Formalizing Scarf, Brouwer, and Nash in Lean

Yuwei Lyu [1] **Kai Li** [1]

Abstract

We formalize in Lean 4 a complete combinatorial route from Scarf's theorem to Brouwer's fixed point theorem and to the existence of mixed Nash equilibria in finite games. The development follows Ivanov's indexed-order formulation of Scarf's theorem, formalizes the room–door incidence structure and parity argument, instantiates the theorem on finite grids of the standard simplex, and carries out the compactness and continuity argument needed to obtain a fixed point. We then extend the result to finite products of simplices by an explicit embedding–projection construction and use this product theorem to prove mixed Nash equilibrium existence via the Nash map. As a secondary by-product, we derive BrouwerBench, a preliminary 80-item Lean-grounded benchmark for probing proof-structure understanding within this single formal development.

1. Introduction

Brouwer's fixed point theorem is a fundamental existence result in topology (Brouwer, 1911) and a standard tool in analysis, economics, and game theory (Border, 1985). A central application is Nash's theorem for finite games (Nash, 1950): mixed strategy profiles form a finite product of simplices, and an appropriately defined continuous self-map has a fixed point exactly at a mixed Nash equilibrium. For formalization, it is useful not only to invoke Brouwer's theorem as a black-box topological principle, but also to expose the finite combinatorial structure from which one can derive the fixed-point result. This makes the proof pipeline modular: the finite parity argument, the limiting construction, the product-of-simplices reduction, and the game-theoretic endpoint can be checked as separate formal components.

This paper formalizes such a route in Lean 4. We start from Ivanov's indexed-order formulation of Scarf's combinatorial theorem (Scarf, 1982; Ivanov, 2019), where a finite set is equipped with indexed linear orders and a coloring. The proof builds a finite incidence structure of dominant sets, cells, rooms, and doors: outside doors are incident to one room, internal doors are incident to two rooms, and the resulting parity argument produces a colorful room. We then instantiate this theorem on increasingly fine grids of the standard simplex. The colorful rooms provide discrete approximations, while compactness, continuity, and vanishing-diameter estimates yield a Brouwer fixed point. Finally, we extend the standard-simplex theorem to finite products of simplices by an explicit embedding–projection construction and apply the product theorem to the Nash map for finite games.

The formalization treats this route as a sequence of reusable proof components rather than only as a final fixed-point theorem. In particular, the room–door parity argument, the dominance estimates on simplex grids, and the embedding–projection reduction for products of simplices are exposed as named Lean definitions and lemmas. This makes the development inspectable as a structured proof artifact and allows the final Nash theorem to be traced back to finite combinatorial data.

The development builds on Mathlib (The mathlib Community, 2020) and is written in Lean 4 (Moura & Ullrich, 2021). Mathlib provides finite types, finite sets, real analysis, compactness, continuity, finite-dimensional spaces, and standard simplices. The development consists of 4,768 lines of Lean across five files, covering the Scarf combinatorics, the standard-simplex Brouwer proof, the product-simplex reduction, the Nash endpoint, and supporting simplex infrastructure; Appendix B gives the file-level breakdown. The paper-specific Lean artifact is publicly available.[1]

The body of the paper presents the proof pipeline and main formalization choices. Appendices C–F summarize the

[1]Department of Mathematics, Xiamen University Malaysia, Sepang, Selangor, Malaysia. Correspondence to: Yuwei Lyu <Lyuyuwei@gmail.com>, Kai Li <donk28510@gmail.com>.

Proceedings of the 43rd International Conference on Machine Learning, Seoul, South Korea. PMLR 306, 2026. Copyright 2026 by the author(s).

[1]Code and benchmark materials are available in the camera-ready artifact; continued development is maintained in the upstream repository.

main proof components, and Appendix G records representative Lean interfaces. The clickable symbol links mathematical statements to their full definitions and proofs in the source files. Since each stage is represented by named Lean definitions and lemmas, the development also yields local proof-understanding tasks. We package these tasks as BrouwerBench, a small pilot benchmark intended to probe whether models can answer proof-role questions inside this particular Lean-grounded development. BrouwerBench should therefore be read as a secondary artifact of the formalization, not as a broad benchmark for general mathematical reasoning or Lean theorem proving.

1.1. Contributions

Our main contribution is a Lean-checked proof pipeline from Scarf's combinatorial theorem to Brouwer's fixed point theorem and mixed Nash equilibrium existence. Specifically, we formalize Ivanov's indexed-order version of Scarf's theorem, including dominance, rooms, doors, the two-room property, and the parity argument for colorful rooms; instantiate this theorem on finite simplex grids to derive Brouwer's theorem on the standard simplex by dominance estimates, compactness, and continuity; extend the result to finite products of simplices using an explicit embedding–projection construction; and apply the product theorem to prove mixed Nash equilibrium existence for finite games. As a secondary by-product, we extract BrouwerBench, an 80-item Lean-grounded pilot benchmark for proof-structure understanding within this formalization.

2. Related Work

Formalized mathematics in Lean. Lean and Mathlib have supported increasingly large formalized mathematics projects, from undergraduate algebra and topology to research-level developments (The mathlib Community, 2020; Moura & Ullrich, 2021). These projects show that modern proof assistants can serve not only as proof checkers, but also as environments for organizing reusable mathematical infrastructure. Our work follows this line, but focuses on a proof route whose structure combines finite combinatorics with analysis: a Scarf-style parity argument is formalized first, then transported to Brouwer's fixed point theorem by approximation and compactness, and finally applied to Nash equilibria.

Fixed-point and equilibrium theorems. Brouwer's fixed point theorem is a classical result with many proofs and applications, including the standard route to the existence of mixed Nash equilibria in finite games (Brouwer, 1911; Nash, 1950; Border, 1985). The emphasis of our development is not only the final fixed-point or equilibrium statement, but the formalized proof pipeline leading to it. We formalize Scarf's combinatorial theorem (Scarf, 1982; Ivanov, 2019), derive Brouwer's theorem on the standard simplex, extend it to finite products of simplices, and then obtain Nash equilibria through the Nash map. This makes the finite room–door incidence argument, dominance estimates on simplex grids, limiting construction, and product-simplex reduction explicit as reusable Lean components, rather than treating Brouwer's theorem as a black box.

Autoformalization and benchmarks. Recent work on neural theorem proving and autoformalization studies how language models translate mathematical text into formal statements and proofs, retrieve useful lemmas, and construct or repair proof scripts (Polu & Sutskever, 2020; Zheng et al., 2022). Many benchmarks in this area consist of independent problem instances or theorem statements. BrouwerBench is instead extracted from a single kernel-checked Lean development: each item is tied to a concrete definition, dependency, lemma role, or design decision in the Scarf–Brouwer–Nash pipeline. It is intended as a small pilot probe of proof-structure understanding inside this coherent formalization, rather than as a broad model-ranking benchmark for general theorem proving or mathematical reasoning.

3. Formalizing Scarf's Combinatorial Theorem

This section contains the finite combinatorial part of the development. We follow Ivanov's abstract formulation of Scarf's theorem (Ivanov, 2019), where the input is a finite set equipped with a family of indexed linear orders and a coloring. The proof is organized around three layers. First, dominance turns order-theoretic data into finite cells. Second, rooms and doors provide the incidence structure used for counting. Third, colorings and nearly colorful cells turn this incidence structure into a parity argument producing a colorful room.

