Using ML to guide and inspire rigorous computational number theory

David Lowry-Duda
Visiting Scholar, ICERM and Brown University
Principal ML Research Engineer, Institutional Data Initiative

February 2026

VaNTAGe Math Seminar

This talk touches on work with many collaborators, including

- Joanna Bieri, Giorgi Butbaia, Edgar Costa, Aly Deines, Kyu-Hwan Lee, David Lowry-Duda, Tom Oliver, Yidi Qi, and Tamara Veenstra (ML and Maass forms)
- Andy Booker, Andrei Seymour-Howell, and Min Lee (Rigorously computing Maass forms)
- Many people at the Harvard CMSA (ML and Möbius)

The previous talks have presented some points of view on how ML is being applied to number theory. We've heard about theorem proving software (bottom-up) development, meta-mathematical assistants, and ML as experiment.

Today: I want to think about how we can try to use *fundamentally heuristic* ML to learn something about *rigorous* number theory, possibly even some research direction.

1. The Möbius function $\mu(\cdot)$
2. Maass forms $f_\lambda(z)$

Warmup: Computing the Möbius function

The Möbius function $\mu(n)$ is defined by

$$\mu(n) = \begin{cases} 0 & \text{if } p^2 \mid n \text{ for some } p \\ (-1)^d & \text{if } n \text{ is a product of } d \text{ distinct primes.} \end{cases}$$

For example, $\mu(2) = \mu(3) = \mu(5) = -1$, and $\mu(6) = \mu(10) = \mu(15) = 1$, and $\mu(4) = \mu(9) = \mu(12) = 0$. The Möbius function is fundamental in number theory and appears frequently.

For example, the prime number theorem is "equivalent" to the fact that

$$\sum_{n \leq X} \mu(n) = o(X).$$

The function $\mu(\cdot)$ encodes data about prime numbers and prime factorization.

Despite[1] being fundamental, we are very bad at understanding $\mu(n)$.

Question: How do you compute $\mu(n)$?

Morally, the best we know is to first factor n and count the factors. Up to small modification, this is also the best way to determine whether n is squarefree (i.e. to compute $\mu^2(n)$).

[1] *because?*

Suppose we don't think too hard and ask:

Can ML algorithms learn to approximate $\mu(n)$?

For example, make a deep neural network and train on input-output pairs $(n, \mu(n))$ (or almost equally interesting, $(n, \mu^2(n))$ pairs). What happens?

First, let's ask what the baseline is. About $6/\pi^2 \approx 60\%$ of integers are squarefree, and squarefree integers have approximately equal probabilities of having $\mu(n) = 1$ or $\mu(n) = -1$. The naive approach would be to guess the most common class: $\mu(n) = 0$ (true 40% of the time) or $\mu^2(n) = 1$ (true 60% of the time).

Baseline: $\mu(n)$ correct about 40% of the time, $\mu^2(n)$ correct about 60% of the time.

The neural network quickly learns to predict $\mu(n)$ with over 60% accuracy and to predict $\mu^2(n)$ with over 90% accuracy. With more training, the network slowly learns to predict $\mu(n)$ with nearly 70% accuracy and $\mu^2(n)$ with nearly 100% accuracy.

Should we be amazed?

No.

To paraphrase Jordan Ellenberg, the machine is learning an algorithm with nearly 100% accuracy and nearly 0% understanding.

What's happening is that the network quickly learns to detect if n is divisible by 4, and then by 9, and eventually by 25, and so on. Then there is the important fact that *most numbers that are divisible by a square are divisible by a* ***small*** *square*.

> *Observation*
>
> *Learning to predict a function with high probability is, on its own, uninteresting for proof-based mathematics.*
>
> *However, successful prediction also means that there is some information contained in the input features. Maybe that's a good place to look.*

In this example, we find that "testing divisibility by small squares" is a good predictor for divisibility by any squares. That's not surprising or interesting.

We could stop there, but let's double down.

Question

Can machines learn something about $\mu(n)$ and $\mu^2(n)$ beyond testing divisibility by squares of small prime numbers?

