Carnegie Mellon University

Broad Perspectives on Formal Mathematics

Formal Mathematics and AI at Carnegie Mellon, New York City

Sidharth Hariharan

18 September 2026

sidharthhariharan@cmu.edu

Contents

1. The World of Formal Mathematics

2. Autoformalisation

3. AI and the Formal Turn

4. Canonisation, Trust and Understanding

The World of Formal Mathematics

What is an Interactive Theorem Prover?

What is an Interactive Theorem Prover?

An interactive theorem prover (ITP) is a programming language with the capability of checking proofs down to foundational axioms.

What is an Interactive Theorem Prover?

An interactive theorem prover (ITP) is a programming language with the capability of checking proofs down to foundational axioms.

While ITPs have been around for decades, the last $\sim$20 years have seen them take on prominent roles in mainstream mathematics.

What is an Interactive Theorem Prover?

An interactive theorem prover (ITP) is a programming language with the capability of checking proofs down to foundational axioms.

While ITPs have been around for decades, the last $\sim$20 years have seen them take on prominent roles in mainstream mathematics.

The most commonly used ITPs are Lean, Rocq, Isabelle, HOL Light, and Agda.

What is an Interactive Theorem Prover?

An interactive theorem prover (ITP) is a programming language with the capability of checking proofs down to foundational axioms.

While ITPs have been around for decades, the last $\sim$20 years have seen them take on prominent roles in mainstream mathematics.

The most commonly used ITPs are Lean, Rocq, Isabelle, HOL Light, and Agda. They have different foundations, meaning some areas of mathematics are easier to formalise in some ITPs than others.

What is an Interactive Theorem Prover?

An interactive theorem prover (ITP) is a programming language with the capability of checking proofs down to foundational axioms.

While ITPs have been around for decades, the last $\sim$20 years have seen them take on prominent roles in mainstream mathematics.

The most commonly used ITPs are Lean, Rocq, Isabelle, HOL Light, and Agda. They have different foundations, meaning some areas of mathematics are easier to formalise in some ITPs than others.

ITPs typically contain small kernels that are isolated from the rest of the system. It is the kernel that checks correctness.

What is an Interactive Theorem Prover?

An interactive theorem prover (ITP) is a programming language with the capability of checking proofs down to foundational axioms.

While ITPs have been around for decades, the last $\sim$20 years have seen them take on prominent roles in mainstream mathematics.

The most commonly used ITPs are Lean, Rocq, Isabelle, HOL Light, and Agda. They have different foundations, meaning some areas of mathematics are easier to formalise in some ITPs than others.

ITPs typically contain small kernels that are isolated from the rest of the system. It is the kernel that checks correctness.

Tactics, automation and (as we shall see) AI models can all be used to write proofs.

What is an Interactive Theorem Prover?

An interactive theorem prover (ITP) is a programming language with the capability of checking proofs down to foundational axioms.

While ITPs have been around for decades, the last $\sim$20 years have seen them take on prominent roles in mainstream mathematics.

The most commonly used ITPs are Lean, Rocq, Isabelle, HOL Light, and Agda. They have different foundations, meaning some areas of mathematics are easier to formalise in some ITPs than others.

ITPs typically contain small kernels that are isolated from the rest of the system. It is the kernel that checks correctness.

Tactics, automation and (as we shall see) AI models can all be used to write proofs.

Historically, there have been many advantages to formalisation, beyond just correctness, including

What is an Interactive Theorem Prover?

An interactive theorem prover (ITP) is a programming language with the capability of checking proofs down to foundational axioms.

While ITPs have been around for decades, the last $\sim$20 years have seen them take on prominent roles in mainstream mathematics.

The most commonly used ITPs are Lean, Rocq, Isabelle, HOL Light, and Agda. They have different foundations, meaning some areas of mathematics are easier to formalise in some ITPs than others.

ITPs typically contain small kernels that are isolated from the rest of the system. It is the kernel that checks correctness.

Tactics, automation and (as we shall see) AI models can all be used to write proofs.

Historically, there have been many advantages to formalisation, beyond just correctness, including developing theory,

What is an Interactive Theorem Prover?

An interactive theorem prover (ITP) is a programming language with the capability of checking proofs down to foundational axioms.

While ITPs have been around for decades, the last $\sim$20 years have seen them take on prominent roles in mainstream mathematics.

The most commonly used ITPs are Lean, Rocq, Isabelle, HOL Light, and Agda. They have different foundations, meaning some areas of mathematics are easier to formalise in some ITPs than others.

ITPs typically contain small kernels that are isolated from the rest of the system. It is the kernel that checks correctness.

Tactics, automation and (as we shall see) AI models can all be used to write proofs.

Historically, there have been many advantages to formalisation, beyond just correctness, including developing theory, building understanding,

What is an Interactive Theorem Prover?

An interactive theorem prover (ITP) is a programming language with the capability of checking proofs down to foundational axioms.

While ITPs have been around for decades, the last $\sim$20 years have seen them take on prominent roles in mainstream mathematics.

The most commonly used ITPs are Lean, Rocq, Isabelle, HOL Light, and Agda. They have different foundations, meaning some areas of mathematics are easier to formalise in some ITPs than others.

ITPs typically contain small kernels that are isolated from the rest of the system. It is the kernel that checks correctness.

Tactics, automation and (as we shall see) AI models can all be used to write proofs.

Historically, there have been many advantages to formalisation, beyond just correctness, including developing theory, building understanding, easing collaboration,

What is an Interactive Theorem Prover?

An interactive theorem prover (ITP) is a programming language with the capability of checking proofs down to foundational axioms.

While ITPs have been around for decades, the last $\sim$20 years have seen them take on prominent roles in mainstream mathematics.

The most commonly used ITPs are Lean, Rocq, Isabelle, HOL Light, and Agda. They have different foundations, meaning some areas of mathematics are easier to formalise in some ITPs than others.

ITPs typically contain small kernels that are isolated from the rest of the system. It is the kernel that checks correctness.

Tactics, automation and (as we shall see) AI models can all be used to write proofs.

Historically, there have been many advantages to formalisation, beyond just correctness, including developing theory, building understanding, easing collaboration, and centralising authoritatively.

A Brief History of Interactive Theorem Provers

A Brief History of Interactive Theorem Provers

- **1967:** de Bruijn creates **Automath**, which implements propositions as types and checks proofs using an isolated kernel (sometimes called the “de Bruijn Principle”)

A Brief History of Interactive Theorem Provers

- **1967:** de Bruijn creates **Automath**, which implements propositions as types and checks proofs using an isolated kernel (sometimes called the “de Bruijn Principle”)
- **1973:** Trybulec gives an influential presentation leading to the development of **Mizar**, which has a set-theoretic foundation

A Brief History of Interactive Theorem Provers

- **1967:** de Bruijn creates **Automath**, which implements propositions as types and checks proofs using an isolated kernel (sometimes called the “de Bruijn Principle”)
- **1973:** Trybulec gives an influential presentation leading to the development of **Mizar**, which has a set-theoretic foundation
- **1984:** T. Coquand and Huet implement the Calculus of Constructions at INRIA-Rocquencourt.

A Brief History of Interactive Theorem Provers

- **1967:** de Bruijn creates **Automath**, which implements propositions as types and checks proofs using an isolated kernel (sometimes called the “de Bruijn Principle”)
- **1973:** Trybulec gives an influential presentation leading to the development of **Mizar**, which has a set-theoretic foundation
- **1984:** T. Coquand and Huet implement the Calculus of Constructions at INRIA-Rocquencourt.
- **1986:** Paulson launches **Isabelle**, based on simple type theory

A Brief History of Interactive Theorem Provers

- **1967:** de Bruijn creates **Automath**, which implements propositions as types and checks proofs using an isolated kernel (sometimes called the “de Bruijn Principle”)
- **1973:** Trybulec gives an influential presentation leading to the development of **Mizar**, which has a set-theoretic foundation
- **1984:** T. Coquand and Huet implement the Calculus of Constructions at INRIA-Rocquencourt.
- **1986:** Paulson launches **Isabelle**, based on simple type theory
- **1989:** T. Coquand and Paulin expand Coquand’s implementation to the Calculus of Inductive Constructions, leading to the first version of **Coq** (recently renamed **Rocq**).