3.1. Orders, Dominant Sets, and Cells

We work with a finite type T together with a finite index type I. For each $i \in I$, we are given a linear order $<_i$ on T. The indexed family of orders is the basic object from which dominance is defined. In Lean, this data is bundled as a typeclass , so that the notation for comparisons can depend on the index.

Definition 3.1 (Indexed family of linear orders)**.** A family of linear orders on T indexed by I is a collection $(<_i)_{i \in I}$ such that each $<_i$ is a linear order on T .

Given a nonempty subset $\sigma \subseteq T$ and an index $i \in I$, we write $\min_i \sigma$ for the minimum of σ with respect to $<_i$.

Dominance says that every point of T is controlled by at least one of the orders indexed by C. This is the order-theoretic condition that later forces the relevant finite sets to have the correct cardinalities.

Definition 3.2 (Dominant set)**.** Let C be a nonempty subset of I. A subset σ of T is dominant with respect to C if, for every element y of T, one can find an index i in C such that y is below every element of σ in the order $<_i$.

When σ is nonempty, this is equivalent to saying that no element of T is strictly larger than all the minima $\min_i \sigma$ for i in C.

We adopt the convention that $\emptyset \subset T$ is dominant with respect to every non-empty subset $C \subset I$. The following lemma collects the basic closure and cardinality facts about dominant sets. These facts are used repeatedly when passing between cells, rooms, and doors.

Lemma 3.3 (Basic properties of dominant sets)**.** *If σ is nonempty and dominant with respect to C, then $\sigma = \{\min_i \sigma \mid i \in C\}$, and hence $|\sigma| \leq |C|$. If $\tau \subset \sigma$, then τ is dominant with respect to C, and if $C \subset D$, then σ is dominant with respect to D.*

The preceding lemma allows us to treat dominant pairs (σ, C) as cells of a finite combinatorial object. Rooms are the full-dimensional cells, while doors are their codimension-one boundaries in the counting argument.

Definition 3.4 (Cells, rooms, and doors)**.** A *cell* is a pair (σ, C) where σ is dominant with respect to C. A cell is called a *room* if $|C| = |\sigma|$, and a *door* if $|C| = |\sigma| + 1$.

3.2. Rooms, Doors, and the Two-Room Property

The next step is to make precise how rooms meet along doors. In the formalization, this is captured by the predicate `isDoorof`, which records that a door is obtained from a room either by removing an element from the underlying set or by adding a color. This relation is the local incidence relation used later in the parity count.

Definition 3.5 (Outside and internal doors)**.** A door (τ, D) is called an *outside door* if $\tau = \emptyset$, and an *internal door* otherwise.

The following two lemmas give the degree structure of this incidence relation. Outside doors behave as boundary objects: they have degree one. Internal doors behave as shared faces: they have degree two.

Lemma 3.6 (Outside-door property)**.** *Suppose that $|D| = 1$. Then $(\varnothing, D)$ is an outside door and is a door of exactly one room. Every outside door has this form.*

Lemma 3.7 (Two-room property)**.** *Every internal door belongs to exactly two rooms.*

Thus the rooms and doors form a finite graph-like incidence structure: starting from an outside door, one enters a unique room, and every time one crosses an internal door there is a unique other room on the opposite side. This is the combinatorial shape behind the later path-following and parity argument.

The proof of the two-room property is the most involved local part of the Scarf formalization. It analyzes an internal door (τ, D) via two complementary constructions: either removing a color from D, or extending τ by a suitable element. These are implemented in Lean via the auxiliary constructions `M_set` and maximal elements `m_element`, which ensure that both constructions yield valid rooms. In the paper, we use the two-room property as a black-box incidence theorem; in Lean, the proof supplies the exact finite-set witnesses needed to make the incidence count well typed.

3.3. Colorings and the Parity Argument

We now add the coloring data. The coloring selects, for each element of T, an index in I. A colorful cell is one whose index set is exactly the image of its underlying set under this coloring. Nearly colorful cells differ from being colorful by exactly one missing color; the missing color is recorded as the type. This typed version is useful because the parity argument counts objects of a fixed type.

Definition 3.8 (Coloring)**.** Fix a coloring $c : T \to I$. A cell (σ, C) is called *colorful* if $C = c(\sigma)$, and *nearly colorful* if $|C \setminus c(\sigma)| = 1$. For a nearly colorful cell (σ, C), the unique element of $C \setminus c(\sigma)$ is called its *type*.

The next lemmas connect the uncolored incidence structure with the coloring data. They identify the boundary terms, describe how colorful rooms contribute nearly colorful doors, and show that nearly colorful rooms have the expected number of nearly colorful doors.

Lemma 3.9 (Outside doors and doors of colorful rooms)**.** *For every $i \in I$ there is exactly one outside door of the type i. Every door of a colorful room is nearly colorful and there is exactly one door of each type among them.*

Lemma 3.10 (Nearly colorful door transition)**.** *Suppose that (τ, D) is a nearly colorful door of a room (σ, C). Then (σ, C) is either colorful, or nearly colorful of the same type as (τ, D).*

Lemma 3.11 (Nearly colorful cardinality properties)**.** *If (σ, C) is a nearly colorful room, then $|c(\sigma)|$ is equal either to $|\sigma|$ or to $|\sigma| - 1$. If (σ, C) is a nearly colorful door, then $c(\sigma) \subset C$.*

Lemma 3.12 (Doors of nearly colorful rooms)**.** *If (σ, C) is a nearly colorful room, then (σ, C) has two nearly colorful doors.*

Together, these lemmas turn the room–door incidence relation into a parity count. Fixing a type i, the unique outside door of type i contributes one boundary term. Internal nearly colorful doors are counted twice because of the two-room property. Nearly colorful rooms also contribute an even number of typed doors. Therefore the remaining contribution comes from colorful rooms, forcing the relevant colorful-room count to be odd.

Theorem 3.13 (Scarf's combinatorial theorem)**.** *For every coloring $c : T \to I$, there exists a colorful room. Moreover, the number of colorful rooms is odd.*

This theorem completes the finite combinatorial core of the formalization. In the next section, the finite set T will be instantiated by a grid in the standard simplex. The colorful rooms supplied by Scarf's theorem then become the discrete approximations from which Brouwer fixed points are obtained by a limiting argument.

4. From Scarf's Theorem to Brouwer's Fixed Point Theorem

This section derives Brouwer's fixed point theorem from the formalized Scarf theorem. The argument has two stages. We first prove Brouwer's theorem for the standard simplex by applying Scarf's theorem to increasingly fine finite grids and then passing to a limit. We then obtain the corresponding theorem for finite products of simplices by reducing the product case to the standard-simplex case through an explicit embedding–projection construction.

4.1. Brouwer's Theorem on the Standard Simplex

Fix a positive integer n, and let Δ^{n-1} be the standard simplex with coordinates indexed by $\mathrm{Fin}\ n$. The goal is to prove that every continuous self-map

$$f : \Delta^{n-1} \to \Delta^{n-1}$$

has a fixed point. The proof does not use Scarf's theorem directly on the simplex itself. Instead, for each positive integer l, we work with a finite grid T_l of rational points whose coordinates have denominator l. Informally,

$$T_l = \left\{ x \in \Delta^{n-1} \mid x_i \in \frac{1}{l}\mathbb{Z} \text{ for all } i \right\}.$$

In Lean, the grid is represented by integer coordinates in `Fin (l+1)` whose sum is l; the corresponding point of the simplex is obtained by dividing each coordinate by l. The subtype `TT` defines the finite grid, while `TTtostdSimplex` and the coercion instance `TT.CoestdSimplex` realize a grid point as an element of the standard simplex.