Smaller Question

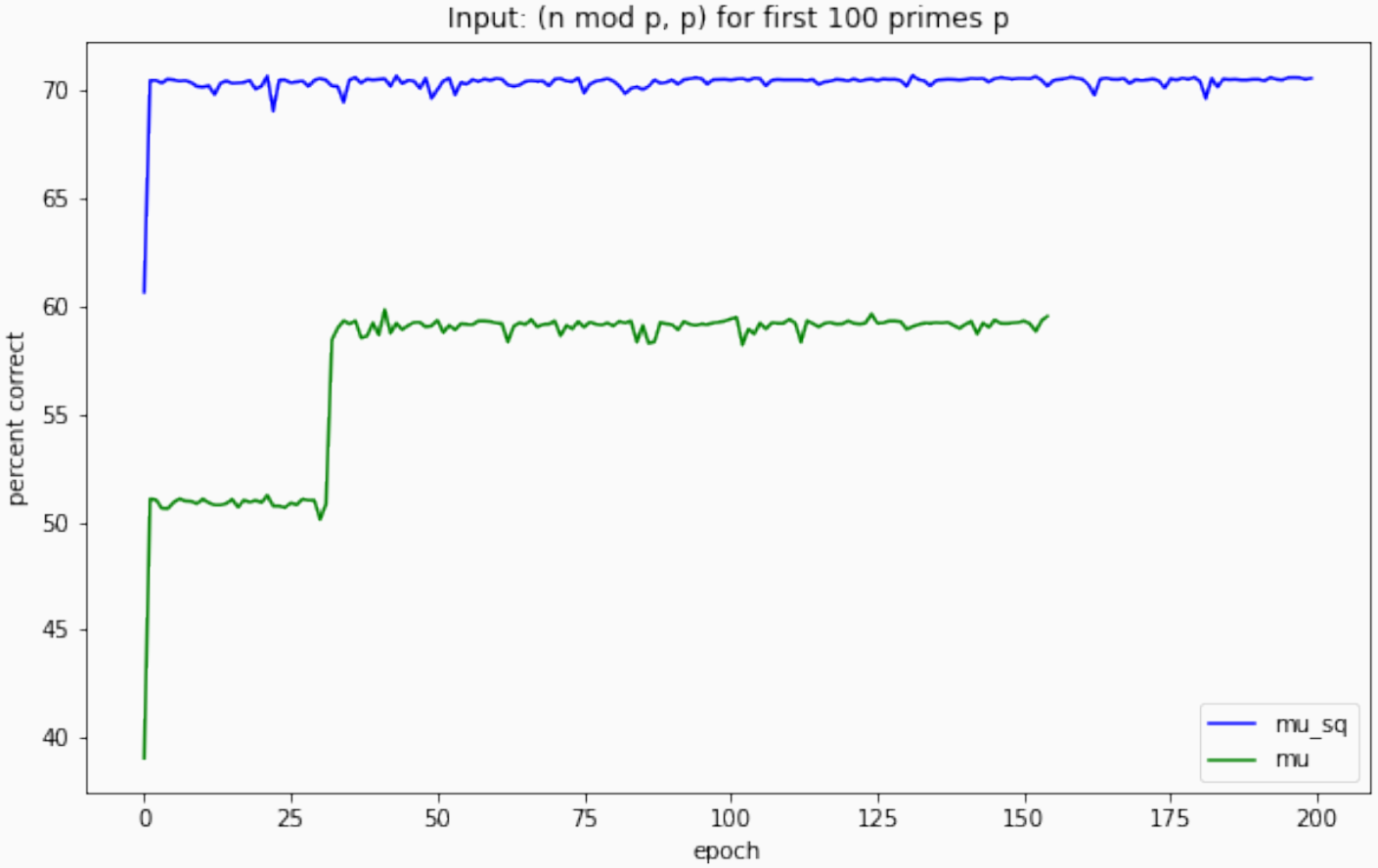
How do we prevent the machine from learning about squares?

One approach is to choose an input representation that deliberately makes it hard to determine divisibility by squares.

For example, let's use a Chinese Remainder Theorem representation. Represent n by the sequence

$$(n \bmod 2, n \bmod 3, n \bmod 5, n \bmod 7, \ldots, n \bmod 541) \in \mathbb{Z}^{100}.$$

Using these sequences as inputs and $\mu(n)$ or $\mu^2(n)$ as outputs for a neural network, what happens?



Vastly better than chance. But is it interesting?

When I saw this, I had no idea what was going on.

One way to see what matters is to deliberately ruin inputs. For example: apply random permutations to input sequences and see which permutations matter or don't matter. On the ML side, this is computationally efficient. It's not necessary to retrain the model.

Applied here, one sees that any permutation that leaves

$$n \bmod 2, n \bmod 3, n \bmod 5$$

alone has practically no effect on accuracy. Indeed, training a model on *only* these three inputs yields almost the same accuracy.

Most numbers that are divisible by a square are divisible by 2, 3, or 5.

In fact, one can show that knowing $n \bmod 6$ is enough to guess $\mu^2(n)$ approximately 70 percent of the time. This is purely correlative and completely independent of any true algorithm to determine $\mu(n)$.

> *On the one hand, this shows some dangers of applying ML blindly. In many ways accuracy does not mean understanding.*
>
> *On the other hand, I learned something about the distribution of squarefree numbers. It wasn't particularly deep, but it was largely taught to me by a machine.*
>
> *That is also powerful.*

Computing Maass Forms

I spend a lot of time thinking about Maass forms. You can learn a lot about them on the LMFDB. Check them out!

`https://www.lmfdb.org/ModularForm/GL2/Q/Maass/`

What is a Maass form?

The Laplace-Beltrami operator acting on the upper halfplane $\mathcal{H}$ with the hyperbolic metric is given by

$$\Delta = -y^2 \left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} \right).$$

A Maass cuspform (of weight 0 and trivial nebentypus on $\Gamma_0(N)$) is a real-analytic eigenfunction of Δ satisfying

1. $\Delta f = \lambda f$ (f has the eigenvalue λ)
2. $f(\gamma z) = f(z)$ (f is invariant under $\Gamma_0(N)$)
3. $f \in L^2(\Gamma_0(N) \backslash \mathcal{H})$ (f is a cuspform)

Why care about Maass forms?