A Brief History of Interactive Theorem Provers

- **1967:** de Bruijn creates **Automath**, which implements propositions as types and checks proofs using an isolated kernel (sometimes called the “de Bruijn Principle”)
- **1973:** Trybulec gives an influential presentation leading to the development of **Mizar**, which has a set-theoretic foundation
- **1984:** T. Coquand and Huet implement the Calculus of Constructions at INRIA-Rocquencourt.
- **1986:** Paulson launches **Isabelle**, based on simple type theory
- **1989:** T. Coquand and Paulin expand Coquand’s implementation to the Calculus of Inductive Constructions, leading to the first version of **Coq** (recently renamed **Rocq**).
- **1996–98:** C. Coquand implements the original version of Agda in Haskell

A Brief History of Interactive Theorem Provers

- **1967:** de Bruijn creates **Automath**, which implements propositions as types and checks proofs using an isolated kernel (sometimes called the “de Bruijn Principle”)
- **1973:** Trybulec gives an influential presentation leading to the development of **Mizar**, which has a set-theoretic foundation
- **1984:** T. Coquand and Huet implement the Calculus of Constructions at INRIA-Rocquencourt.
- **1986:** Paulson launches **Isabelle**, based on simple type theory
- **1989:** T. Coquand and Paulin expand Coquand’s implementation to the Calculus of Inductive Constructions, leading to the first version of **Coq** (recently renamed **Rocq**).
- **1996–98:** C. Coquand implements the original version of Agda in Haskell
- **2013:** de Moura creates the first version of **Lean**

21st Century Formalisation Milestones

21st Century Formalisation Milestones

2003 — Hales launches **Flyspeck**: a formal proof of the Kepler conjecture

21st Century Formalisation Milestones

2003 — Hales launches **Flyspeck**: a formal proof of the Kepler conjecture

2004 — **Prime Number Theorem** (elementary proof) in Isabelle: Avigad, Donnelly, Gray, Raff [1]

21st Century Formalisation Milestones

2003 Hales launches **Flyspeck**: a formal proof of the Kepler conjecture

2004 **Prime Number Theorem** (elementary proof) in Isabelle: Avigad, Donnelly, Gray, Raff [1]

2005 **Four Colour Theorem** in Coq: Gonthier, with Werner [3]. **Jordan Curve Theorem** in HOL Light: Hales [5]

21st Century Formalisation Milestones

- **2003** — Hales launches **Flyspeck**: a formal proof of the Kepler conjecture
- **2004** — **Prime Number Theorem** (elementary proof) in Isabelle: Avigad, Donnelly, Gray, Raff [1]
- **2005** — **Four Colour Theorem** in Coq: Gonthier, with Werner [3]. **Jordan Curve Theorem** in HOL Light: Hales [5]
- **2009** — **Prime Number Theorem** (analytic proof) in HOL Light: Harrison [7]

21st Century Formalisation Milestones

- **2003** — Hales launches **Flyspeck**: a formal proof of the Kepler conjecture
- **2004** — **Prime Number Theorem** (elementary proof) in Isabelle: Avigad, Donnelly, Gray, Raff [1]
- **2005** — **Four Colour Theorem** in Coq: Gonthier, with Werner [3]. **Jordan Curve Theorem** in HOL Light: Hales [5]
- **2009** — **Prime Number Theorem** (analytic proof) in HOL Light: Harrison [7]
- **2010s** — Isabelle's **Archive of Formal Proofs** grows into a large research library [2]

21st Century Formalisation Milestones

- **2003** — Hales launches **Flyspeck**: a formal proof of the Kepler conjecture
- **2004** — **Prime Number Theorem** (elementary proof) in Isabelle: Avigad, Donnelly, Gray, Raff [1]
- **2005** — **Four Colour Theorem** in Coq: Gonthier, with Werner [3]. **Jordan Curve Theorem** in HOL Light: Hales [5]
- **2009** — **Prime Number Theorem** (analytic proof) in HOL Light: Harrison [7]
- **2010s** — Isabelle's **Archive of Formal Proofs** grows into a large research library [2]
- **2012** — **Odd Order Theorem** in Coq: Gonthier, Asperti, Avigad et al. [4]

21st Century Formalisation Milestones

- **2003** — Hales launches **Flyspeck**: a formal proof of the Kepler conjecture
- **2004** — **Prime Number Theorem** (elementary proof) in Isabelle: Avigad, Donnelly, Gray, Raff [1]
- **2005** — **Four Colour Theorem** in Coq: Gonthier, with Werner [3]. **Jordan Curve Theorem** in HOL Light: Hales [5]
- **2009** — **Prime Number Theorem** (analytic proof) in HOL Light: Harrison [7]
- **2010s** — Isabelle's **Archive of Formal Proofs** grows into a large research library [2]
- **2012** — **Odd Order Theorem** in Coq: Gonthier, Asperti, Avigad et al. [4]
- **2012–13** — Special Year on **Univalent Foundations** at the IAS: Awodey, Coquand, Voevodsky; the *HoTT Book* [9]

21st Century Formalisation Milestones

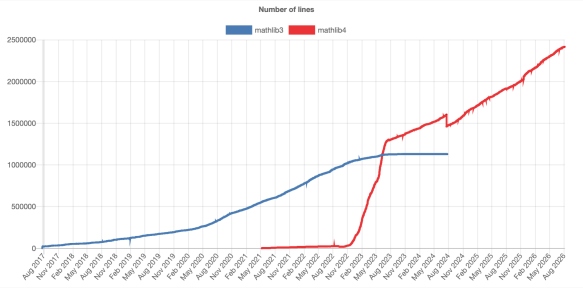
- **2003** — Hales launches **Flyspeck**: a formal proof of the Kepler conjecture
- **2004** — **Prime Number Theorem** (elementary proof) in Isabelle: Avigad, Donnelly, Gray, Raff [1]
- **2005** — **Four Colour Theorem** in Coq: Gonthier, with Werner [3]. **Jordan Curve Theorem** in HOL Light: Hales [5]
- **2009** — **Prime Number Theorem** (analytic proof) in HOL Light: Harrison [7]
- **2010s** — Isabelle's **Archive of Formal Proofs** grows into a large research library [2]
- **2012** — **Odd Order Theorem** in Coq: Gonthier, Asperti, Avigad et al. [4]
- **2012–13** — Special Year on **Univalent Foundations** at the IAS: Awodey, Coquand, Voevodsky; the *HoTT Book* [9]
- **2014** — **Flyspeck** completed, eleven years on; published in 2017 [6]

21st Century Formalisation Milestones

2003	Hales launches **Flyspeck**: a formal proof of the Kepler conjecture
2004	**Prime Number Theorem** (elementary proof) in Isabelle: Avigad, Donnelly, Gray, Raff [1]
2005	**Four Colour Theorem** in Coq: Gonthier, with Werner [3]. **Jordan Curve Theorem** in HOL Light: Hales [5]
2009	**Prime Number Theorem** (analytic proof) in HOL Light: Harrison [7]
2010s	Isabelle's **Archive of Formal Proofs** grows into a large research library [2]
2012	**Odd Order Theorem** in Coq: Gonthier, Asperti, Avigad et al. [4]
2012–13	Special Year on **Univalent Foundations** at the IAS: Awodey, Coquand, Voevodsky; the *HoTT Book* [9]
2014	**Flyspeck** completed, eleven years on; published in 2017 [6]
2017	***Mathlib* separates from Lean Core and becomes a standalone library of formalised mathematics**

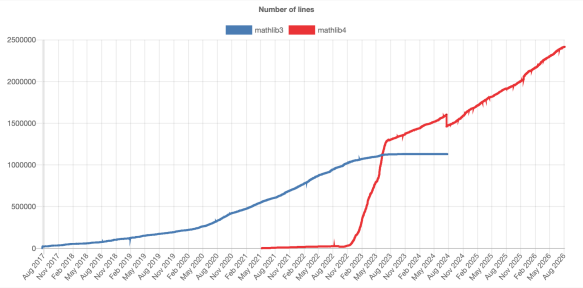
The Development of Mathlib

The Development of Mathlib



Taken from `https://leanprover-community.github.io/mathlib_stats.html`

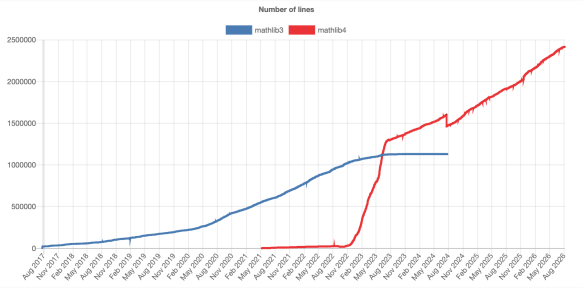
The Development of Mathlib



Taken from `https://leanprover-community.github.io/mathlib_stats.html`

Mathlib seeks to be highly optimised for downstream use. To that end:

The Development of Mathlib

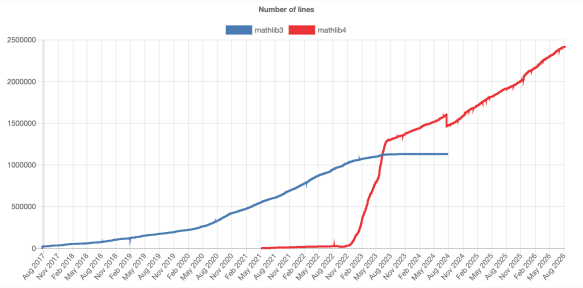


Taken from `https://leanprover-community.github.io/mathlib_stats.html`

Mathlib seeks to be highly optimised for downstream use. To that end:

- Theorems are stated in maximal (reasonable) generality.