4.1.1. The ordered grid and dominance estimates

To apply Scarf's theorem, each grid T_l is equipped with an indexed family of linear orders, one order for each coordinate. For a fixed coordinate i, the order first compares the i-th coordinate and then uses a lexicographic tie-breaker on the whole grid point. This is encoded by `TT.Ilt` and assembled into the indexed linear order `TT.ILO`. These orders are chosen so that dominance in the sense of Scarf's theorem imposes strong metric restrictions on the corresponding grid points. This is the point at which the abstract combinatorial theorem becomes a quantitative statement about the simplex.

The key quantitative input is that dominance on the grid gives metric control. A dominant set has small diameter, and the coordinates outside its color set are uniformly small. These estimates are the bridge between the finite combinatorial theorem and the limiting argument.

Theorem 4.1 (Dominance estimates on the grid)**.** *Let $\sigma \subseteq T_l$ and $C \subseteq \mathrm{Fin}\ n$. If σ is dominant with respect to C, then*

$$|x_i - y_i| < \frac{2(n+1)}{l}$$

for every $x, y \in \sigma$ and every $i \in \mathrm{Fin}\ n$. Moreover, if $i \notin C$, then

$$x_i < \frac{n+1}{l}$$

for every $x \in \sigma$.

Thus, as l tends to infinity, dominant rooms become small, and the coordinates not belonging to their color sets vanish in the limit.

4.1.2. The coloring induced by a self-map

Let $f : \Delta^{n-1} \to \Delta^{n-1}$ be continuous. For any point $x \in \Delta^{n-1}$, the coordinate sums of x and $f(x)$ are both equal to 1. Hence not all coordinates can strictly decrease: there exists an index i such that

$$x_i \leq f(x)_i.$$

The formalization packages this elementary observation as the nonemptiness lemma `stdSimplex.upidx` and the choice function `stdSimplex.pick` on the simplex. Restricting this choice to the grid T_l gives the coloring required by Scarf's theorem, namely `Fcolor`.

Applying the formalized Scarf theorem to `Fcolor` produces a colorful room on each grid T_l. In Lean, these rooms are chosen as a sequence `room_seq`, and a representative point from each room is chosen as `room_point_seq`. This sequence of rooms is the bridge between the finite combinatorial theorem and the limiting proof of Brouwer's theorem.

4.1.3. Compactness and the limiting argument

Choose one representative point from the colorful room obtained at each grid level. Since there are only finitely many possible color sets, we first pass to a subsequence on which the color set of the chosen colorful rooms is constant. This finite-image subsequence extraction is formalized by `exists_subseq_constant_of_finite_image`; the resulting constant color set and order embedding are packaged as `gpkg` and `g1`. We then use compactness of the standard simplex to extract a convergent subsequence of the corresponding room points; this step is packaged in Lean as `hpkg_aux` and `hpkg`. Let $z \in \Delta^{n-1}$ denote the resulting limit, and write the constant color set as C.

The dominance estimates now control the limit. First, the diameters of the selected colorful rooms tend to zero. Therefore every point in the same selected room converges to the same limit z. Second, for each coordinate $i \notin C$, the outside-coordinate estimate gives convergence to zero, and hence

$$z_i = 0 \qquad \text{for all } i \notin C.$$

Since z lies in the simplex, it follows that

$$\sum_{i \in C} z_i = 1.$$

On the other hand, the coloring rule says that for each color $i \in C$, the chosen room contains grid points whose i-th coordinate does not decrease under f. Passing to the limit and using continuity of f gives

$$f(z)_i \geq z_i \qquad \text{for all } i \in C.$$

Finally, $f(z)$ is also a point of the simplex. The inequalities on C, together with $\sum_{i \in C} z_i = 1$, force equality on all coordinates in C and force all coordinates outside C to be zero. Hence $f(z) = z$.

Theorem 4.2 (Brouwer's fixed point theorem for the standard simplex)**.** *Every continuous map $f : \Delta^{n-1} \to \Delta^{n-1}$ has a fixed point.*

4.2. Brouwer's Theorem on Products of Simplices

We next extend Brouwer's fixed point theorem from the standard simplex to finite products of simplices. This is the form needed for the game-theoretic application, where a mixed strategy profile is a point in a product of simplices. The formal proof reduces the product case to the standard-simplex case by constructing an explicit embedding–projection pair between the product and a larger simplex.

4.2.1. Reducing products to a standard simplex

Let I be a finite index type, and let $\mathrm{card}(i)$ be a positive integer for each $i \in I$. The product space considered in Lean is

$$\prod_{i \in I} \Delta^{\mathrm{card}(i)-1}.$$

It is represented by the type `ProductSimplices card`. The ambient simplex has total coordinate number

$$N = \sum_{i \in I} \mathrm{card}(i),$$

and is represented by `BigSimplex card`.

The main bookkeeping issue is that coordinates in the ambient simplex are indexed by a single flat index, while coordinates in the product are indexed by a pair consisting of a component $i \in I$ and a coordinate inside the i-th simplex. In Lean, this conversion is handled by `index_split` and `index_combine`. These maps allow the formalization to move between the block decomposition of the product and the flat coordinate system of the larger simplex.

4.2.2. The embedding-projection construction

The embedding sends a point of the product of simplices to a point of the ambient simplex by placing each component simplex into its corresponding coordinate block. The i-th block is rescaled by the weight $\mathrm{card}(i)/N$, so that the total sum of all coordinates is one. This construction is formalized as `embed_from_product`.

The projection in the opposite direction is more delicate. Starting from a point of the ambient simplex, the natural idea is to recover each component simplex by normalizing the corresponding coordinate block. However, this normalization is not well-defined if one of the blocks has total mass zero. To avoid this degenerate case, the formal construction first pushes the ambient point toward a fixed uniform point `z_uniform`. This auxiliary operation is implemented by `pushTowardsZ`, together with the definitions `deficit` and `tPush`. The key property is that after this push, every coordinate block has positive total mass. The projection is then obtained by normalizing each block, and is formalized as `project_to_product`.

The central property of the construction is the retraction identity

$$\texttt{project_to_product} \circ \texttt{embed_from_product} = \mathrm{id}.$$

For points already coming from the product, each coordinate block has the prescribed positive total mass, so the projection recovers the original product point. This identity is formalized by `project_embed_id`.

4.2.3. The product fixed-point theorem

It remains to transfer Brouwer's theorem from the ambient simplex back to the product. The embedding and projection are proved continuous as `embed_continuous` and

`project_continuous`. Given a continuous map

$$f : \prod_{i \in I} \Delta^{\mathrm{card}(i)-1} \to \prod_{i \in I} \Delta^{\mathrm{card}(i)-1},$$

we form a continuous self-map of the ambient simplex by composing projection, f, and embedding:

$$F = \texttt{embed_from_product} \circ f \circ \texttt{project_to_product}.$$

The standard-simplex version of Brouwer's theorem gives a fixed point of F. Projecting this fixed point back to the product and using the retraction identity gives a fixed point of f.