Maass cuspforms form the discrete component of the spectral resolution of Δ. Stated differently, any $g \in L^2(\Gamma_0(N)\backslash H)$ will have an expansion of the form

$$g(z) = \sum_{\lambda \text{ eigenvalue}} \langle f_\lambda, g \rangle f_\lambda(z) + \sum_{\text{cusps}} (\text{Eisenstein series}).$$

Maass forms also have L-functions and are (a poorly understood) part of the Langlands program.

One major challenge is that all data associated to a generic Maass form is conjecturally transcendental, and conjecturally algebraically independent from each other and reasonable constants.

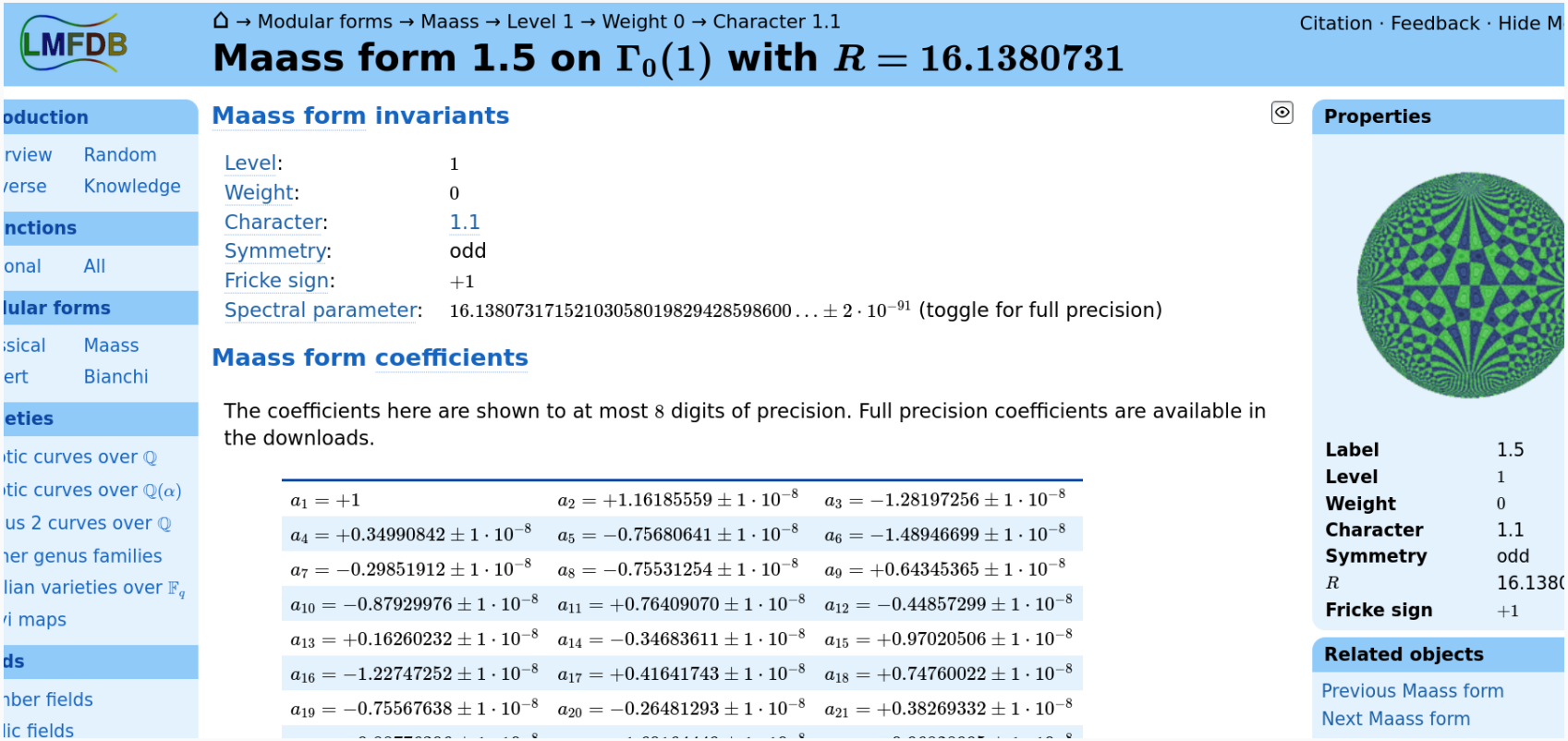
Each Maass form has an expansion

$$f_\lambda(z) = \sum_{n \geq 1} \frac{a(m)}{\sqrt{m}} W_\lambda(2\pi m y) \mathrm{cs}(2\pi m x),$$

where W_λ is a Whittaker function (a modified K-Bessel function of the third kind) and $\mathsf{cs}(\cdot)$ is either $\mathsf{cos}(\cdot)$ or $\mathsf{sin}(\cdot)$, depending on the symmetry type of the Maass form.

Andrei Seymour-Howell and I worked hard to *rigorously* compute as many Maass forms as possible. (Most of these are now in the LMFDB). By this, we mean to give intervals guaranteed to contain the eigenvalue λ and the coefficients $a(m)$.