The Development of Mathlib

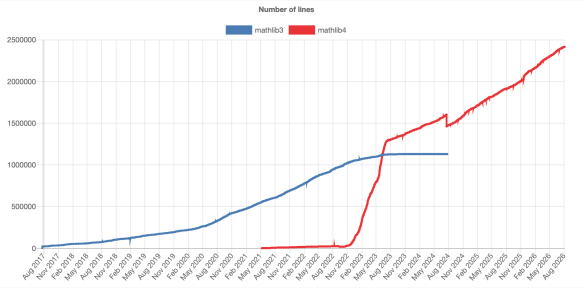


Taken from `https://leanprover-community.github.io/mathlib_stats.html`

Mathlib seeks to be highly optimised for downstream use. To that end:

- Theorems are stated in maximal (reasonable) generality.
- Definitions are designed to be flexible enough to accommodate generality while still serving the desired purpose.

The Development of Mathlib

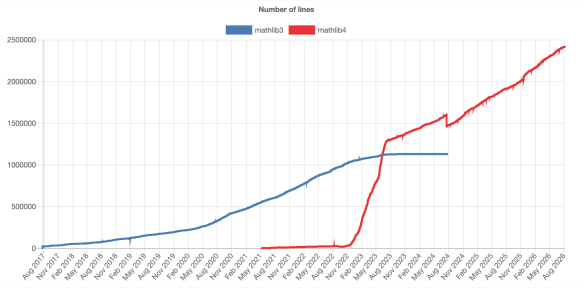


Taken from `https://leanprover-community.github.io/mathlib_stats.html`

Mathlib seeks to be highly optimised for downstream use. To that end:

- Theorems are stated in maximal (reasonable) generality.
- Definitions are designed to be flexible enough to accommodate generality while still serving the desired purpose.
- Each definition is accompanied by extensive API.

The Development of Mathlib

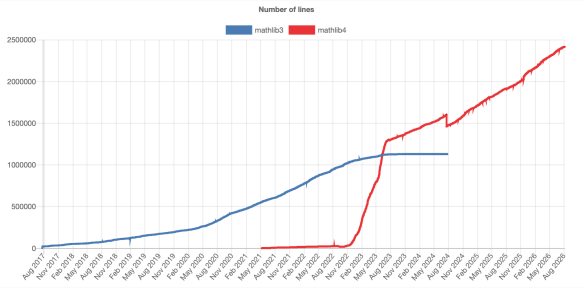


Taken from `https://leanprover-community.github.io/mathlib_stats.html`

Mathlib seeks to be highly optimised for downstream use. To that end:

- Theorems are stated in maximal (reasonable) generality.
- Definitions are designed to be flexible enough to accommodate generality while still serving the desired purpose.
- Each definition is accompanied by extensive API.
- Commonly used techniques in proofs are combined into tactics.

The Development of Mathlib



Taken from `https://leanprover-community.github.io/mathlib_stats.html`

Mathlib seeks to be highly optimised for downstream use. To that end:

- Theorems are stated in maximal (reasonable) generality.
- Definitions are designed to be flexible enough to accommodate generality while still serving the desired purpose.
- Each definition is accompanied by extensive API.
- Commonly used techniques in proofs are combined into tactics.
- Naming and style conventions are strictly enforced.

The Formal Microcosm

The Formal Microcosm

I think of formal mathematics as a different way of doing mathematics.

The Formal Microcosm

I think of formal mathematics as a different way of doing mathematics. A bit like doing maths in a different language, but with greater demands.

The Formal Microcosm

I think of formal mathematics as a different way of doing mathematics. A bit like doing maths in a different language, but with greater demands.

Formal	General

The Formal Microcosm

I think of formal mathematics as a different way of doing mathematics. A bit like doing maths in a different language, but with greater demands.

	Formal	General
Novelty	New formalisation	Solving an open problem

The Formal Microcosm

I think of formal mathematics as a different way of doing mathematics. A bit like doing maths in a different language, but with greater demands.

	Formal	General
Novelty	New formalisation	Solving an open problem
Theory-Building	Building formal infrastructure	Developing new methods, ideas, techniques

The Formal Microcosm

I think of formal mathematics as a different way of doing mathematics. A bit like doing maths in a different language, but with greater demands.

	Formal	General
Novelty	New formalisation	Solving an open problem
Theory-Building	Building formal infrastructure	Developing new methods, ideas, techniques
End-Product	A file/codebase/repository	A paper

The Formal Microcosm

I think of formal mathematics as a different way of doing mathematics. A bit like doing maths in a different language, but with greater demands.

	Formal	General
Novelty	New formalisation	Solving an open problem
Theory-Building	Building formal infrastructure	Developing new methods, ideas, techniques
End-Product	A file/codebase/repository	A paper
Consolidation	Upstreaming to existing libraries/creating a new library	Submitting to a journal/writing a textbook

The Formal Microcosm

I think of formal mathematics as a different way of doing mathematics. A bit like doing maths in a different language, but with greater demands.

	Formal	General
Novelty	New formalisation	Solving an open problem
Theory-Building	Building formal infrastructure	Developing new methods, ideas, techniques
End-Product	A file/codebase/repository	A paper
Consolidation	Upstreaming to existing libraries/creating a new library	Submitting to a journal/writing a textbook
Understanding	Comes from studying, writing, revising and reviewing code	Comes from studying, writing, revising and reviewing literature

The Formal Microcosm

I think of formal mathematics as a different way of doing mathematics. A bit like doing maths in a different language, but with greater demands.

	Formal	General
Novelty	New formalisation	Solving an open problem
Theory-Building	Building formal infrastructure	Developing new methods, ideas, techniques
End-Product	A file/codebase/repository	A paper
Consolidation	Upstreaming to existing libraries/creating a new library	Submitting to a journal/writing a textbook
Understanding	Comes from studying, writing, revising and reviewing code	Comes from studying, writing, revising and reviewing literature
⋮	⋮	⋮

The Formal Microcosm

I think of formal mathematics as a different way of doing mathematics. A bit like doing maths in a different language, but with greater demands.

	Formal	General
Novelty	New formalisation	Solving an open problem
Theory-Building	Building formal infrastructure	Developing new methods, ideas, techniques
End-Product	A file/codebase/repository	A paper
Consolidation	Upstreaming to existing libraries/creating a new library	Submitting to a journal/writing a textbook
Understanding	Comes from studying, writing, revising and reviewing code	Comes from studying, writing, revising and reviewing literature
⋮	⋮	⋮

Some things are easier in the formal medium:

The Formal Microcosm

I think of formal mathematics as a different way of doing mathematics. A bit like doing maths in a different language, but with greater demands.

	Formal	General
Novelty	New formalisation	Solving an open problem
Theory-Building	Building formal infrastructure	Developing new methods, ideas, techniques
End-Product	A file/codebase/repository	A paper
Consolidation	Upstreaming to existing libraries/creating a new library	Submitting to a journal/writing a textbook
Understanding	Comes from studying, writing, revising and reviewing code	Comes from studying, writing, revising and reviewing literature
⋮	⋮	⋮

Some things are easier in the formal medium: collaboration, unambiguous communication, refactoring/generalising, black-boxing ‘known’ results

The Formal Microcosm

I think of formal mathematics as a different way of doing mathematics. A bit like doing maths in a different language, but with greater demands.