Theorem 4.3 (Brouwer fixed point theorem for products of simplices)**.** *Every continuous self-map of a finite product of simplices has a fixed point.*

5. Nash Equilibrium Existence in Finite Games

The final layer of the development turns the product fixed-point theorem into the classical existence theorem for finite games. This section is deliberately small: once Brouwer has been established for products of simplices, the remaining work is to define the mixed payoff function, construct the Nash self-map, and prove that fixed points of this map are exactly mixed equilibria.

5.1. Finite Games and Mixed Profiles

A strategic-form game is represented by a type of players I, a dependent family of pure strategy types S^i, and payoff functions $g^i : (\Pi_i S^i) \to \mathbb{R}$. The base structure `Game` only assumes that the player and strategy types are inhabited; the finite assumptions are isolated in `FinGame`. For a finite game, the mixed strategy space is the dependent product

$$\Delta(G) = \prod_{i \in I} \Delta(S^i),$$

implemented as `mixedS := (i : G.I) -> stdSimplex R (G.SS i)`.

The expected payoff of player i at a mixed profile σ is the finite multilinear extension of the pure payoff function:

$$\bar{g}^i(\sigma) = \sum_{s \in \Pi_j S^j} \left(\prod_j \sigma^j(s^j) \right) g^i(s).$$

In Lean this is `mixed_g`. The key algebraic lemma is that `mixed_g` is affine in one player's mixed strategy when the other components are fixed:

$$\bar{g}^i(\sigma[i \leftarrow y]) = \sum_{a \in S^i} y(a)\, \bar{g}^i(\sigma[i \leftarrow \delta_a]).$$

This lemma is the bridge from pure deviations to arbitrary mixed deviations in the final equilibrium proof.

A mixed Nash equilibrium is a mixed profile σ^* such that no player can improve expected payoff by replacing only their own component:

$$\forall i,\ \forall y \in \Delta(S^i), \qquad \bar{g}^i(\sigma^*) \geq \bar{g}^i(\sigma^*[i \leftarrow y]).$$

This is the predicate `mixedNashEquilibrium`.

5.2. The Nash Map

For a mixed profile σ, define the positive gain of switching player i to the pure strategy $a \in S^i$ by

$$A_i(a, \sigma) = \max\big(0, \bar{g}^i(\sigma[i \leftarrow \delta_a]) - \bar{g}^i(\sigma)\big).$$

The Nash map increases the weight of profitable pure deviations and then renormalizes within each player block:

$$F(\sigma)^i(a) = \frac{\sigma^i(a) + A_i(a, \sigma)}{\sum_{b \in S^i} \big(\sigma^i(b) + A_i(b, \sigma)\big)}.$$

The denominator is at least one, because the original mixed strategy has total mass one and all gain terms are nonnegative. Hence every block of $F(\sigma)$ is again a point of the corresponding simplex.

The continuity proof follows the syntax of the definition. Each coordinate is built from coordinate projections, finite sums and products, subtraction, `max`, and division by a nonzero denominator; Lean packages this in `nash_map_cont`. Applying the product version of Brouwer's theorem to this continuous self-map gives a fixed point $F(\sigma) = \sigma$.

It remains to relate fixed points of F to equilibrium. If every pure positive gain $A_i(a, \sigma)$ is zero, then every pure deviation is nonprofitable. By the linearity lemma for `mixed_g`, every mixed deviation is a convex combination of pure deviations, so no mixed deviation is profitable. Conversely, if some pure gain were positive at a fixed point, the normalization equation would force the denominator in player i's block to be strictly larger than one. The weighted-sum identity for the current mixed strategy then produces a strategy with positive mass and nonpositive gain, contradicting the fixed-point equation for that coordinate.

Theorem 5.1 (Existence of mixed Nash equilibria)**.** *Every finite strategic-form game has a mixed Nash equilibrium.*

6. BrouwerBench: A Lean-Grounded Pilot Benchmark

The formalization also yields a small benchmark for probing how language models read and explain Lean-grounded

mathematical developments. We call this benchmark *BrouwerBench*. Its purpose is not to measure end-to-end proof generation: the models are not asked to produce Lean terms that are compiled by the kernel. Instead, the benchmark asks whether a model can read local Lean context, identify the role of definitions and lemmas, and explain how proof components fit together across the Scarf–Brouwer–Nash pipeline. We therefore treat BrouwerBench v1 as a pilot proof-structure probe derived from this formalization, not as a broad benchmark for general Lean theorem proving.

6.1. Benchmark Construction

BrouwerBench v1 contains 80 question–answer items extracted from the formalization. The items cover four parts of the development: Scarf's theorem, the standard-simplex proof of Brouwer's theorem, the product-of-simplices extension, and the Nash equilibrium application. Each item contains a short task-specific Lean-grounded context, a natural-language question, a reference answer, evidence pointers to the Lean source, and a 0–2 grading rubric. The questions are organized by proof role, including definition roles, construction roles, proof dependencies, parity arguments, analytic limiting steps, theorem summaries, and endpoint connections.

During evaluation, each model receives the section-level Lean context together with the task-specific context and question. The section-level context includes the relevant definitions, structures, and abbreviations, such as the Scarf cell/room/door predicates, the finite-grid constructions used in Brouwer's theorem, the embedding–projection maps for products of simplices, and the Nash map definitions. The answer is then scored manually according to the item rubric: 0 means that the answer misses the main mathematical point, 1 means that it identifies part of the intended structure but omits an essential dependency, and 2 means that it explains the relevant Lean-grounded proof role accurately enough to match the reference answer. This scoring protocol is intended to support a qualitative pilot evaluation; the limitations of manual scoring are discussed in Section 7.1.

6.2. Models and Results

We evaluated four locally deployed models: `qwen3:8b` as a small baseline, `gemma3:12b`, `kimina-prover:7b`, and `gpt-oss:20b`. The results in Table 1 use the 80-item BrouwerBench v1 dataset, with a maximum score of 160 points. The percentage column reports the fraction of the maximum rubric score. Runtime is included only as descriptive information for these local runs.

Table 1. BrouwerBench v1 results. Scores use a 0–2 rubric per item.

Model	Score	Pct.	Time (s)	Avg. (s)
`gpt-oss:20b`	122/160	76.2%	1448.1	18.10
`qwen3:8b`	101/160	63.1%	428.1	5.35
`gemma3:12b`	89/160	55.6%	822.8	10.29
`kimina-prover:7b`	58/160	36.2%	986.4	12.33

Table 2. BrouwerBench v1 scores by formalization section.

Model	Scarf	Brouwer	Prod.	Nash
`gpt-oss:20b`	40/48	25/40	29/36	28/36
`qwen3:8b`	30/48	22/40	25/36	24/36
`gemma3:12b`	25/48	21/40	24/36	19/36
`kimina-prover:7b`	14/48	12/40	14/36	18/36

In this pilot setting, `gpt-oss:20b` obtains the highest score, with 122 out of 160 points. It scores especially well on theorem-summary and proof-dependency questions, where the task is to connect named Lean lemmas to the surrounding proof strategy. The smaller `qwen3:8b` baseline obtains 101 points and is comparatively strong on the product and Nash sections. This suggests that, for these 80 items, the score is not determined only by model size. The lower score of `kimina-prover:7b` should not be read as a general statement about prover-oriented models: BrouwerBench asks for explanatory proof-structure answers rather than low-level proof search or kernel-checked proof synthesis.