For example: `https://www.lmfdb.org/ModularForm/GL2/Q/Maass/1.0.1.5.1`



LMFDB

⌂ → Modular forms → Maass → Level 1 → Weight 0 → Character 1.1

Citation · Feedback · Hide M

Maass form 1.5 on $\Gamma_0(1)$ with $R = 16.1380731$

oduction
rview Random
/erse Knowledge
nctions
onal All
lular forms
sical Maass
ert Bianchi
eties
tic curves over $\mathbb{Q}$
tic curves over $\mathbb{Q}(\alpha)$
us 2 curves over $\mathbb{Q}$
her genus families
lian varieties over $\mathbb{F}_q$
i maps
ds
ber fields
lic fields

Maass form invariants

Level: 1
Weight: 0
Character: 1.1
Symmetry: odd
Fricke sign: $+1$
Spectral parameter: $16.138073171521030580198294285986 00 \ldots \pm 2 \cdot 10^{-91}$ (toggle for full precision)

Maass form coefficients

The coefficients here are shown to at most 8 digits of precision. Full precision coefficients are available in the downloads.

$a_1 = +1$	$a_2 = +1.16185559 \pm 1 \cdot 10^{-8}$	$a_3 = -1.28197256 \pm 1 \cdot 10^{-8}$
$a_4 = +0.34990842 \pm 1 \cdot 10^{-8}$	$a_5 = -0.75680641 \pm 1 \cdot 10^{-8}$	$a_6 = -1.48946699 \pm 1 \cdot 10^{-8}$
$a_7 = -0.29851912 \pm 1 \cdot 10^{-8}$	$a_8 = -0.75531254 \pm 1 \cdot 10^{-8}$	$a_9 = +0.64345365 \pm 1 \cdot 10^{-8}$
$a_{10} = -0.87929976 \pm 1 \cdot 10^{-8}$	$a_{11} = +0.76409070 \pm 1 \cdot 10^{-8}$	$a_{12} = -0.44857299 \pm 1 \cdot 10^{-8}$
$a_{13} = +0.16260232 \pm 1 \cdot 10^{-8}$	$a_{14} = -0.34683611 \pm 1 \cdot 10^{-8}$	$a_{15} = +0.97020506 \pm 1 \cdot 10^{-8}$
$a_{16} = -1.22747252 \pm 1 \cdot 10^{-8}$	$a_{17} = +0.41641743 \pm 1 \cdot 10^{-8}$	$a_{18} = +0.74760022 \pm 1 \cdot 10^{-8}$
$a_{19} = -0.75567638 \pm 1 \cdot 10^{-8}$	$a_{20} = -0.26481293 \pm 1 \cdot 10^{-8}$	$a_{21} = +0.38269332 \pm 1 \cdot 10^{-8}$

Properties

Label	1.5
Level	1
Weight	0
Character	1.1
Symmetry	odd
R	16.138(
Fricke sign	$+1$

Related objects

Previous Maass form
Next Maass form

But we have a obvious problem. Our current method sometimes can't handle coefficients $a(n)$ where $n \mid N$. Specifically, for squarefree N, we know these are $\pm 1/\sqrt{n}$. But sometimes we cannot compute the coefficients to sufficient precision to distinguish between the signs.

