prevents the transformer from inferring unwanted additional information by looking at the higher degree monomial.

Generic generation: P_i is generated as $P_i(x) = Q_i(x) - Q_i(0)$, where $Q_i(x)$ is a random function generated as per Appendix B.1 and f_i and g_i are also generated as per Appendix B.1. Optionally the functions can be generated as polynomials of non-polynomial operators taken from a pre-defined set of *operators*.

Other generation modes: we have other generation modes corresponding to interesting particular cases: gradient flow systems, systems where the 2-norm (resp. a weighted 2-norm) is a Lyapunov function, etc.

16

B.4 Generation design parameters

Our generator allows us to generate generic stable systems and yet to have a large control on the distribution. For polynomials, for instance, we have a control on the maximal and average degree, number of monomials, power and number of variables of the monomials, coefficients, etc. We can also specify whether the coefficients are integers, floats, with which precision. Overall we have a total of 36 generation hyper-parameters that influence the distribution of the synthetic data created. The main generation design parameters are:

- int_base: encoding base for integers
- max_int: Maximum integer value
- precision: Float numbers precision
- prob_int: Probability of sampling integers vs variables (for non-polynomial expressions)
- min_dim: minimal number of equations in the system
- max_dim: maximal number of equations
- max_degree: maximal degree of polynomial terms in a Lyapunov function
- n_terms: maximal number of terms in polynomials for the Lyapunov function
- nb_ops_proper: maximal number of operators in V_{proper} (non polynomial generation)
- nb_ops_lyap: maximal number of operators in V_{proper} (non polynomial generation)
- operators_lyap: list of operators to be considered (non polynomial generation)
- polynomial_V: if true generated expressions are polynomials of (potentially non-polynomial) operators
- pure_polynomial: generate polynomial systems only
- cross_term: $V_{cross} = 0$ if False.
- max_nb_cross_term: bound on m in V_{cross}
- proba_diagonal: with this probability, the positive definite form of V_{proper} is imposed to be diagonal
- only_2_norm: if True, the Lyapunov function is the 2-norm.
- strict: if True, generates a strict Lyapunov function (i.e. $\nabla V \cdot f < 0$)
- proper: if set to false, $V_{proper} = 0$ and V is only a barrier function.
- float_resolution_poly: float resolution of the polynomials generated by generate_bounded_polynomial.
- generate_gradient_flow: When set to True, the backward generation only generates gradient flows systems.
- gen_weight: exponential weight which bias the sampling of k_1 and p, the number of components of non-zero h_i and g_i.
- max_order_pure_poly: maximal polynomial order of h_i
- max_n_term_fwd: maximal number of terms in each equations in the fwd generation
- SOS_checker: if True, uses a SOS verifier to evaluate the candidate Lyapunov function (if False uses the verifier based on `shgo`)
- SMT_checker: if True, uses an SMT verifier to evaluate the candidate Lyapunov function (if False uses the verifier based on `shgo`)
- multigen: number of different system generated per Lyapunov function.
- increasing_func: the set of increasing functions used in the generation (see Step 1b). Default is $\{\exp, \ln(1 + x^2), \sqrt{1 + x}\}$.
- positive_func: the set of positive functions used in the generation (see Step 1b). Default is $\{\exp, 1 + \cos(x), 1 + \sin(x)\}$.
- bounded_func: the set of bounded functions used in the generation (see Step 1b). Default is $\{\cos, \sin\}$.

B.5 Forward SOS solver

SOSTOOLS is one of the most famous toolbox for sum-of-square optimization, in particular for finding SOS Lyapunov functions [32, 33]. It is natively available in MATLAB and relies on an underlying SDP solver that can be chosen. In Python an analogous toolbox is the package SumOf-Squares [43] which relies on the same principle, however does not have specific functionalities for Lyapunov functions. As a consequence we implemented these functionalities in our codebase based on the MATLAB implementations in SOSTOOLS. We implemented a function `SOS_checker`, which takes in input a system of equations in sympy and a candidate Lyapunov function and checks SOS conditions on $V(x)$ and $-\nabla V(x) \cdot f(x)$, and a function `findlyap`, analogous to the `findlyap`

17

function in SOSTOOLS, which takes a system of equations in sympy and either returns a function satisfying SOS conditions on $V(x)$ and $-\nabla V(x) \cdot f(x)$, returns false if no such function exists, or returns none if it fails to provide an answer. SumOfSquares relies itself on picos [36] and we use the default solver cvxopt [3].

B.6 List of datasets

Dataset	Description	Size (000)	Resources (CPU.hours)
BPoly	Backward polynomial systems, non-zero	1,000	210
BNonPoly	Backward non-polynomial systems, non-zero	1,000	220
FBarr	Forward, non-homogeneous polynomial barrier functions	300	9,670
FLyap	Forward, homogeneous polynomial Lyapunov functions	100	4,620
FSOSTOOLS	Forward, SOSTOOLS solved systems	1.5	

Table 8: **Datasets generated.** Backward systems are degree 2 to 5, forward systems degree 2 to 3. All forward systems are polynomial.