6.3. Interpretation

These results should be interpreted as evidence about Lean-grounded proof understanding within this development, not as direct evidence about the ability to synthesize complete formal proofs. A high-scoring answer typically names the relevant Lean objects and explains why they are needed: for example, why `typed_colorful_room_odd c default` uses an inhabited default type, why Brouwer's limiting argument first stabilizes a color/index set, why `project_embed_id` is the retraction identity needed in the product theorem, or how `mixed_g_linear` moves the Nash proof from pure deviations to mixed deviations. These are intermediate reasoning skills that support autoformalization workflows, but they do not by themselves certify a Lean proof.

BrouwerBench is therefore complementary to end-to-end theorem-proving evaluations. End-to-end tasks test whether a model can produce code accepted by Lean; BrouwerBench tests whether a model can explain the architecture of an existing formal proof from local formal context. For a formalization project like ours, this distinction is useful: many practical proof-assistant workflows require models to read existing definitions, identify reusable lem-

mas, explain failures, and suggest proof plans before a final proof script is produced.

7. Discussion

The formalization demonstrates that Scarf's combinatorial theorem can serve as a concrete bridge between finite discrete arguments and classical fixed-point applications in Lean. The main technical value of the development lies in the intermediate proof objects: dominant sets, rooms and doors, nearly colorful configurations, grid estimates on the standard simplex, and the embedding–projection construction for products of simplices. These objects make the proof route explicit and allow the final Nash equilibrium theorem to be traced back to a finite parity argument rather than to an opaque invocation of a fixed-point principle.

This structure is also useful from an AI-for-mathematics perspective. Large formal developments require more than isolated theorem proving: a model must understand how definitions are intended to be used, which lemmas carry the main mathematical content, and how local proof components support later arguments. BrouwerBench is designed to probe this kind of proof-structure understanding within the present development. Its results suggest that models differ substantially in their ability to explain Lean-grounded proof architecture, but the evaluation should be interpreted only as a pilot study.

7.1. Limitations

The formalization is specialized to one route from Scarf's theorem to Brouwer's fixed point theorem and Nash equilibrium existence. Other proofs of Brouwer's theorem, such as those based on simplicial approximation, degree theory, or classical topology, are outside the scope of the present development. Similarly, the Nash application covers finite games with mixed strategies; it does not address more general equilibrium concepts or games with infinite strategy spaces.

BrouwerBench also has important limitations. The current version contains 80 items, all extracted from a single formalization pipeline. It is therefore not large or diverse enough to support broad conclusions about general mathematical reasoning, Lean theorem proving, or the relative strength of different model families. The questions are designed to test local proof-structure understanding inside this development, not end-to-end synthesis of kernel-checked Lean proofs.

The scoring protocol is preliminary. Each answer is graded manually using a 0–2 rubric, and the current version does not report inter-annotator agreement, adjudication procedures for disputed samples, or a systematic error analysis. The reported numbers should therefore be read as pilot evidence about the usefulness of the task format, not as a statistically robust benchmark comparison. Future versions should include a larger and more diverse set of formal developments, multiple graders, explicit disagreement resolution, and more detailed error categories.

7.2. Future Work

There are several natural directions for extending this development. First, the Nash endpoint can be expanded into a broader Lean library for finite game theory. The present formalization establishes the existence of mixed Nash equilibria by reducing finite games to Brouwer's fixed point theorem on a product of simplices. A next step is to build on the same representation of mixed profiles and expected payoffs to formalize more structured classes of games, such as zero-sum games, extensive-form games, repeated games, and equilibrium refinements. This would turn the current endpoint into reusable infrastructure for verified game theory rather than a single isolated existence theorem.

Second, BrouwerBench can be extended from proof-understanding questions to kernel-checked autoformalization tasks. The current version evaluates whether models can identify definitions, dependencies, and proof roles inside an existing Lean development. A richer benchmark could include statement translation, lemma retrieval, proof-script generation, and proof repair from Lean error messages. More broadly, the Scarf–Brouwer–Nash pipeline suggests a reusable formalization pattern: isolate a finite combinatorial core, derive quantitative approximation estimates, and then transfer the result to analysis using compactness and continuity. Applying this pattern to related Sperner- or Scarf-style arguments may help build larger verified libraries at the interface of topology, optimization, and game theory.

Acknowledgements

We thank the anonymous reviewers for their constructive comments. We are grateful to Jiajun Ma for his guidance and support, and to Haocheng Wang for helpful discussions and feedback on the Lean development. This work was partially supported by the Xiamen University Malaysia Research Fund (Grant No. XMUMRF/2025-C15/IMAT/0036).

Impact Statement

This paper presents work whose goal is to advance the formalization of mathematics and the evaluation of AI systems for mathematical reasoning. The Lean development may contribute to more reliable and reproducible machine-checked proofs, while the accompanying bench-

mark may help evaluate autoformalization systems on structured proof developments involving combinatorics, topology, and game theory. However, the benchmark is narrow in scope and should not be interpreted as a comprehensive measure of mathematical reasoning ability. We do not anticipate direct negative societal impacts beyond the general risks of over-reliance on automated reasoning tools without sufficient proof-assistant or human verification.

References

Border, K. C. *Fixed Point Theorems with Applications to Economics and Game Theory*. Cambridge University Press, 1985.

Brouwer, L. E. Beweis der Invarianz der Dimensionenzahl. *Mathematische Annalen*, 70(2):161–165, 1911.

Ivanov, N. V. Beyond Sperner's lemma. *arXiv preprint arXiv:1902.00827*, 2019.

Moura, L. d. and Ullrich, S. The Lean 4 theorem prover and programming language. In *International Conference on Automated Deduction*, pp. 625–635. Springer, 2021.

Nash, J. F. Equilibrium points in N-person games. *Proceedings of the National Academy of Sciences*, 36(1):48–49, 1950.

Polu, S. and Sutskever, I. Generative language modeling for automated theorem proving. *arXiv preprint arXiv:2009.03393*, 2020.

Scarf, H. E. The computation of equilibrium prices: An exposition. In Arrow, K. J. and Intriligator, M. D. (eds.), *Handbook of Mathematical Economics*, volume 2, chapter 21, pp. 1007–1061. Elsevier, 1982. doi: 10.1016/S1573-4382(82)02016-5.

The mathlib Community. The Lean Mathematical Library. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*, CPP 2020, New Orleans, LA, USA, January 2020. ACM. doi: 10.1145/3372885.3373824. URL https://doi.org/10.1145/3372885.3373824.

Zheng, K., Han, J. M., and Polu, S. MiniF2F: A cross-system benchmark for formal olympiad-level mathematics. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=9ZPegFuFTFv.

A. Artifact and Version Information

The paper-specific Lean artifact is available at the stable tag https://github.com/Solo-ary/Game-Theory-Formalization/tree/camera-ready-icml2026. This repository contains the Lean files and BrouwerBench materials corresponding to the camera-ready version of this paper. The upstream maintained version of the formalization is available at https://github.com/math-xmum/Brouwer; this repository may continue to evolve after publication.

The camera-ready artifact uses Lean 4.22.0, pinned by `Brouwer/lean-toolchain`; exact dependency revisions are recorded in `Brouwer/lake-manifest.json`. The line counts reported in Table 3 are obtained by running `wc -l` on the Lean files in this tagged artifact.