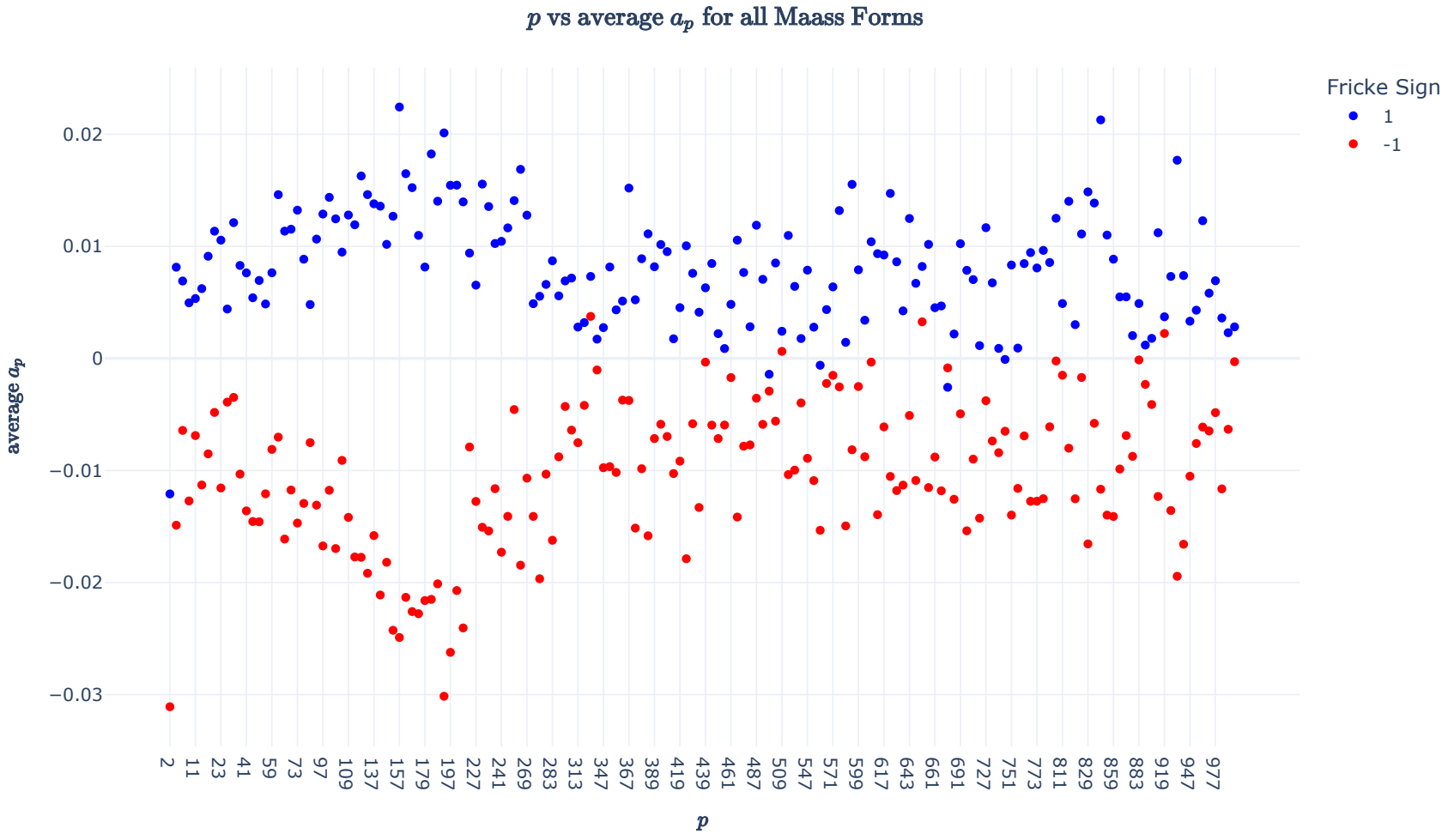
These signs together determine the sign of the Fricke involution.

In different terms, the Maass form has an L-function $L(f_\lambda, s)$ with a functional equation

$$(\sqrt{N}/\pi)^s L(f_\lambda, s)G(s) = \Lambda(f_\lambda, s) = \varepsilon\Lambda(f_\lambda, 1-s).$$

For 15,423 out of the 35,461 Maass forms in the LMFDB, we haven't yet determined ε rigorously.

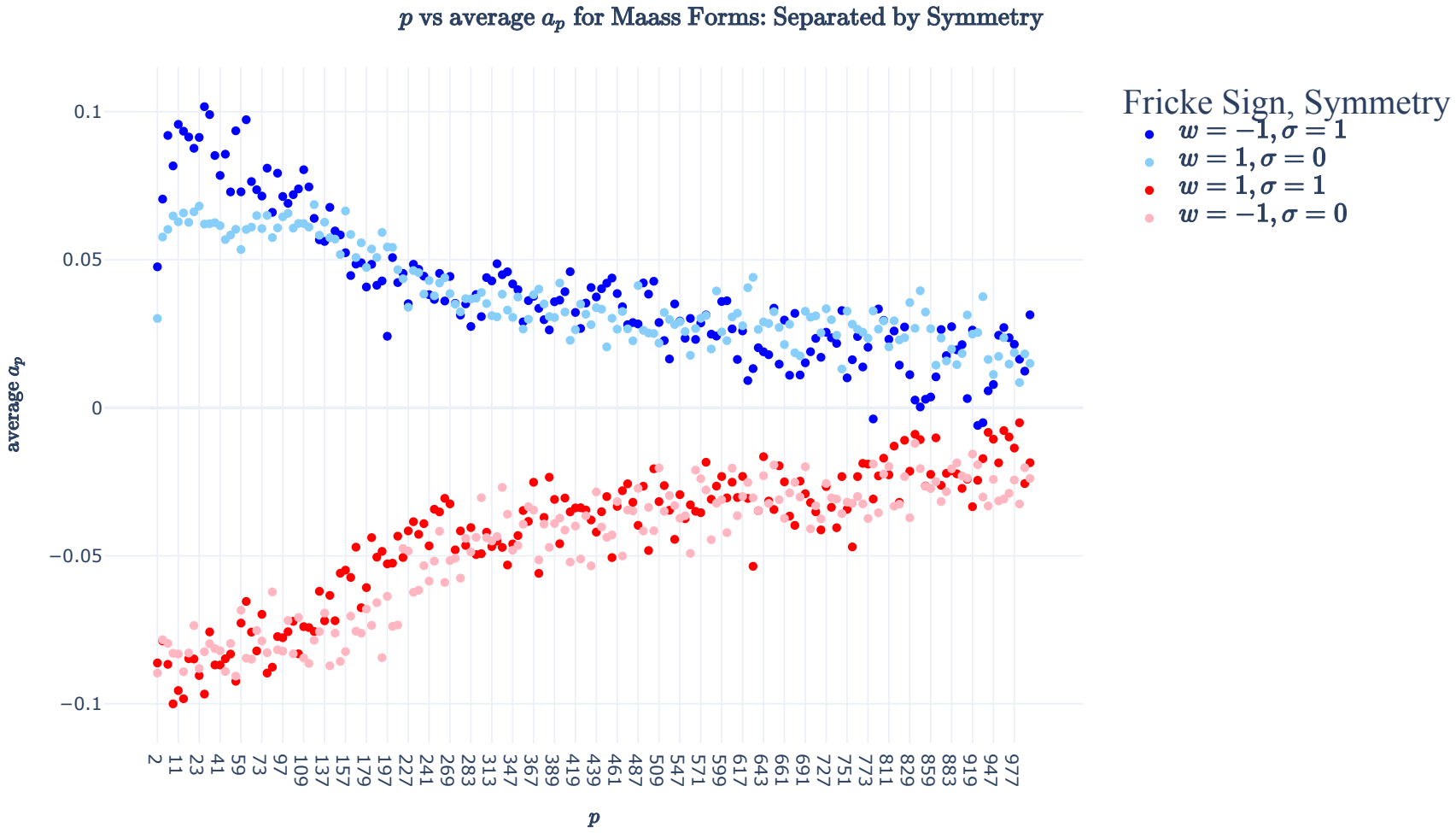
Murmurations!