	Formal	General
Novelty	New formalisation	Solving an open problem
Theory-Building	Building formal infrastructure	Developing new methods, ideas, techniques
End-Product	A file/codebase/repository	A paper
Consolidation	Upstreaming to existing libraries/creating a new library	Submitting to a journal/writing a textbook
Understanding	Comes from studying, writing, revising and reviewing code	Comes from studying, writing, revising and reviewing literature
⋮	⋮	⋮

Some things are easier in the formal medium: collaboration, unambiguous communication, refactoring/generalising, black-boxing ‘known’ results

Other things are easier in the non-formal medium:

The Formal Microcosm

I think of formal mathematics as a different way of doing mathematics. A bit like doing maths in a different language, but with greater demands.

	Formal	General
Novelty	New formalisation	Solving an open problem
Theory-Building	Building formal infrastructure	Developing new methods, ideas, techniques
End-Product	A file/codebase/repository	A paper
Consolidation	Upstreaming to existing libraries/creating a new library	Submitting to a journal/writing a textbook
Understanding	Comes from studying, writing, revising and reviewing code	Comes from studying, writing, revising and reviewing literature
⋮	⋮	⋮

Some things are easier in the formal medium: collaboration, unambiguous communication, refactoring/generalising, black-boxing ‘known’ results

Other things are easier in the non-formal medium: reading other people’s work, reasoning intuitively

Autoformalisation

Why Formalisation Matters in the Age of AI

Why Formalisation Matters in the Age of AI

Why is formalisation important for AI?

Why is formalisation important for AI?

- Verified training data

Why Formalisation Matters in the Age of AI

Why is formalisation important for AI?

- Verified training data
- Instant feedback from a compiler

Why Formalisation Matters in the Age of AI

Why is formalisation important for AI?

- Verified training data
- Instant feedback from a compiler

Why is AI important for formalisation?

Why Formalisation Matters in the Age of AI

Why is formalisation important for AI?

- Verified training data
- Instant feedback from a compiler

Why is AI important for formalisation?

- Eliminates the need for mathematicians to learn the syntax needed to check correctness

Why Formalisation Matters in the Age of AI

Why is formalisation important for AI?

- Verified training data
- Instant feedback from a compiler

Why is AI important for formalisation?

- Eliminates the need for mathematicians to learn the syntax needed to check correctness
- Reduces the time and effort needed to formalise a proof

Why Formalisation Matters in the Age of AI

Why is formalisation important for AI?

- Verified training data
- Instant feedback from a compiler

Why is AI important for formalisation?

- Eliminates the need for mathematicians to learn the syntax needed to check correctness
- Reduces the time and effort needed to formalise a proof

Why is AI making formalisation more important for mathematics?

Why Formalisation Matters in the Age of AI

Why is formalisation important for AI?

- Verified training data
- Instant feedback from a compiler

Why is AI important for formalisation?

- Eliminates the need for mathematicians to learn the syntax needed to check correctness
- Reduces the time and effort needed to formalise a proof

Why is AI making formalisation more important for mathematics?

- AI is producing a lot of mathematics, and we need to be able to check it

Why Formalisation Matters in the Age of AI

Why is formalisation important for AI?

- Verified training data
- Instant feedback from a compiler

Why is AI important for formalisation?

- Eliminates the need for mathematicians to learn the syntax needed to check correctness
- Reduces the time and effort needed to formalise a proof

Why is AI making formalisation more important for mathematics?

- AI is producing a lot of mathematics, and we need to be able to check it
- Formalisation eases that burden (even if it does not completely eliminate it)

Autoformalisation

Autoformalisation

Autoformalisation is when a machine writes up a proof in a formal theorem prover.

Autoformalisation

Autoformalisation is when a machine writes up a proof in a formal theorem prover.

When we say “Model X has autoformalised (Theorem/Proof/Definition) Y in Theorem Prover Z”, we typically mean

Autoformalisation

Autoformalisation is when a machine writes up a proof in a formal theorem prover.

When we say “Model X has autoformalised (Theorem/Proof/Definition) Y in Theorem Prover Z”, we typically mean

1. Model X wrote Z code that compiles

Autoformalisation

Autoformalisation is when a machine writes up a proof in a formal theorem prover.

When we say “Model X has autoformalised (Theorem/Proof/Definition) Y in Theorem Prover Z”, we typically mean

1. Model X wrote Z code that compiles
2. Model X did so with a reasonable degree of autonomy

Autoformalisation

Autoformalisation is when a machine writes up a proof in a formal theorem prover.

When we say “Model X has autoformalised (Theorem/Proof/Definition) Y in Theorem Prover Z”, we typically mean

1. Model X wrote Z code that compiles
2. Model X did so with a reasonable degree of autonomy
3. The end product of the Z code written by Model X corresponds semantically to the contents of Y

Autoformalisation

Autoformalisation is when a machine writes up a proof in a formal theorem prover.

When we say “Model X has autoformalised (Theorem/Proof/Definition) Y in Theorem Prover Z”, we typically mean

1. Model X wrote Z code that compiles
2. Model X did so with a reasonable degree of autonomy
3. The end product of the Z code written by Model X corresponds semantically to the contents of Y

An autoformalisation could either involve **translating a mathematical endeavour** from natural language to Z or **performing the entire endeavour** in Z.

Autoformalisation

Autoformalisation is when a machine writes up a proof in a formal theorem prover.

When we say “Model X has autoformalised (Theorem/Proof/Definition) Y in Theorem Prover Z”, we typically mean

1. Model X wrote Z code that compiles
2. Model X did so with a reasonable degree of autonomy
3. The end product of the Z code written by Model X corresponds semantically to the contents of Y

An autoformalisation could either involve **translating a mathematical endeavour** from natural language to Z or **performing the entire endeavour** in Z. Often, some combination of these.

Autoformalisation

Autoformalisation is when a machine writes up a proof in a formal theorem prover.

When we say “Model X has autoformalised (Theorem/Proof/Definition) Y in Theorem Prover Z”, we typically mean

1. Model X wrote Z code that compiles
2. Model X did so with a reasonable degree of autonomy
3. The end product of the Z code written by Model X corresponds semantically to the contents of Y

An autoformalisation could either involve **translating a mathematical endeavour** from natural language to Z or **performing the entire endeavour** in Z. Often, some combination of these.

The benefit of the first mode is that it removes the requirement of learning Z to avail its benefits.

Autoformalisation

Autoformalisation is when a machine writes up a proof in a formal theorem prover.

When we say “Model X has autoformalised (Theorem/Proof/Definition) Y in Theorem Prover Z”, we typically mean

1. Model X wrote Z code that compiles
2. Model X did so with a reasonable degree of autonomy
3. The end product of the Z code written by Model X corresponds semantically to the contents of Y

An autoformalisation could either involve **translating a mathematical endeavour** from natural language to Z or **performing the entire endeavour** in Z. Often, some combination of these.

The benefit of the first mode is that it removes the requirement of learning Z to avail its benefits.

The benefit of the second mode is that it gives us confidence in the machine output.

Autoformalisation

Autoformalisation is when a machine writes up a proof in a formal theorem prover.

When we say “Model X has autoformalised (Theorem/Proof/Definition) Y in Theorem Prover Z”, we typically mean

1. Model X wrote Z code that compiles
2. Model X did so with a reasonable degree of autonomy
3. The end product of the Z code written by Model X corresponds semantically to the contents of Y

An autoformalisation could either involve **translating a mathematical endeavour** from natural language to Z or **performing the entire endeavour** in Z. Often, some combination of these.

The benefit of the first mode is that it removes the requirement of learning Z to avail its benefits.

The benefit of the second mode is that it gives us confidence in the machine output.

MODULO SEMANTIC CORRESPONDENCE

Autoformalisation

Autoformalisation is when a machine writes up a proof in a formal theorem prover.

When we say “Model X has autoformalised (Theorem/Proof/Definition) Y in Theorem Prover Z”, we typically mean

1. Model X wrote Z code that compiles
2. Model X did so with a reasonable degree of autonomy
3. The end product of the Z code written by Model X corresponds semantically to the contents of Y

An autoformalisation could either involve **translating a mathematical endeavour** from natural language to Z or **performing the entire endeavour** in Z. Often, some combination of these.

The benefit of the first mode is that it removes the requirement of learning Z to avail its benefits.