The artifact contains two main parts. First, it contains the checked Lean source for the Scarf–Brouwer–Nash proof pipeline, including the finite Scarf combinatorics, the standard-simplex Brouwer proof, the product-simplex reduction, the Nash endpoint, and supporting simplex infrastructure. Second, it contains the BrouwerBench v1 materials: the 80-item dataset at `benchmarks/data/brouwerbench_v1.jsonl`, section-level context snippets under `benchmarks/context/`, raw outputs, scored JSONL files, and reports under `benchmarks/results/`, and manual score files under `benchmarks/scores/`.

Complete build, validation, model-running, and score-aggregation instructions are provided in the top-level README and in `benchmarks/README.md`.

B. Formalization Overview

The Lean development is organized as a proof pipeline. The Scarf file establishes the finite combinatorial theorem; the Brouwer file instantiates it on finite simplex grids and passes to a limit; the product file transports the standard-simplex fixed-point theorem to finite products of simplices; and the Nash file applies that product theorem to mixed strategy spaces. Table 3 gives the file-level organization and line counts used in this paper.

Table 3. File-level structure of the Lean formalization. Line counts are obtained by `wc -l` on the submitted Lean files.

File	Lines	Main endpoint	Role in pipeline
`Scarf.lean`	2600	`Scarf`	finite parity theorem for colorful rooms
`Brouwer.lean`	794	`Brouwer`	grid approximation and limiting fixed point
`Brouwer_product.lean`	671	`Brouwer_Product`	retraction from a big simplex to a product
`Nash.lean`	584	`ExistsNashEq`	fixed point implies mixed Nash equilibrium
`Simplex.lean`	119	supporting simplex lemmas	reusable simplex infrastructure
Total	4768	–	complete Scarf–Brouwer–Nash pipeline

The formalization also separates reusable infrastructure from proof endpoints. For example, the Scarf layer introduces an indexed family of linear orders before specifying any simplex grid, while the product layer proves its embedding and projection lemmas before the Nash layer uses products of simplices as mixed strategy spaces. This separation is mirrored by BrouwerBench: many benchmark items ask why an intermediate object is needed before the endpoint theorem can be applied.

C. Scarf Proof Components

Table 4 summarizes the combinatorial objects that carry the Scarf proof. The first layer is order theoretic: `isDominant` says that a finite set of points is controlled by a finite set of order indices. The second layer refines dominant pairs into cells, rooms, and doors by cardinality. The third layer equips these pairs with a coloring and counts typed nearly colorful incidences.

Table 4. Scarf proof components and their roles.

Lean object	Proof role	Used to establish
`IndexedLOrder`	family of orders indexed by colors	dominance relation
`isDominant`	order-theoretic cell condition	finite cell structure
`isCell`, `isRoom`, `isDoor`	cardinality refinements	room–door incidence graph
`isDoorof`	relation between a door and adjacent room	incidence counting
`isOutsideDoor`, `isInternalDoor`	boundary/interior split	degree one versus degree two
`isColorful`	cell has exactly its color image	target objects
`isNearlyColorful`, `isTypedNC`	one fixed missing color	fiberwise parity argument
`dbcountingset`	typed door–room incidences	double counting
`typed_colorful_room_odd`	odd colorful-room count	nonempty colorful filter
`Scarf`	endpoint theorem	colorful room exists

The room–door degree facts are the local combinatorial core. Outside doors have one adjacent room, while internal doors have two. After fixing a missing type, the parity proof partitions `dbcountingset` on the door side into outside and internal doors, and on the room side into colorful and non-colorful typed nearly colorful rooms. The outside contribution is odd, while the internal-door and non-colorful-room contributions are even. The remaining colorful-room contribution is therefore odd and hence nonzero.

D. Brouwer Proof Components

The standard-simplex proof converts the finite Scarf theorem into an analytic fixed point. Table 5 records the main proof dependencies.

Table 5. Key objects in the Scarf-to-Brouwer argument.

Lean object	Local meaning	Global use
`TT n l`	finite integer grid with coordinate sum `l`	finite domain for Scarf
`TTtostdSimplex`	divides grid coordinates by `l`	realizes grid points in the simplex
`TT.ILO`	indexed lexicographic orders on the grid	Scarf order structure
`Fcolor`	coordinate with `x_i <= f(x)_i`	grid coloring
`room_seq`	colorful room at each grid level	discrete approximations
`room_point_seq`	selected point from each room	sequence to compactify
`gpkg`	fixed color/index set and subsequence	stable coordinate split
`hpkg`	convergent subsequence package	limit point `z`
`tendsto_diam_to_zero`	rooms shrink along the grid	transfer between room points
`dominant_coords_tend_to_zero`	coordinates outside fixed set vanish	outside-support control
`f_coords_ge_z_coords`	inequalities pass to the limit	endpoint equality argument
`Brouwer`	standard-simplex theorem	fixed point `f z = z`

The final proof endpoint uses two kinds of estimates. On the stabilized color/index set C, colorful witnesses and continuity give coordinate inequalities $(fz)_i \geq z_i$. Outside C, dominance estimates show that the coordinates of z vanish. Since z and fz both lie in the simplex, the sum-one constraints leave no extra mass: the inequalities on C become equalities, the outside coordinates are zero, and extensionality yields $fz = z$.

E. Product-Simplex Proof Components

The product extension reduces a fixed-point problem on a finite product of simplices to a fixed-point problem on one larger simplex. The reduction requires explicit coordinate bookkeeping and a projection whose blockwise denominators are positive.

Table 6. Product-simplex constructions.

Lean object	Purpose	Key dependency
`index_split`, `index_combine`	convert flat and blockwise indices	inverse lemmas for sums
`blockSum`, `blockWeight`	measure mass in each block	positive target block weights
`deficit`, `tPush`	measure and bound mass correction	continuous push amount
`pushTowardsZ`	move toward a uniform interior point	positive post-push block sums
`project_to_product`	normalize each block	nonzero block denominators
`embed_from_product`	place product coordinates in big simplex	prescribed block masses
`project_embed_id`	projection after embedding is identity	fixed-point transport
`Brouwer_Product`	product fixed-point theorem	lift, apply Brouwer, project back

The positivity argument is the nontrivial part of projection. A raw point in the big simplex may have insufficient mass in a block, so blockwise normalization cannot be treated as ordinary coordinate projection. The push construction moves points toward a uniform interior point. If the push amount is positive, positive block weight contributes directly. If the push amount is zero, zero deficit shows that the original block already meets its positive target weight. These cases justify the denominator in `project_to_product` and allow the continuity proof to use division by a nonzero continuous function.

F. Nash Proof Components

The Nash layer turns the product fixed-point theorem into equilibrium existence. The finite game supplies one simplex per player, and the Nash map raises weights on profitable pure deviations before renormalizing each player block.

Table 7. Objects in the Nash endpoint.