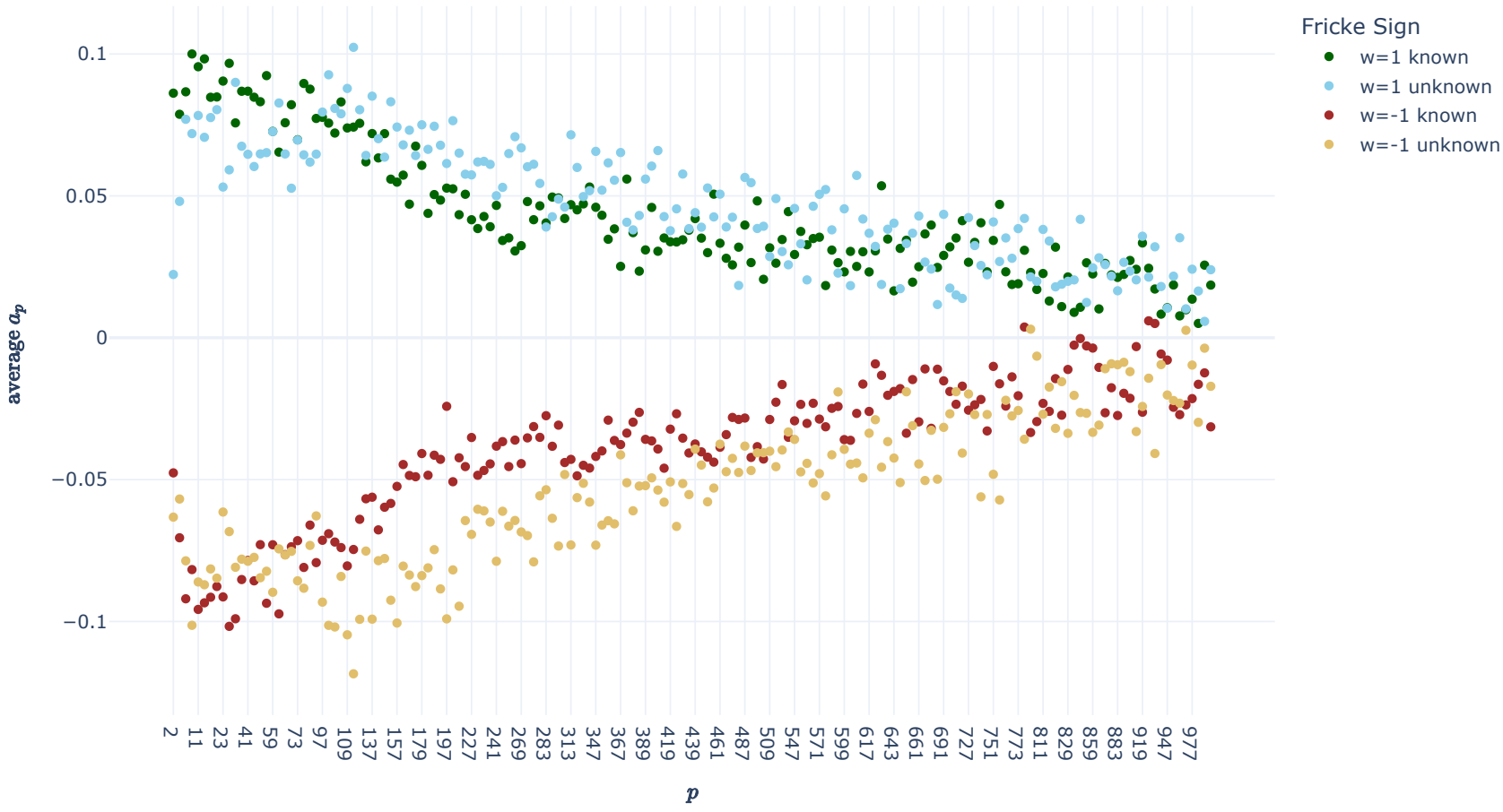
(BLLDSHZ24 proved these murmurations).