The benefit of the second mode is that it gives us confidence in the machine output.

MODULO SEMANTIC CORRESPONDENCE (and correctness of Z)

Models that can Formalise

Models that can Formalise

A number of models and tools have been created solely for the purpose of autoformalisation.

Models that can Formalise

A number of models and tools have been created solely for the purpose of autoformalisation.

- **Évariste:** Early model developed at Meta AI

Models that can Formalise

A number of models and tools have been created solely for the purpose of autoformalisation.

- **Évariste:** Early model developed at Meta AI
- **AlphaProof:** Achieved silver-level performance at the IMO

Models that can Formalise

A number of models and tools have been created solely for the purpose of autoformalisation.

- **Évariste:** Early model developed at Meta AI
- **AlphaProof:** Achieved silver-level performance at the IMO
- **Aristotle:** First dedicated, user-facing Lean 4 autoformalisation model

Models that can Formalise

A number of models and tools have been created solely for the purpose of autoformalisation.

- **Évariste:** Early model developed at Meta AI
- **AlphaProof:** Achieved silver-level performance at the IMO
- **Aristotle:** First dedicated, user-facing Lean 4 autoformalisation model
- **Gauss:** Demonstrated that models can formalise very large volumes of nontrivial mathematics

Models that can Formalise

A number of models and tools have been created solely for the purpose of autoformalisation.

- **Évariste:** Early model developed at Meta AI
- **AlphaProof:** Achieved silver-level performance at the IMO
- **Aristotle:** First dedicated, user-facing Lean 4 autoformalisation model
- **Gauss:** Demonstrated that models can formalise very large volumes of nontrivial mathematics
- **AxiomProver:** Assisted in solving (and subsequently formalising) a number of open problems

Models that can Formalise

A number of models and tools have been created solely for the purpose of autoformalisation.

- **Évariste:** Early model developed at Meta AI
- **AlphaProof:** Achieved silver-level performance at the IMO
- **Aristotle:** First dedicated, user-facing Lean 4 autoformalisation model
- **Gauss:** Demonstrated that models can formalise very large volumes of nontrivial mathematics
- **AxiomProver:** Assisted in solving (and subsequently formalising) a number of open problems
- AlephProver, Kimina Prover, Seed-Prover, AxProver, etc etc etc

Models that can Formalise

A number of models and tools have been created solely for the purpose of autoformalisation.

- **Évariste:** Early model developed at Meta AI
- **AlphaProof:** Achieved silver-level performance at the IMO
- **Aristotle:** First dedicated, user-facing Lean 4 autoformalisation model
- **Gauss:** Demonstrated that models can formalise very large volumes of nontrivial mathematics
- **AxiomProver:** Assisted in solving (and subsequently formalising) a number of open problems
- AlephProver, Kimina Prover, Seed-Prover, AxProver, etc etc etc

Nowadays, most off-the-shelf LLMs are extremely adept at formalisation.

Models that can Formalise

A number of models and tools have been created solely for the purpose of autoformalisation.

- **Évariste:** Early model developed at Meta AI
- **AlphaProof:** Achieved silver-level performance at the IMO
- **Aristotle:** First dedicated, user-facing Lean 4 autoformalisation model
- **Gauss:** Demonstrated that models can formalise very large volumes of nontrivial mathematics
- **AxiomProver:** Assisted in solving (and subsequently formalising) a number of open problems
- AlephProver, Kimina Prover, Seed-Prover, AxProver, etc etc etc

Nowadays, most off-the-shelf LLMs are extremely adept at formalisation.

It is very easy (easy, not necessarily cheap) to point them to a paper and get them to formalise it.

Models that can Formalise

A number of models and tools have been created solely for the purpose of autoformalisation.

- **Évariste:** Early model developed at Meta AI
- **AlphaProof:** Achieved silver-level performance at the IMO
- **Aristotle:** First dedicated, user-facing Lean 4 autoformalisation model
- **Gauss:** Demonstrated that models can formalise very large volumes of nontrivial mathematics
- **AxiomProver:** Assisted in solving (and subsequently formalising) a number of open problems
- AlephProver, Kimina Prover, Seed-Prover, AxProver, etc etc etc

Nowadays, most off-the-shelf LLMs are extremely adept at formalisation.

It is very easy (easy, not necessarily cheap) to point them to a paper and get them to formalise it.

Skills, harnesses and agentic workflows make them even better.

AI and the Formal Turn

Formalisation Before AI

Formalisation Before AI

In the pre-AI era, formalisation was synonymous with building formal infrastructure.

Formalisation Before AI

In the pre-AI era, formalisation was synonymous with building formal infrastructure.

For instance, Gonthier et al. “**developed a comprehensive set of reusable libraries of formalized mathematics**, including results in finite group theory, linear algebra, Galois theory, and the theories of the real and complex algebraic numbers” to support their formalisation of Feit–Thompson [4].

Formalisation Before AI

In the pre-AI era, formalisation was synonymous with building formal infrastructure.

For instance, Gonthier et al. "**developed a comprehensive set of reusable libraries of formalized mathematics**, including results in finite group theory, linear algebra, Galois theory, and the theories of the real and complex algebraic numbers" to support their formalisation of Feit–Thompson [4].

This process typically led to improved mathematical understanding.

Formalisation Before AI

In the pre-AI era, formalisation was synonymous with building formal infrastructure.

For instance, Gonthier et al. "**developed a comprehensive set of reusable libraries of formalized mathematics**, including results in finite group theory, linear algebra, Galois theory, and the theories of the real and complex algebraic numbers" to support their formalisation of Feit–Thompson [4].

This process typically led to improved mathematical understanding.

For instance, halfway through the Liquid Tensor Experiment, Scholze wrote, "[...] half a year ago, **I did not understand why the argument worked** [...]. But during the formalization, **a significant amount of convex geometry had to be formalized** (in order to prove a well-known lemma known as Gordan's lemma), and this made me realize that actually **the key thing happening is a reduction** from a non-convex problem over the reals to a convex problem over the integers." [8]

Formalisation Before AI

In the pre-AI era, formalisation was synonymous with building formal infrastructure.

For instance, Gonthier et al. "**developed a comprehensive set of reusable libraries of formalized mathematics**, including results in finite group theory, linear algebra, Galois theory, and the theories of the real and complex algebraic numbers" to support their formalisation of Feit–Thompson [4].

This process typically led to improved mathematical understanding.

For instance, halfway through the Liquid Tensor Experiment, Scholze wrote, "[...] half a year ago, **I did not understand why the argument worked** [...]. But during the formalization, **a significant amount of convex geometry had to be formalized** (in order to prove a well-known lemma known as Gordan's lemma), and this made me realize that actually **the key thing happening is a reduction** from a non-convex problem over the reals to a convex problem over the integers." [8]

In fact, my own work on formalising sphere packing was motivated purely by my desire to learn new mathematics, by both writing Lean code and collaborating with subject experts.

The Two Families of Formalisation Projects

The Two Families of Formalisation Projects

All our examples thus far fall into two broad families.

The Two Families of Formalisation Projects

All our examples thus far fall into two broad families.

The first family is projects that were motivated purely by **building formal infrastructure** and **improving mathematical understanding**.

The Two Families of Formalisation Projects

All our examples thus far fall into two broad families.

The first family is projects that were motivated purely by **building formal infrastructure** and **improving mathematical understanding**. Examples include PNT (Isabelle - elementary and HOL - analytic), PNT+ (Lean - ongoing), Feit–Thompson (Rocq), Perfectoid Spaces (Lean), Sphere Packing (Lean - ongoing), Buzzard's FLT (Lean - ongoing).

The Two Families of Formalisation Projects

All our examples thus far fall into two broad families.

The first family is projects that were motivated purely by **building formal infrastructure** and **improving mathematical understanding**. Examples include PNT (Isabelle - elementary and HOL - analytic), PNT+ (Lean - ongoing), Feit–Thompson (Rocq), Perfectoid Spaces (Lean), Sphere Packing (Lean - ongoing), Buzzard's FLT (Lean - ongoing).

The second family is projects that were motivated by **correctness**.

The Two Families of Formalisation Projects

All our examples thus far fall into two broad families.

The first family is projects that were motivated purely by **building formal infrastructure** and **improving mathematical understanding**. Examples include PNT (Isabelle - elementary and HOL - analytic), PNT+ (Lean - ongoing), Feit–Thompson (Rocq), Perfectoid Spaces (Lean), Sphere Packing (Lean - ongoing), Buzzard's FLT (Lean - ongoing).