C Additional results

C.1 Impact of multigeneration

In the backward generation procedure, after sampling one random V, it is possible to generate any number of different systems f_i such that V is the Lyapunov function for each of the systems f_i. We call the maximal number of system generated per Lyapunov function the `multigen` parameter. The actual number of systems generated per Lyapunov function is chosen at random for each Lyapunov function between 1 and `multigen`. In Section 5 we reported results using `multigen` equal to 5. Here we report the in-domain and out-of-domain performance of the models trained on backward BPoly datasets of size 1 million varying the parameter `multigen`.

Multigen	In-domain BPoly	OOD FLyap
1	95	58
5	100	73
10	100	75
25	100	76
50	100	70
100	100	68

Table 9: **In-domain and out-of-domain accuracy of models**. Beam size 50.

Table 9 shows that generating a moderate amount of different systems with the same Lyapunov function actually improves the model capability to generalize out-of-domain. This suggests that the model is learning, at least partially, to separate the parts of the system which contribute to the Lyapunov function. Above a certain multigen threshold, model performances start to decline. This may be due to the low diversity present in the dataset, i.e. the limited number of different Lyapunov functions the model is trained on (the total number of systems in the training set remains constant so the total number of Lyapunov function decreases with the value of the parameter `multigen`).

C.2 Performance of smaller transformer models

In Section 5 we report results using a transformer with 8 encoder and decoder layers, 10 attention heads and an embedding dimension of 640. We also trained smaller models with 6 encoder and decoder layers, 8 attention heads and an embedding dimension of 512. Tables 10, 11 report the main results. Results are in line with what we showed in section 5

18

Backward datasets	In-domain	OOD FLyap	Forward datasets	In-domain	OOD BPoly
BPoly (polynomial)	100	70	FBarr (barrier)	97	13
BNonPoly (non-poly)	85	71	FLyap (Lyapunov)	86	11

Table 10: **In-domain and out-of-domain accuracy of models**. Beam size 50.

Forward datasets	Mixing proportion	FBarr	FLyap	Into the wild Poly3	Poly5
No mixture	0%	31	70	2.3	1.6
FBarr	0.01%	60	72	2.5	1.7
	0.1%	93	72	9.2	6.4
FLyap	0.01%	29	73	2.8	1.6
	0.1%	19	76	2.9	1.7

Table 11: **Performance of mixing backward data (BPoly) with a small number of forward examples on forward benchmark and "into the wild"**. Beam size 50.

D Comparison of SOS, SMT and `shgo`

We compare our model performance when we employ them to discover new Lyapunov function. We report performances with dReal SMT and SOS verifiers for Poly and dReal SMT and `shgo` for NonPoly distributions, respectively. Table 12 shows that SMT results are slightly lower, because of timeouts (which we report in Table 13, but comparable. Note that the performances on polynomial systems were already theoretically guaranteed thanks to the former SOS verifier.

Model (by training distribution)	FBarr	BPolyMixture	NonPolyMixture
Poly3	10.5/11.7	11.1/11.8	10.6/11.2
Poly5	6.5/8.0	8.7/10.1	8.4/9.9
NonPoly			8.3/12.7

Table 12: **Results of SMT** with SOS and `shgo` verifiers for Poly and NonPoly systems, respectively.

Test sets Timeout	Into the wild 10 minutes	120 minutes
Correct Lyap function	87.3	92.2
SMT Timeouts	10.8	5.8
Incorrect Lyap function	1.9	2.0

Table 13: **SMT timeout and error rates.** Most SMT failures are due to timeout.

E AI method sweep

To report the AI-based tools results on the seven benchmarks (BPoly, BNonPoly, FLyap, FBarr, Poly3, Poly5, NonPoly) we did a hyperparameter sweep. To get the best hyperparameter setting, we sweep on FLyap and then fix these hyperparameters for the different datasets. In bold we show the chosen parameters, selected to maximize the correctness on FLyap, subject to the 20 minutes timeout.