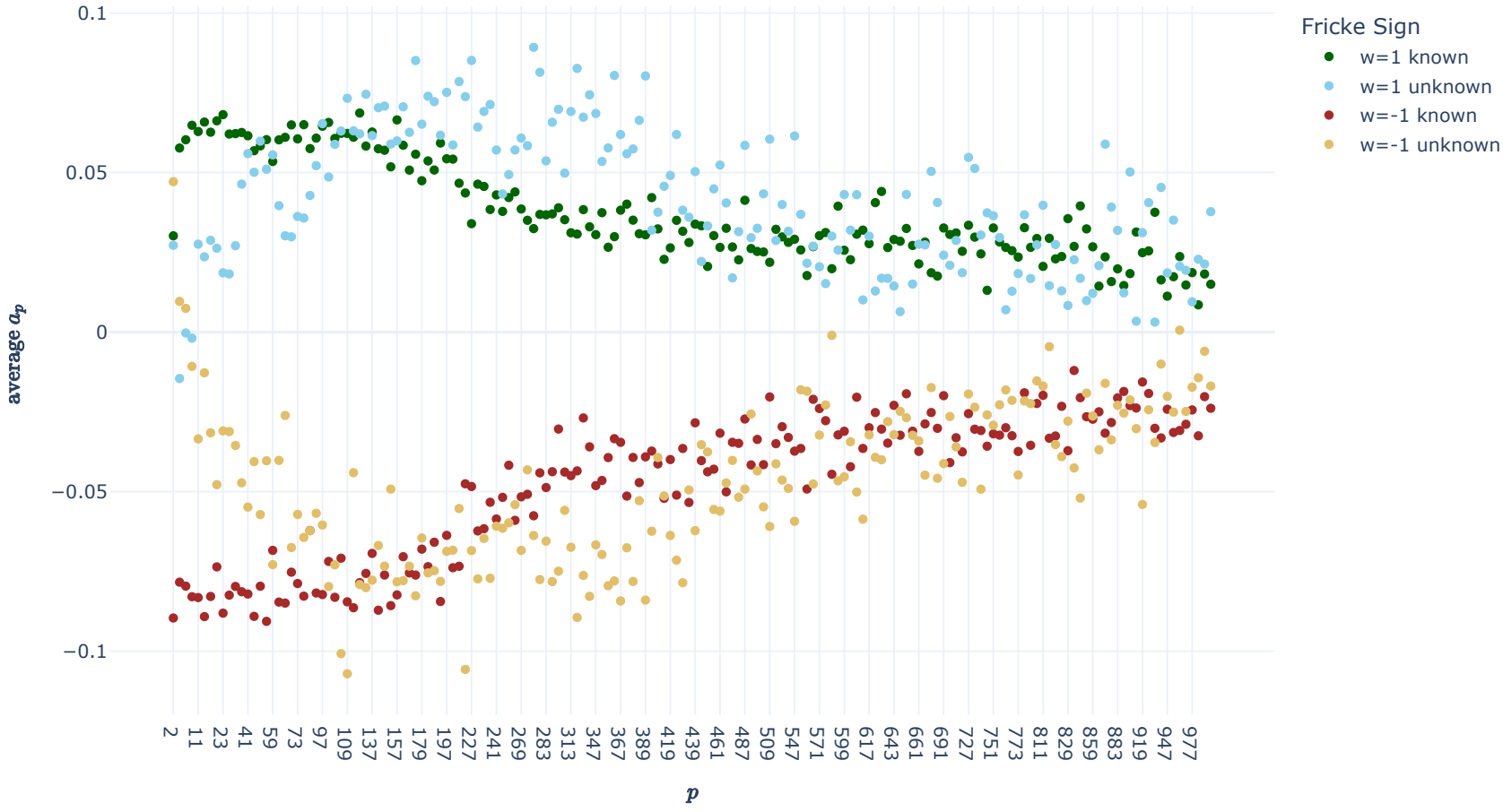
Murmurations with Symmetry!

$\sigma = 0$ (even) or $\sigma = 1$ (odd)



This separation is so clear and straightforward that we should expect even simple ML approaches might learn to distinguish between them.
So we did that (using LDA or neural nets, comparable behavior).
When we include the predicted forms in, we get comparable plots:

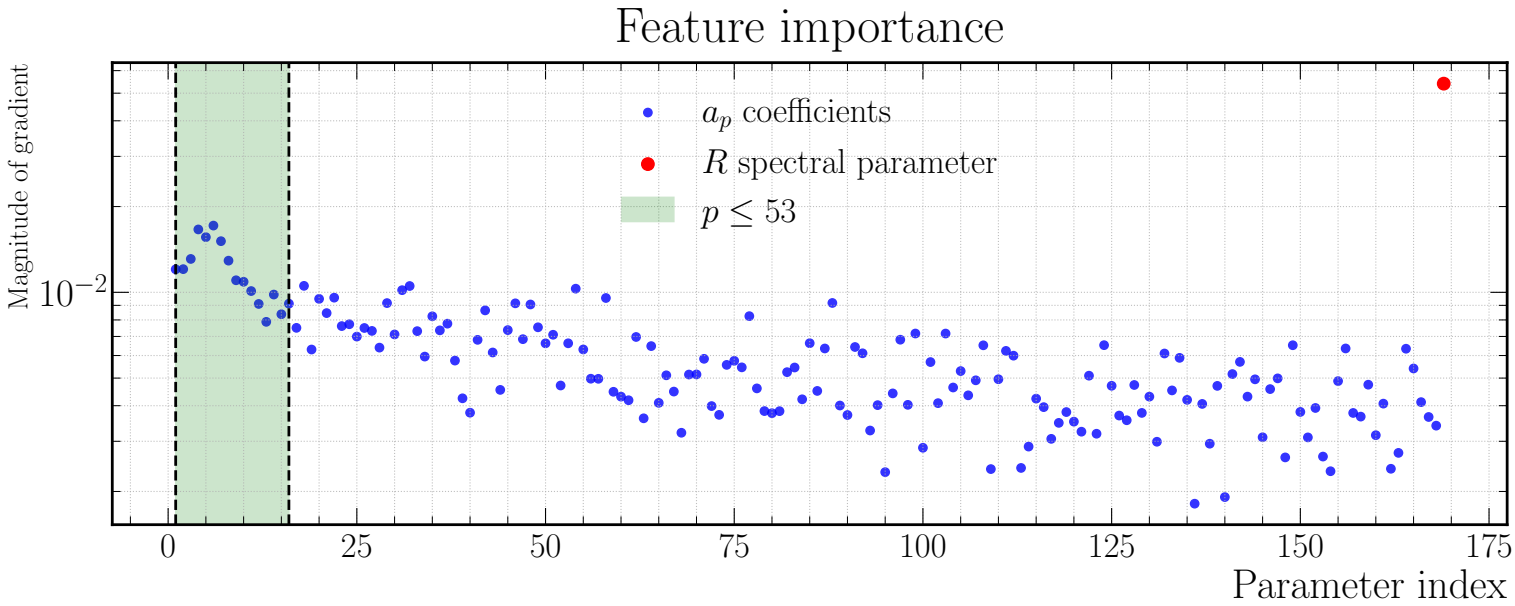

p vs average a_p for ODD Maass forms with known and unknown Fricke sign
Fricke Sign
w=1 known
w=1 unknown
w=-1 known
w=-1 unknown
average a_p
0.1
0.05
0
−0.05
−0.1
2 11 23 41 59 73 97 109 137 157 179 197 227 241 269 283 313 347 367 389 419 439 461 487 509 547 571 599 617 643 661 691 727 751 773 811 829 859 883 919 947 977
p


p vs average a_p for EVEN Maass forms with known and unknown Fricke sign
average a_p
0.1
0.05
0
−0.05
−0.1
2
11
23
41
59
73
97
109
137
157
179
197
227
241
269
283
313
347
367
389
419
439
461
487
509
547
571
599
617
643
661
691
727
751
773
811
829
859
883
919
947
977
p
Fricke Sign
w=1 known
w=1 unknown
w=-1 known
w=-1 unknown

But now we ask: Is this interesting? Is this useful?

As in the Möbius case, we can ask *what information is the ML using?* (Though it's more challenging to answer here).

Permutation occlusion shows that the most salient features are the eigenvalue and the coefficients $a(p)$ with small p.



This is expected. We know three approaches to recognizing constraints that enable one to compute Maass forms:

1. Use the Selberg trace formula. (This is the first step in the LMFDB).
2. Use linear combinations and modularity to generate approximate homogenous systems of equations. (This leads to Hejhal's method, which we use as a second step).
3. Use different test functions in the approximate functional equation to obtain approximate systems of equations. (This leads to work of Farmer and Lemurell).

In each of these, initial coefficients and the spectral data have dominant contributions.

Correctly guessed Fricke involutions directly help the latter two methods.

1. Hejhal: It's no longer necessary to "guess" every potential eigenvalue (as each corresponds to a potentially correct global form).
2. ApproxFE: You can skip directly to creating a coherent system using the correct root number (instead of trying both).

These both save time and are useful to shortcircuit computation.

There is one more application. To apply automorphy, you manipulate the expansions

$$f_\lambda(z) = \sum_{n \geq 1} \frac{a(m)}{\sqrt{m}} W_\lambda(2\pi m y) \mathrm{cs}(2\pi m x).$$

In this expression, the eigenvalue λ, the value z, and the coefficient index m are deeply entangled via the Bessel function W_λ. We can determine if the "easy" estimation learned by the ML is trying to use this by providing the values $W_\lambda(2\pi m y)$ directly for various y. Similarly, to use functional equations, the indices m and eigenvalue λ are entangled via (Fourier transforms of) the Bessel function W_λ. One can compute the Fourier transforms for the first several m and provide these as inputs to the ML models.

But these yield no improvement. ML is picking up something else.

But what is it? Is ML picking up the trace formula at bad primes? (This exists, but isn't worked out — yet! BBKLLDSH are working on it and hope to make this available "soon").

What we know is that using the easy outputs save time in current computational verification, and ablative studies suggests alternate evaluation of new computational directions.

Hopefully we'll have more to say later.

Thank you very much.

Please note that these slides are
(or will soon be) available on my website
davidlowryduda.com.