The second family is projects that were motivated by **correctness**. Examples include Flyspeck (HOL Light), Four Colour (Rocq), Liquid Tensor (Lean), Unit Distance (Lean), Navier–Stokes (Lean).

The Two Families of Formalisation Projects

All our examples thus far fall into two broad families.

The first family is projects that were motivated purely by **building formal infrastructure** and **improving mathematical understanding**. Examples include PNT (Isabelle - elementary and HOL - analytic), PNT+ (Lean - ongoing), Feit–Thompson (Rocq), Perfectoid Spaces (Lean), Sphere Packing (Lean - ongoing), Buzzard's FLT (Lean - ongoing).

The second family is projects that were motivated by **correctness**. Examples include Flyspeck (HOL Light), Four Colour (Rocq), Liquid Tensor (Lean), Unit Distance (Lean), Navier–Stokes (Lean).

Projects from both families are supported by extensive libraries of formal mathematics, and have, in turn, fed back into them.

The Two Families of Formalisation Projects

All our examples thus far fall into two broad families.

The first family is projects that were motivated purely by **building formal infrastructure** and **improving mathematical understanding**. Examples include PNT (Isabelle - elementary and HOL - analytic), PNT+ (Lean - ongoing), Feit–Thompson (Rocq), Perfectoid Spaces (Lean), Sphere Packing (Lean - ongoing), Buzzard's FLT (Lean - ongoing).

The second family is projects that were motivated by **correctness**. Examples include Flyspeck (HOL Light), Four Colour (Rocq), Liquid Tensor (Lean), Unit Distance (Lean), Navier–Stokes (Lean).

Projects from both families are supported by extensive libraries of formal mathematics, and have, in turn, fed back into them.

But a more conscious effort is needed for projects from the second family to do this than the first.

The Two Families of Formalisation Projects

All our examples thus far fall into two broad families.

The first family is projects that were motivated purely by **building formal infrastructure** and **improving mathematical understanding**. Examples include PNT (Isabelle - elementary and HOL - analytic), PNT+ (Lean - ongoing), Feit–Thompson (Rocq), Perfectoid Spaces (Lean), Sphere Packing (Lean - ongoing), Buzzard's FLT (Lean - ongoing).

The second family is projects that were motivated by **correctness**. Examples include Flyspeck (HOL Light), Four Colour (Rocq), Liquid Tensor (Lean), Unit Distance (Lean), Navier–Stokes (Lean).

Projects from both families are supported by extensive libraries of formal mathematics, and have, in turn, fed back into them.

But a more conscious effort is needed for projects from the second family to do this than the first.

Pre-AI, projects from the second family, even if not strictly motivated by infrastructure-building alone, would do this extensively.

The Two Families of Formalisation Projects

All our examples thus far fall into two broad families.

The first family is projects that were motivated purely by **building formal infrastructure** and **improving mathematical understanding**. Examples include PNT (Isabelle - elementary and HOL - analytic), PNT+ (Lean - ongoing), Feit–Thompson (Rocq), Perfectoid Spaces (Lean), Sphere Packing (Lean - ongoing), Buzzard's FLT (Lean - ongoing).

The second family is projects that were motivated by **correctness**. Examples include Flyspeck (HOL Light), Four Colour (Rocq), Liquid Tensor (Lean), Unit Distance (Lean), Navier–Stokes (Lean).

Projects from both families are supported by extensive libraries of formal mathematics, and have, in turn, fed back into them.

But a more conscious effort is needed for projects from the second family to do this than the first.

Pre-AI, projects from the second family, even if not strictly motivated by infrastructure-building alone, would do this extensively. Examples include the theory of Euclidean spaces developed in Flyspeck and the convex geometry developed in Liquid Tensor.

The Two Families of Formalisation Projects

All our examples thus far fall into two broad families.

The first family is projects that were motivated purely by **building formal infrastructure** and **improving mathematical understanding**. Examples include PNT (Isabelle - elementary and HOL - analytic), PNT+ (Lean - ongoing), Feit–Thompson (Rocq), Perfectoid Spaces (Lean), Sphere Packing (Lean - ongoing), Buzzard's FLT (Lean - ongoing).

The second family is projects that were motivated by **correctness**. Examples include Flyspeck (HOL Light), Four Colour (Rocq), Liquid Tensor (Lean), Unit Distance (Lean), Navier–Stokes (Lean).

Projects from both families are supported by extensive libraries of formal mathematics, and have, in turn, fed back into them.

But a more conscious effort is needed for projects from the second family to do this than the first.

Pre-AI, projects from the second family, even if not strictly motivated by infrastructure-building alone, would do this extensively. Examples include the theory of Euclidean spaces developed in Flyspeck and the convex geometry developed in Liquid Tensor.

In the age of AI, this is happening less and less.

Actually, there are 3 families

Actually, there are 3 families

There is a third family of formalisations, unique to the era of AI:

Actually, there are 3 families

There is a third family of formalisations, unique to the era of AI: **projects designed to benchmark or showcase models**.

Actually, there are 3 families

There is a third family of formalisations, unique to the era of AI: **projects designed to benchmark or showcase models**.

Some projects showcase the ability to **formalise**.

Actually, there are 3 families

There is a third family of formalisations, unique to the era of AI: **projects designed to benchmark or showcase models**.

Some projects showcase the ability to **formalise**. Examples include Math Inc's Strong PNT (Lean), Math Inc's Sphere Packing (Lean), Meta's Atlas (Lean) and Anthropic's FLT (Lean).

Actually, there are 3 families

There is a third family of formalisations, unique to the era of AI: **projects designed to benchmark or showcase models**.

Some projects showcase the ability to **formalise**. Examples include Math Inc's Strong PNT (Lean), Math Inc's Sphere Packing (Lean), Meta's Atlas (Lean) and Anthropic's FLT (Lean).

Others showcase the ability to **reason**, where the formal medium is used for standardisation and trust.

Actually, there are 3 families

There is a third family of formalisations, unique to the era of AI: **projects designed to benchmark or showcase models**.

Some projects showcase the ability to **formalise**. Examples include Math Inc's Strong PNT (Lean), Math Inc's Sphere Packing (Lean), Meta's Atlas (Lean) and Anthropic's FLT (Lean).

Others showcase the ability to **reason**, where the formal medium is used for standardisation and trust. Examples include formalisations of IMO problems, Putnam problems, and other benchmarks written in Lean.

Actually, there are 3 families

There is a third family of formalisations, unique to the era of AI: **projects designed to benchmark or showcase models**.

Some projects showcase the ability to **formalise**. Examples include Math Inc's Strong PNT (Lean), Math Inc's Sphere Packing (Lean), Meta's Atlas (Lean) and Anthropic's FLT (Lean).

Others showcase the ability to **reason**, where the formal medium is used for standardisation and trust. Examples include formalisations of IMO problems, Putnam problems, and other benchmarks written in Lean.

These formalisations offer much more value to the AI community than to the mathematical community (formal or general).

The Two Modes of Formalisation

The Two Modes of Formalisation

Having made a distinction in motivation, we now make a distinction in content.

The Two Modes of Formalisation

Having made a distinction in motivation, we now make a distinction in content. The contents of this slide are motivated largely by talks given by Alex Kontorovich at the *AI for Maths and Open Science* event at the INI and the ICM and a talk by Terence Tao at the ICM.

The Two Modes of Formalisation

Having made a distinction in motivation, we now make a distinction in content. The contents of this slide are motivated largely by talks given by Alex Kontorovich at the *AI for Maths and Open Science* event at the INI and the ICM and a talk by Terence Tao at the ICM.

A "**Formal Verification** of (Theorem/Proof/Definition) Y in the Theorem Prover Z" will refer to

The Two Modes of Formalisation

Having made a distinction in motivation, we now make a distinction in content. The contents of this slide are motivated largely by talks given by Alex Kontorovich at the *AI for Maths and Open Science* event at the INI and the ICM and a talk by Terence Tao at the ICM.

A "**Formal Verification** of (Theorem/Proof/Definition) Y in the Theorem Prover Z" will refer to

- a fully compiling artefact expressed in Z

The Two Modes of Formalisation

Having made a distinction in motivation, we now make a distinction in content. The contents of this slide are motivated largely by talks given by Alex Kontorovich at the *AI for Maths and Open Science* event at the INI and the ICM and a talk by Terence Tao at the ICM.