Lyznet [25]

- lr = $[3 \cdot 10^{-5}, \mathbf{10^{-4}}, 3 \cdot 10^{-4}]$
- points = [**100,000**, 300,000, 1,000,000]
- layer width = [(2,20), **(3,6)**, (6,2)]
- epoch = [1, 5, **25**]
- net type = [None, **Poly**]

19

Fossil 2.0 [12]

- iters = [10, **50**, 250]
- activations = [(**x²**), (x^2,x^2), (sigmoid), (sigmoid, sigmoid), (poly$_4$), (poly$_4$, poly$_4$)]
- hidden neurons = [6, **10**, 20]
- data = [500, **1000**, 2000]
- lr = [0.01, 0.03, **0.1**]

ANLC v2 [15]

- iters = [10, **50**, 250]
- activations = [(**x²**, **x**, **x**), (x^2, x^2, x), (x^2, x^2, x, x), (x^2, x^2, x^2, x)]
- hidden neurons = [6, **10**, 20]
- max data = [500, **1000**, 2000]
- lr = [**0.01**, 0.03, 0.1]

F   Some examples

To understand the model performance and compare against the SOSTOOL performance, we manually inspect some systems with 2 or 3 equations where the following conditions hold: (1) the Jacobian of the system has the maximum eigenvalue with real part equal to 0 (i.e. tools like the spectral mapping theorem cannot decide on the stability), (2) no weighted 2-norm functions can be a Lyapunov function, (3) `findlyap` times out after 4 hours. We show some examples below.

F.1   A polynomial system with non polynomial solution

System

$$\begin{cases} \dot{x}_0 & = -x_0 + x_0 x_1 \\ \dot{x}_1 & = -x_1 \end{cases}$$

It's known that there is no polynomial Lyapunov function for this system [2]. Our poly models and `findlyap` failed, as expected. Nonetheless, one of our non-poly models with beam search of beam size 100 proposed $V(x) = \ln(1 + 5x_0^2) + x_1^2$ similar to the one that was recently found in [2].

It's clear that $V(0) = 0$ and $V(x) > 0$ for all $x \neq 0$. Also

$$
\begin{aligned}
V(x) \cdot f(x) &= \frac{-10x_0^2 + 10x_0^2 x_1 - 2x_1^2(1 + 5x_0^2)}{1 + 5x_0^2} \\
&= \frac{-5x_0^2 - 5x_0^2 x_1^2 - 5(x_0 - x_0 x_1)^2 - 2x_1^2}{1 + 5x_0^2} \leq 0
\end{aligned}
\tag{17}
$$

as desired.

F.2   A system that has no diagonal Lyapunov function

System

$$\begin{cases} \dot{x}_0 & = 2x_1^2 \\ \dot{x}_1 & = -10x_1 \end{cases}$$

Model inference: Our model recovers $V(x) = 10x_0^2 + 2x_0 x_1^2 + 3x_1^4 + 6x_1^2$.

Clearly $V(0) = 0$ and $V(x) = 9(x_0)^2 + (x_0 + x_1^2)^2 + 2(x_1^2)^2 + 6x_1^2 > 0$ for all $x \neq 0$. Also $\nabla V(x) \cdot f(x) = -x_1^2(116x_1^2 + 120) \leq 0$.

Non existence of a Diagonal Lyapunov function: Suppose for the sake of contradiction that there exists a function V_1 which satisfies 3 and can be expressed as

$$V_1(x) = \sum_{i=1}^{n} a_i x_0^i + \sum_{j=1}^{m} b_j x_1^j.$$

20

Clearly $V_1(0) = 0$. Given that $V_1(x_0, 0) > 0$ for $x_0 \neq 0$, it follows that n is even and $a_n > 0$. Also we know that $\nabla V_1(x) \cdot f(x) = 2 \sum_{i=1}^{n} i a_i x_0^{i-1} x_1^2 - 10 \sum_{j=1}^{m} j b_j x_1^j \leq 0$ for all choices of (x_0, x_1). If we let $x_1 = 1$ we obtain $\nabla V_1(x) \cdot f(x) = 2 \sum_{i=1}^{n} i a_i x_0^{i-1} - 10 \sum_{j=1}^{m} j b_j$. This expression can be seen as a polynomial $g(x_0)$ with real coefficients and odd degree $n - 1$. The leading coefficient, $2n a_n$, is positive because $a_n > 0$ and $n \geq 1$. This means that $\lim_{x_0 \to +\infty} g(x_0) = +\infty$, meaning that there exists an x_0 such that $g(x_0) > 0$. This contradicts 3.

F.3 A system with 3 equations and a higher degree

System

$$
\begin{cases}
\dot{x}_0 & = -7x_0^5 - 4x_0^3 x_1^2 - 5x_0^3 \\
\dot{x}_1 & = 7x_0^4 - 3x_1 - 2x_2 \\
\dot{x}_2 & = -8x_0^2 - 9x_2
\end{cases}
$$

Model inference: Our model recovers different solutions. Here we show two of them

$$
V_1(x) = 4x_0^4 + 10x_0^2 x_1^2 + 2x_0^2 x_1 + 10x_0^2 x_2^2 - 4x_0^2 x_2 + 20x_0^2 + 10x_1^2 x_2^2 + 4x_1^2 - 2x_1 x_2 + 8x_2^4 + 4x_2^2,
$$
$$
V_2(x) = 2x_0^4 + 2x_0^2 x_1^2 + 3x_0^2 + 2x_1^2 + x_2^2.
$$

We checked with SumOfSquares that $V_1 > 0$, $V_2 > 0$, $\nabla V_1 \cdot f \leq 0$ and $\nabla V_2 \cdot f \leq 0$.