Lean object	Local meaning	Endpoint role
`Game`, `FinGame`	strategic-form data with finiteness	finite mixed space
`mixedS`	product of player simplices	fixed-point domain
`mixed_g`	expected payoff	payoff comparison
`g_function`	current weight plus positive payoff gain	Nash-map numerator
`nash_map_aux`	normalized coordinate	simplex coordinate
`nash_map_cert`	sum-one and nonnegativity certificate	mixed profile output
`nash_map_cont`	continuity of Nash map	Brouwer hypothesis
`mixed_g_linear`	mixed payoff as pure-deviation average	pure-to-mixed step
`self_div_lemma`	normalization contradiction	rules out profitable pure deviations
`ExistsNashEq`	endpoint theorem	mixed Nash equilibrium exists

At a fixed point of `nash_map`, a profitable pure deviation would make the normalization denominator strictly larger than one. The proof then finds a nonzero coordinate with no positive improvement. Fixedness gives an equality of the form $\sigma_s = \sigma_s/\mathrm{sum_g}$, and `self_div_lemma` forces $\mathrm{sum_g} = 1$, contradicting the strict inequality. The remaining pure-deviation bound extends to arbitrary mixed deviations through `mixed_g_linear`.

G. Selected Lean Interfaces

This appendix records selected interfaces rather than full proof scripts. The complete proof terms and auxiliary lemmas are available in the repository; the snippets below show the formal boundaries between the main layers of the development.

G.1. Scarf's Combinatorial Core

```
class IndexedLOrder (I T : Type*) where
  IST : I → LinearOrder T
def isDominant (σ : Finset T) (C : Finset I) : Prop :=
  ∀ y, ∃ i ∈ C, ∀ x ∈ σ, y ≤[i] x
abbrev isRoom (σ : Finset T) (C : Finset I) : Prop :=
  isCell σ C ∧ C.card = σ.card
abbrev isDoor (σ : Finset T) (C : Finset I) : Prop :=
  isCell σ C ∧ C.card = σ.card + 1
theorem internal_door_two_rooms [Fintype T]
    (τ : Finset T) (D : Finset I)
    (h : IST.isInternalDoor τ D) :
    ∃ (σ₁ σ₂ : Finset T) (C₁ C₂ : Finset I),
      (σ₁, C₁) ≠ (σ₂, C₂) ∧
      IST.isRoom σ₁ C₁ ∧ IST.isRoom σ₂ C₂ ∧
      isDoorof τ D σ₁ C₁ ∧ isDoorof τ D σ₂ C₂ ∧
      (∀ σ C, IST.isRoom σ C → isDoorof τ D σ C →
        (σ = σ₁ ∧ C = C₁) ∨ (σ = σ₂ ∧ C = C₂)) := by
  ...
lemma NC_or_C_of_door
    (h1 : isTypedNC c i τ D) (h2 : isDoorof τ D σ C) :
    isTypedNC c i σ C ∨ isColorful c σ C := by
  ...
lemma typed_colorful_room_odd (i : I) :
    Odd (Finset.filter
      (fun x => isColorful c x.2.1 x.2.2)
      (dbcountingset c i)).card := by
  ...
theorem Scarf : (IST.colorful c).Nonempty := by
  ...
```

Listing 1. Representative interfaces for indexed orders, dominance, and Scarf's theorem.

G.2. Brouwer from Scarf

```
abbrev TT :=
  {x : Π₁ (_ : Fin n), Fin (l+1) |
    ∑ i, (x i : ℕ) = l}

def TTtostdSimplex (x : TT n l) : stdSimplex ℝ (Fin n) :=
  ...

def Fcolor (x : TT n l) : Fin n :=
  stdSimplex.pick x (f x)

theorem size_bound_in
    (σ : Finset (TT n l)) (C : Finset (Fin n))
    (h : TT.ILO.isDominant σ C) :
    ∀ x ∈ σ, ∀ y ∈ σ, ∀ i,
      |(x : stdSimplex ℝ (Fin n)) i -
       (y : stdSimplex ℝ (Fin n)) i| < 2 * (n + 1) / l := by
  ...

theorem size_bound_out
    (σ : Finset (TT n l)) (C : Finset (Fin n))
    (h : TT.ILO.isDominant σ C) :
    ∀ x ∈ σ, ∀ i ∉ C,
      (x : stdSimplex ℝ (Fin n)) i < (n + 1) / l := by
  ...

theorem f_coords_ge_z_coords
    (f : stdSimplex ℝ (Fin n) → stdSimplex ℝ (Fin n))
    (hf : Continuous f) :
    ∀ i ∈ (gpkg f).1.1,
      (f (hpkg f).1.1).1 i ≥ ((hpkg f).1.1).1 i := by
  ...

theorem Brouwer
    (f : stdSimplex ℝ (Fin n) → stdSimplex ℝ (Fin n))
    (hf : Continuous f) : ∃ x, f x = x := by
  ...
```

Listing 2. Representative interfaces for the grid construction and Brouwer endpoint.

G.3. Product Simplices

```
abbrev BigSimplex := stdSimplex ℝ (Fin (total_card card))

abbrev ProductSimplices :=
  (i : I) → stdSimplex ℝ (Fin (card i))

lemma project_embed_id (y : ProductSimplices card) :
    project_to_product card (embed_from_product card y) = y := by
  ...

lemma project_continuous :
    Continuous (project_to_product card) := by
  ...

lemma embed_continuous :
    Continuous (embed_from_product card) := by
  ...

theorem Brouwer_Product
    (f : ProductSimplices card → ProductSimplices card)
    (hf : Continuous f) :
    ∃ x : ProductSimplices card, f x = x := by
  ...
```

Listing 3. Representative interfaces for the embedding–projection reduction.

G.4. Nash Equilibria

```
variable (α : Type*) [Fintype α] [DecidableEq α]
abbrev S:= stdSimplex ℝ α
structure Game where
    I : Type*            -- The set of player
    --deEqI : DecidableEq I := inferInstance -- Decidable Eq
    HI : Inhabited I      -- at least one player
    SS : I → Type*        -- S is the set of strategies
    HSS (i :I) : Inhabited (SS i) -- The set of strategies is nonempty
    --deEqSS (i : I) : DecidableEq (SS i)
    g : I → (Π i, SS i) →  ℝ
    -- an elements in Π i, SS is a move of all players.
    -- g i is the payoff of the i-th player
structure FinGame extends Game where
  FinI : Fintype I
  FinSS : ∀ i : I , Fintype (SS i)

namespace FinGame
variable {G : FinGame}
variable (G) in
abbrev mixedS  := (i : G.I) → stdSimplex ℝ (G.SS i)

def mixed_g (i : G.I) (m : Π i, S (G.SS i) ) : ℝ := ∑ s : (Π j, G.SS j) , (∏ j,  m j (s j)) * (G.g i s)

lemma mixed_g_linear : G.mixed_g i (update  x i y) = ∑ s : G.SS i, y s * G.mixed_g i (update x i (stdSimplex.pure s)) := by
  ...

def mixedNashEquilibrium {G: FinGame} (x : G.mixedS) :=
  ∀ (i:G.I), ∀ (y : S (G.SS  i)),
     G.mixed_g i x ≥ G.mixed_g i (update  x i y)

end FinGame
section mixedNashEquilibrium
variable (G : FinGame)
open FinGame
variable {G}

noncomputable abbrev g_function (i : G.I) (σ : G.mixedS) (a : G.SS i) : ℝ :=
  σ i a + max 0 (mixed_g i (Function.update σ i (stdSimplex.pure a)) - mixed_g i σ)

lemma one_le_sum_g (i : G.I) (σ : G.mixedS) :
    1 ≤ ∑ b : G.SS i, g_function i σ b := by
  ...

noncomputable abbrev nash_map_aux (σ : G.mixedS) (i : G.I) (a : G.SS i) : ℝ :=
  g_function i σ a / ∑ b : G.SS i, g_function i σ b

lemma nash_map_cert (σ : G.mixedS) (i : G.I) :
  (nash_map_aux σ i) ∈ S (G.SS i) := by
  ...

variable (G)

noncomputable def nash_map (σ: G.mixedS) : G.mixedS :=
  fun (i : G.I) ↦ ⟨nash_map_aux σ i, nash_map_cert σ i⟩

lemma nash_map_cont : Continuous $ nash_map G :=
  by
  ...

theorem ExistsNashEq : ∃ σ : G.mixedS , mixedNashEquilibrium σ := by {
  ...
}

end mixedNashEquilibrium
```

Listing 4. Representative interfaces for the Nash endpoint.