A "**Formal Verification** of (Theorem/Proof/Definition) Y in the Theorem Prover Z" will refer to

- a fully compiling artefact expressed in Z
- whose main result corresponds semantically to Y.

The Two Modes of Formalisation

Having made a distinction in motivation, we now make a distinction in content. The contents of this slide are motivated largely by talks given by Alex Kontorovich at the *AI for Maths and Open Science* event at the INI and the ICM and a talk by Terence Tao at the ICM.

A "**Formal Verification** of (Theorem/Proof/Definition) Y in the Theorem Prover Z" will refer to

- a fully compiling artefact expressed in Z
- whose main result corresponds semantically to Y.

A "**Canonisation** of (Theorem/Proof/Definition) Y in the Theorem Prover Z" will refer to

The Two Modes of Formalisation

Having made a distinction in motivation, we now make a distinction in content. The contents of this slide are motivated largely by talks given by Alex Kontorovich at the *AI for Maths and Open Science* event at the INI and the ICM and a talk by Terence Tao at the ICM.

A "**Formal Verification** of (Theorem/Proof/Definition) Y in the Theorem Prover Z" will refer to

- a fully compiling artefact expressed in Z
- whose main result corresponds semantically to Y.

A "**Canonisation** of (Theorem/Proof/Definition) Y in the Theorem Prover Z" will refer to

- a formal verification of Y in Z

The Two Modes of Formalisation

Having made a distinction in motivation, we now make a distinction in content. The contents of this slide are motivated largely by talks given by Alex Kontorovich at the *AI for Maths and Open Science* event at the INI and the ICM and a talk by Terence Tao at the ICM.

A "**Formal Verification** of (Theorem/Proof/Definition) Y in the Theorem Prover Z" will refer to

- a fully compiling artefact expressed in Z
- whose main result corresponds semantically to Y.

A "**Canonisation** of (Theorem/Proof/Definition) Y in the Theorem Prover Z" will refer to

- a formal verification of Y in Z
- where both the main result and the intermediate artefacts are designed to be reusable, maintainable, general, flexible, extensible... (in other words, designed as per the principles of a library).

The Two Modes of Formalisation

Having made a distinction in motivation, we now make a distinction in content. The contents of this slide are motivated largely by talks given by Alex Kontorovich at the *AI for Maths and Open Science* event at the INI and the ICM and a talk by Terence Tao at the ICM.

A "**Formal Verification** of (Theorem/Proof/Definition) Y in the Theorem Prover Z" will refer to

- a fully compiling artefact expressed in Z
- whose main result corresponds semantically to Y.

A "**Canonisation** of (Theorem/Proof/Definition) Y in the Theorem Prover Z" will refer to

- a formal verification of Y in Z
- where both the main result and the intermediate artefacts are designed to be reusable, maintainable, general, flexible, extensible... (in other words, designed as per the principles of a library).

A "**Formalisation** of (Theorem/Proof/Definition) Y in the Theorem Prover Z" will refer to either a formal verification or a canonisation.

The Value of Each Mode

The Value of Each Mode

I think the value of each mode comes from the **alignment of mode and objective**.

The Value of Each Mode

I think the value of each mode comes from the **alignment of mode and objective**.

In an age where generating AI slop is easier than ever, formal verification offers a means of ‘trusting’ new results.

The Value of Each Mode

I think the value of each mode comes from the **alignment of mode and objective**.

In an age where generating AI slop is easier than ever, formal verification offers a means of ‘trusting’ new results.

One significant threat to formal mathematics is the conflation of motivations: over-hyping a formal verification as a canonisation undermines the value of canonisation, as does glorifying correctness as the only significant outcome of a canonisation.

The Value of Each Mode

I think the value of each mode comes from the **alignment of mode and objective**.

In an age where generating AI slop is easier than ever, formal verification offers a means of 'trusting' new results.

One significant threat to formal mathematics is the conflation of motivations: over-hyping a formal verification as a canonisation undermines the value of canonisation, as does glorifying correctness as the only significant outcome of a canonisation.

At the same time, canonisation is slow: expecting every formal verification to be canonised is, in my view, unreasonable.

The Value of Each Mode

I think the value of each mode comes from the **alignment of mode and objective**.

In an age where generating AI slop is easier than ever, formal verification offers a means of 'trusting' new results.

One significant threat to formal mathematics is the conflation of motivations: over-hyping a formal verification as a canonisation undermines the value of canonisation, as does glorifying correctness as the only significant outcome of a canonisation.

At the same time, canonisation is slow: expecting every formal verification to be canonised is, in my view, unreasonable.

Both modes are valuable and need to be encouraged, but must be recognised as addressing different needs.

Canonisation, Trust and Understanding

The Need for Canonisation in Formal Mathematics

The Need for Canonisation in Formal Mathematics

Virtually every formalisation of mathematical significance depends on Mathlib. Many depend on, or have used code from, PNT+, FLT, even Sphere Packing.

The Need for Canonisation in Formal Mathematics

Virtually every formalisation of mathematical significance depends on Mathlib. Many depend on, or have used code from, PNT+, FLT, even Sphere Packing.

Formalisations building on bad formalisations eventually become unintelligible.

The Need for Canonisation in Formal Mathematics

Virtually every formalisation of mathematical significance depends on Mathlib. Many depend on, or have used code from, PNT+, FLT, even Sphere Packing.

Formalisations building on bad formalisations eventually become unintelligible.

Even though tokens are getting cheaper, the frontier is being pushed faster than canonised code is being built.

The Need for Canonisation in Formal Mathematics

Virtually every formalisation of mathematical significance depends on Mathlib. Many depend on, or have used code from, PNT+, FLT, even Sphere Packing.

Formalisations building on bad formalisations eventually become unintelligible.

Even though tokens are getting cheaper, the frontier is being pushed faster than canonised code is being built.

Besides, it is cheaper, faster, and more effective to canonise from the outset than to attempt to canonise poor code.

The Need for Canonisation in Formal Mathematics

Virtually every formalisation of mathematical significance depends on Mathlib. Many depend on, or have used code from, PNT+, FLT, even Sphere Packing.

Formalisations building on bad formalisations eventually become unintelligible.

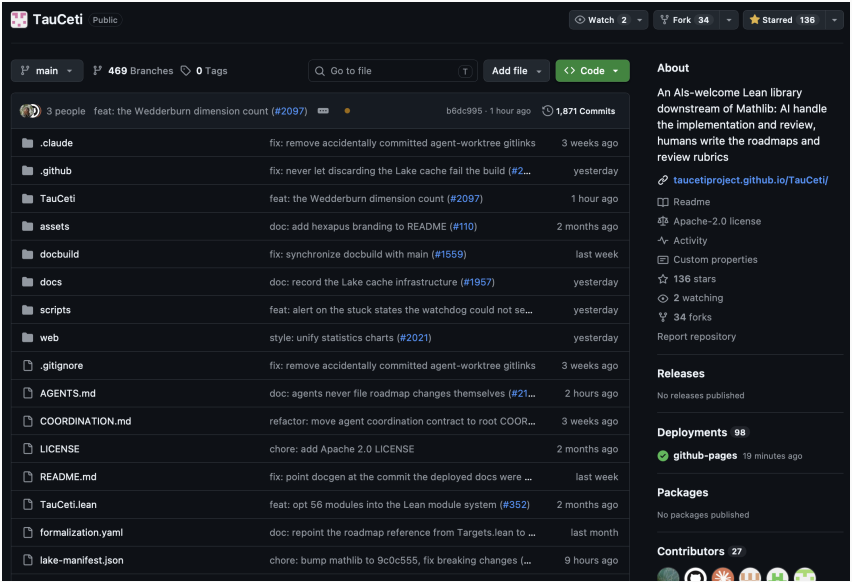
Even though tokens are getting cheaper, the frontier is being pushed faster than canonised code is being built.

Besides, it is cheaper, faster, and more effective to canonise from the outset than to attempt to canonise poor code.

Canonisation is essential for verification to scale.

Experiments in Autocanonisation

Experiments in Autocanonisation



Trust and Understanding[1]

[1] This slide is inspired by a talk yesterday at CMU's Graduate Student and Postdoc Seminar by my friend Yongxi (Aaron) Lin.