F.4 Other examples

System	Lyapunov function
$\begin{cases} \dot{x}_0 = -5x_0^3 - 2x_0 x_1^2 \\ \dot{x}_1 = -9x_0^4 + 3x_0^3 x_1 - 4x_1^3 \end{cases}$	$V(x) = 6x_0^6 + 7x_0^4 + x_0^3 + 10x_0^2 + 8x_1^2$
$\begin{cases} \dot{x}_0 = -x_0^5 - 4x_0^3 - 9x_0 x_1^4 + 3x_0 x_1^3 \\ \dot{x}_1 = -3x_0^4 x_1^2 - 10x_0^3 x_1 + 3x_0 x_1^2 - 7x_1^3 \end{cases}$	$V(x) = x_0^4 + 9x_0^2 + 3x_1^2$
$\begin{cases} \dot{x}_0 = -3x_0^3 + 3x_0 x_2 - 9x_0 \\ \dot{x}_1 = -x_0^3 - 5x_1 + 5x_2^2 \\ \dot{x}_2 = -9x_2^3 \end{cases}$	$V(x) = x_0^4 + 7x_0^2 x_2^2 + 3x_0^2 + 4x_0 x_2^2 + 3x_1^2 + 2x_2^4 + 10x_2^2$
$\begin{cases} \dot{x}_0 = -8x_0 x_1^2 - 10x_1^4 \\ \dot{x}_1 = -8x_1^3 + 3x_1^2 - 8x_1 \\ \dot{x}_2 = -x_2 \end{cases}$	$V(x) = 4x_0^2 - 2x_0 x_1^2 + 6x_1^4 + 4x_1^2 + x_2^2$

Table 14: **Some additional examples generated from our models**.

21

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The claims in the abstract and introduction summarize the results of section "Main results".

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Yes, in the discussion.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory Assumptions and Proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

22

Justification: This paper is mostly experimental, the only theoretical results is the stability of specific systems and we do verify the assumptions and state the theorem we use about Lyapunov functions.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental Result Reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: Yes, we detail the experiments, the parameters and the data generator. The code will eventually be provided.

Guidelines:

- The answer NA means that the paper does not include experiments.
- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

23

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [No]

Justification: The code and datasets are currently pending approval from some of the authors' institution for being open-sourced. They should be by the conference dates.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental Setting/Details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyper-parameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: We specify the main training and test details.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment Statistical Significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: We provide the number of elements in each sets, as well as the accuracies in percentage. Having error bars is not adapted when dealing with the infinite dimensional spaces of functions involved.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.

24

- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments Compute Resources**

Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

Answer: [Yes]

Justification: We provide the type of compute workers, the type of GPU, the memory, and the GPU and CPU time needed for training the models and generating the datasets.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code Of Ethics**

Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics https://neurips.cc/public/EthicsGuidelines?

Answer: [Yes]

Justification: This paper is about training neural networks to find solutions to mathematical problems. As such, it's societal aspects (safety, security, discrimination, surveillance, deception and harassment, environment, human rights, Bias and fairness) are either irrelevant or respect the NeurIPS Code of Ethics. The only data coming from human sources are answers to mathematics exercises from 25 students as part of one of their course. All students were all offered the choice of allowing or not the use of these data for the purpose of research, the answers of students who agreed were then anonymized and aggregated.

Guidelines:

- The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
- If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader Impacts**

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

25

Answer: [NA]

Justification: While there is no direct societal impact, there is an impact on the research in mathematics and science in general and we mention it.

Guidelines:

- The answer NA means that there is no societal impact of the work performed.
- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

Answer: [NA]

Justification: The models are small language models trained to solve a specific mathematical question.

Guidelines:

- The answer NA means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: With the exception of common Python's packages for which we only give the name, we cite the specific packages we are relying on, when relevant. No data or model from another source is used.

Guidelines:

26

- The answer NA means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New Assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [NA]

Justification: No new assets are released yet.

Guidelines:

- The answer NA means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and Research with Human Subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [No]

Justification: The data coming from human sources are only an element of the discussion and not the main point of the paper. The framework and instructions are given as well in the discussion.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional Review Board (IRB) Approvals or Equivalent for Research with Human Subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

27

Answer: [Yes]

Justification: The only data coming from human sources are answers to mathematics exercises from 25 students as part of one of their course. All students were all offered the choice of allowing or not the use of these data for the purpose of research, the answers of student who agreed were then anonymized and aggregated. This was done in coordination with the administration of the mathematics department.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

28