H. BrouwerBench Dataset

BrouwerBench v1 is stored as JSONL. Every row corresponds to one hand-checkable question about a local proof role in the formalization. The 80 items are distributed across the same four proof layers as the paper.

Table 8. BrouwerBench v1 coverage by formalization section.

Section	Scarf	Brouwer	Product Brouwer	Nash	Total
Items	24	20	18	18	80

Table 9. BrouwerBench v1 task types.

Type	Def. role	Proof dep.	Parity	Summary	Constr. role	Analysis	Endpoint
Items	7	28	4	3	17	11	10

H.1. Dataset Schema

Each item contains the fields listed in Table 10. The `context` and `question` fields are shown to the model together with a section-level Lean prelude. The reference answer and rubric are used for grading rather than included in the prompt.

Table 10. Fields in one BrouwerBench JSONL item.

Field	Meaning
`id`	stable item identifier
`section`	one of `Scarf`, `Brouwer`, `Brouwer_product`, or `Nash`
`task_type`	proof-role category used for analysis
`source`	Lean source file from which the item is drawn
`section_prelude`	Lean-style declaration snippet prepended to the prompt
`context`	compact task-specific formal context
`question`	natural-language query answered by the model
`gold_answer`	reference explanation used during grading
`evidence`	file-line anchors used to create and check the item
`rubric`	item-specific 0–2 criteria and required ideas

H.2. Example Items

Scarf definition-role item. Item `scarf_001` supplies the local context

> `isCell` abbreviates `isDominant`; `isRoom` adds equal point/index cardinalities; `isDoor` adds one extra index.

It asks how `isDominant`, `isCell`, `isRoom`, and `isDoor` relate. A score-2 answer must explain dominance as the core cell condition, state the two cardinality refinements, and connect them to the room–door parity structure.

Product endpoint item. Item `product_011` provides the lifted map

$$\texttt{f_lifted} = \texttt{embed_from_product} \circ f \circ \texttt{project_to_product}$$

and asks why the fixed point is first found in `BigSimplex` rather than directly in `ProductSimplices`. A score-2 answer must explain that the available Brouwer theorem is applied to the lifted single-simplex map, then projection and `project_embed_id` transport the equation back to the product.

Nash endpoint item. Item `nash_011` gives the contradiction branch in `ExistsNashEq`: the hypothesis $1 < \mathrm{sum_g}$, the fixed-point equation for `nash_map`, and `self_div_lemma`. A complete answer explains how a nonzero coordinate gives $\sigma_s = \sigma_s/\mathrm{sum_g}$, why `self_div_lemma` forces $\mathrm{sum_g} = 1$, and how this contradicts the profitable-deviation branch.

I. Scoring Protocol

The benchmark uses manual item-level scoring. Each item has a rubric tailored to its proof role, but all rubrics follow the shared scale in Table 11.

Table 11. Shared BrouwerBench score scale.

Score	Interpretation	Typical case
0	misses the formal proof role	generic Brouwer or Nash explanation unrelated to the given Lean context
1	partially identifies the role	names a subsequence but misses why the fixed set C is needed
2	explains the Lean-grounded role	connects the named object, dependency, and proof endpoint

The grading distinguishes a mathematically plausible answer from a formalization-specific one. An answer that correctly describes a classical theorem but ignores the supplied Lean objects receives partial credit only when it still captures an essential proof dependency. Hallucinated Lean claims, such as applying the wrong endpoint theorem or inventing an unsupported proof route, lower the score. Exact spelling of every Lean identifier is not required when the mathematical dependency is unambiguous, but named objects matter when the question is specifically about their proof role.

The current study uses one manual score file per model output. This makes errors auditable at item level: the scored JSONL artifacts store the answer, rubric, score, and score note together. As discussed in §7, a larger benchmark should add independent graders and adjudication for borderline partial-credit cases.

J. Detailed Benchmark Results

The main text reports overall and section-level scores. Table 12 gives the task-type breakdown. It shows that the evaluation probes several kinds of proof understanding rather than a single style of question.

Table 12. BrouwerBench v1 scores by task type.

Model	Def.	Dep.	Parity	Summary	Constr.	Analysis	Endpoint
`gpt-oss:20b`	9/14	45/56	7/8	6/6	26/34	15/22	14/20
`qwen3:8b`	8/14	38/56	5/8	4/6	23/34	13/22	10/20
`gemma3:12b`	5/14	34/56	5/8	4/6	20/34	11/22	10/20
`kimina-prover:7b`	6/14	20/56	1/8	1/6	16/34	9/22	5/20

Table 13. Reported model artifact stems and decoding settings.

Model	Shared artifact stem	Decoding note
`gpt-oss:20b`	`gpt-oss_20b_np4096`	`num_predict=4096`
`qwen3:8b`	`qwen3_8b`	default short-answer run
`gemma3:12b`	`gemma3_12b`	default short-answer run
`kimina-prover:7b`	`kimina-prover_7b`	default short-answer run

The stems in Table 13 appear after the shared dataset prefix `brouwerbench_v1__`. Raw outputs use the result directory and the `.jsonl` suffix, while manual score files use the score directory and the `.manual.jsonl` suffix. The full per-item responses, manual score notes, scored JSONL files, and generated Markdown reports are retained in the benchmark artifact directory. This keeps the paper tables compact while preserving the item-level audit trail used for the reported totals.

K. Reproducibility Notes

The reported runs were produced locally with Ollama on an Apple M4 Pro machine with 24 GB memory. Lean 4.22.0 is pinned by `Brouwer/lean-toolchain`, and package revisions are recorded in `Brouwer/lake-manifest.json`. The benchmark artifact records the dataset at `benchmarks/data/brouwerbench_v1.jsonl`, raw outputs and generated reports under `benchmarks/results/`, and manual score files under `benchmarks/scores/`.

Benchmark validation checks JSONL parsing, unique ids, rubric fields, source and prelude links, evidence anchors, and whether task contexts leak proof-body material. Detailed command-line instructions for reproducing validation, local model runs, and score aggregation are provided in the repository README.

The reported GPT run uses a single longer output budget, `num_predict=4096`, for all 80 items. Row `product_008` still exhausts the thinking budget, so its empty response is retained and scored as a failure.

Section-level prompt preludes are stored as Lean-style snippets rather than standalone Lean modules. They expose the relevant definitions, structures, abbreviations, and inductive declarations to the model while avoiding proof-body leakage. The snippets are therefore prompt context artifacts; they are validated by the benchmark tooling but are not compiled as independent Lean files.