Trust and Understanding[1]

Understanding was once considered one of the most mathematically valuable benefits of formalisation.

[1] This slide is inspired by a talk yesterday at CMU's Graduate Student and Postdoc Seminar by my friend Yongxi (Aaron) Lin.

Trust and Understanding[1]

Understanding was once considered one of the most mathematically valuable benefits of formalisation.

Canonisation, at least with human architects, still largely achieves this outcome.

[1] This slide is inspired by a talk yesterday at CMU's Graduate Student and Postdoc Seminar by my friend Yongxi (Aaron) Lin.

Trust and Understanding[1]

Understanding was once considered one of the most mathematically valuable benefits of formalisation.

Canonisation, at least with human architects, still largely achieves this outcome.

However, as AI becomes more important to mathematics, the focus is shifting to **trust** as the chief benefit of formalisation.

[1] This slide is inspired by a talk yesterday at CMU's Graduate Student and Postdoc Seminar by my friend Yongxi (Aaron) Lin.

Trust and Understanding[1]

Understanding was once considered one of the most mathematically valuable benefits of formalisation.

Canonisation, at least with human architects, still largely achieves this outcome.

However, as AI becomes more important to mathematics, the focus is shifting to **trust** as the chief benefit of formalisation.

Given how easy it is to generate tokens (code and natural language alike), we are entering an era where we verify before even *attempting* to understand.

[1] This slide is inspired by a talk yesterday at CMU's Graduate Student and Postdoc Seminar by my friend Yongxi (Aaron) Lin.

Trust and Understanding[1]

Understanding was once considered one of the most mathematically valuable benefits of formalisation.

Canonisation, at least with human architects, still largely achieves this outcome.

However, as AI becomes more important to mathematics, the focus is shifting to **trust** as the chief benefit of formalisation.

Given how easy it is to generate tokens (code and natural language alike), we are entering an era where we verify before even *attempting* to understand.

Before autoformalisation, we would **develop trust through understanding**.

[1] This slide is inspired by a talk yesterday at CMU's Graduate Student and Postdoc Seminar by my friend Yongxi (Aaron) Lin.

Trust and Understanding[1]

Understanding was once considered one of the most mathematically valuable benefits of formalisation.

Canonisation, at least with human architects, still largely achieves this outcome.

However, as AI becomes more important to mathematics, the focus is shifting to **trust** as the chief benefit of formalisation.

Given how easy it is to generate tokens (code and natural language alike), we are entering an era where we verify before even *attempting* to understand.

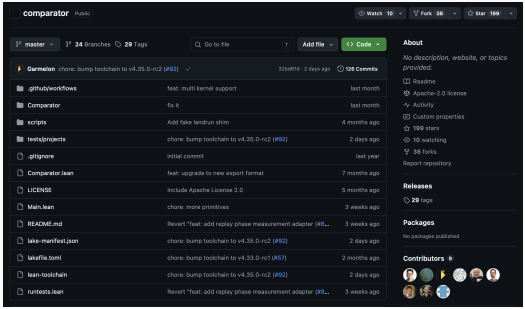
Before autoformalisation, we would **develop trust through understanding**.

Autoformalisation is making it possible to **trust *without* understanding**.

[1] This slide is inspired by a talk yesterday at CMU's Graduate Student and Postdoc Seminar by my friend Yongxi (Aaron) Lin.

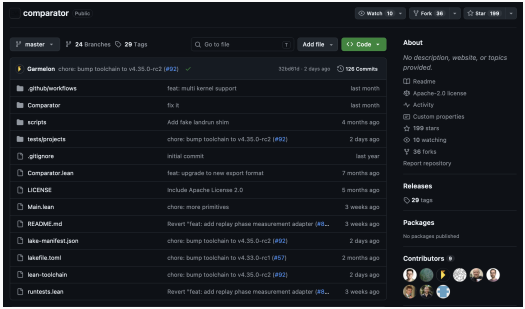
Trusting Proof Artefacts

Trusting Proof Artefacts

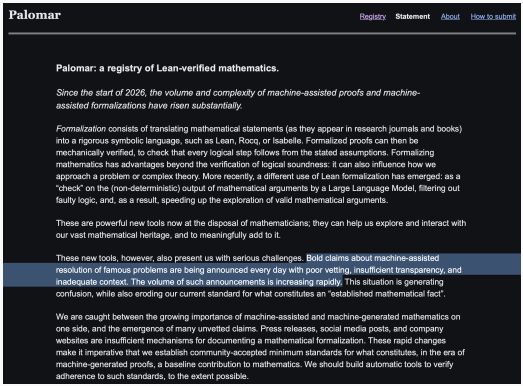


(a) `comparator`: a fidelity (formal statement-formal proof) and correctness checker

Trusting Proof Artefacts



(a) `comparator`: a fidelity (formal statement-formal proof) and correctness checker



(b) Palomar: a registry of Lean-verified mathematics, encouraging transparent reporting and enforcing fidelity

The Macrocosmic View

The Macrocosmic View

Formal	General

The Macrocosmic View

	Formal	General
Is it still worth doing?	Yes	

The Macrocosmic View

	Formal	General
Is it still worth doing?	Yes	
Why?	Building understanding, improving infrastructure, developing new theories...	

The Macrocosmic View

	Formal	General
Is it still worth doing?	Yes	
Why?	Building understanding, improving infrastructure, developing new theories...	
How do we assign credit?	Self-reporting (eg. on Palomar), Git history	

The Macrocosmic View

	Formal	General
Is it still worth doing?	Yes	
Why?	Building understanding, improving infrastructure, developing new theories...	
How do we assign credit?	Self-reporting (eg. on Palomar), Git history	
What responsibilities remain human?	Semantic correspondence, responding to feedback and review when upstreaming	

The Macrocosmic View

	Formal	General
Is it still worth doing?	Yes	
Why?	Building understanding, improving infrastructure, developing new theories...	
How do we assign credit?	Self-reporting (eg. on Palomar), Git history	
What responsibilities remain human?	Semantic correspondence, responding to feedback and review when upstreaming	
What do we teach?	How to make good design decisions, interpret the mathematical content of code, and use AI with intent and deliberation	

Questions?

References i

[1] J. Avigad, K. Donnelly, D. Gray, and P. Raff.
A formally verified proof of the prime number theorem.
ACM Trans. Comput. Logic, 9(1):2–es, Dec. 2007.

[2] M. Eberl.
Nine Chapters of Analytic Number Theory in Isabelle/HOL.
In J. Harrison, J. O'Leary, and A. Tolmach, editors, *10th International Conference on Interactive Theorem Proving (ITP 2019)*, volume 141 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 16:1–16:19, Dagstuhl, Germany, 2019. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[3] G. Gonthier.
A computer-checked proof of the Four Color Theorem.
Technical report, Inria, Mar. 2023.

[4] G. Gonthier, A. Asperti, J. Avigad, Y. Bertot, C. Cohen, F. Garillot, S. Le Roux, A. Mahboubi, R. O'Connor, S. O. Biha, I. Pasca, L. Rideau, A. Solovyev, E. Tassi, and L. Théry.
A Machine-Checked Proof of the Odd Order Theorem.
In *Proceedings of the 4th International Conference on Interactive Theorem Proving*, ITP'13, pages 163–179, Berlin, Heidelberg, 2013. Springer-Verlag.

[5] T. C. Hales.
The Jordan Curve Theorem, Formally and Informally.
The American Mathematical Monthly, 114(10):882–894, 2007.

[6] T. C. Hales, M. Adams, G. Bauer, T. D. Dang, J. Harrison, L. T. Hoang, C. Kaliszyk, V. Magron, S. McLaughlin, T. T. Nguyen, et al.
A Formal Proof of the Kepler Conjecture.
Forum of Mathematics, Pi, 5:e2, 2017.

[7] J. Harrison.
Formalizing an Analytic Proof of the Prime Number Theorem.
Journal of Automated Reasoning, 43(3):243–261, Oct 2009.

[8] P. Scholze.
Half a year of the Liquid Tensor Experiment: Amazing developments.
https://xenaproject.wordpress.com/2021/06/05/half-a-year-of-the-liquid-tensor-experiment-amazing-developments/, June 2021.
Guest post on the *Xena Project* blog.

[9] The Univalent Foundations Program.
Homotopy Type Theory: Univalent Foundations of Mathematics.
Institute for Advanced Study, Princeton, NJ, 2013.
https://homotopytypetheory.org/book.