AVSAD: AUTOMATING VECTOR SYMBOLIC ARCHITECTURE DISCOVERY WITH ITERATIVE EVOLUTION

Deja N. Scott
Molinaroli College of Engineering and Computing
University of South Carolina
Columbia, SC 29208, USA
ds17@email.sc.edu

Dmitry Zubarev
IBM Research

Massimiliano Esposito
IBM Research

Avraham Shinnar
IBM Research

Abbas Rahimi
IBM Research – Zurich

Kenneth L. Clarkson
IBM Research

Lior Horesh
IBM Research

Michael Hersche
IBM Research – Zurich

Shashanka Ubaru
IBM Research

ABSTRACT

We present the Automated Vector Symbolic Architecture Discovery (AVSAD) pipeline, a novel framework for automating the design and formal verification of Vector Symbolic Architecture (VSA) binding operations under user-defined algebraic constraints. Traditional VSA development is a labour-intensive, manual process, relying on expert knowledge to design and verify binding operations, a bottleneck that limits both the pace and scope of VSA research. AVSAD addresses this by integrating an ensemble of Large Language Models (LLMs) with formal verification via Lean 4 to autonomously generate, assess, and iteratively refine candidate VSA binding operations. The pipeline operates through five key stages: Formal Specification, Prompt Generation, LLM Ensemble candidate generation, Verification via Python benchmarking and Lean 4 property testing, and Semantic Feedback-driven iterative refinement. To evaluate the pipeline, we instantiate AVSAD on the problem of discovering binding operations satisfying associativity, non-commutativity, and computational efficiency constraints, properties motivated by their relevance to State Space Model and Finite State Automata emulation. Our results demonstrate that AVSAD successfully characterises generated binding operations under formal constraints, discovering candidates that satisfy user-defined algebraic properties and providing performance profiles comparable to known VSA operations, representing a step towards fully automated, constraint-driven VSA design.

1 INTRODUCTION

The design of efficient data representations is a fundamental challenge in modern computing, particularly as applications increasingly demand systems capable of structured reasoning under uncertainty. Traditional approaches rooted in set logic suffer from combinatorial explosion: as set size N increases, the number of possible subsets grows exponentially as 2^N, rendering such approaches impractical for large-scale datasets and prohibitive in time complexity for intricate operations. Addressing these limitations requires representational frameworks that support symbolic reasoning while remaining scalable, noise-tolerant, and compatible with modern learning paradigms.

Vector Symbolic Architectures (VSAs), also known as Hyperdimensional Computing (HDC) systems, offer a solution to these challenges. VSAs represent symbols as high-dimensional random vectors, and construct complex representations through algebraic operations, such as binding, bundling, and permutation, that preserve semantic structure without explicit enumeration. This fusion of symbolic and vector-based computation enables structured information processing at scale, circumventing the combinatorial explosion inherent in set-theoretic approaches while maintaining compatibility

with neural learning paradigms. The binding operation, in particular, is central to VSA expressivity, encoding relational structure between symbolic entities in a manner that supports both composition and retrieval.

Despite significant progress in VSA research, the development and verification of novel VSA variants remains a largely manual, labour-intensive process. Designing a new binding operation requires expert knowledge of the target algebraic properties, careful implementation, and rigorous verification in which the current workflow is both time-consuming and difficult to scale. Existing automated discovery frameworks, such as FunSearch Romera-Paredes et al. (2024), have demonstrated the potential of LLM-guided search for mathematical and algorithmic structures, but have not addressed the specific demands of VSA design, where formal correctness proofs and user-defined algebraic constraints are essential requirements.

To address this gap, we introduce the Automated VSA Discovery (AVSAD) pipeline, a novel framework that automates the generation, assessment, and iterative refinement of VSA binding operations under user-defined constraints. AVSAD integrates an ensemble of Large Language Models with formal verification via Lean 4, enabling autonomous exploration of the VSA design space while providing machine-checked correctness guarantees. Candidate operations are generated from formal specifications derived from domain knowledge, assessed against Python benchmarks, and formally verified through Lean 4 property testing. A Semantic Feedback component leverages LLMs to diagnose and resolve verification failures, enabling the pipeline to iteratively converge on valid solutions without manual intervention.

The contributions of this paper are as follows:

1. The AVSAD Pipeline: A novel, end-to-end automated pipeline for the discovery and formal verification of VSA binding operations under user-defined algebraic constraints, integrating an LLM ensemble employing six open-source models with systematically varied sampling hyperparameters, and Lean 4 formal verification to provide machine-checked correctness guarantees beyond unit testing.
2. Semantic Feedback for Iterative Refinement: A structured error correction mechanism that classifies and resolves both syntactic and semantic Lean 4 verification failures using specialised LLMs, enabling automated convergence on formally correct VSA candidates without manual intervention.
3. Empirical Evaluation on Constrained VSA Discovery: A demonstration of AVSAD's capacity to characterise and discover novel binding operations satisfying associativity, non-commutativity, and computational efficiency constraints, motivated by their relevance to State Space Model and Finite State Automata emulation, with performance benchmarking against eight known VSA operations

2 Related Works

2.1 Vector Symbolic Architecture

VSAs differ by vector space and operation implementation, with each variant exhibiting distinct representational properties. Kanerva (1996) introduced the concept of hyperdimensional computing, in which computation is performed using high-dimensional random vectors to support distributed representations and algebraic manipulation, establishing the foundational principles of scalability and noise robustness that characterise the field. Plate subsequently developed Holographic Reduced Representations (HRR), introducing binding and unbinding operations in continuous vector space to encode symbolic structures. Building on this, Kanerva (1996) developed Binary Spatter Codes (BSC), employing binary vectors with bitwise XOR binding to enhance noise resilience. Gayler & Wales (1998), who coined the term "Vector Symbolic Architecture," constructed the Multiply-Add-Permute (MAP) variant for bipolar vectors, incorporating element-wise multiplication, addition, and a permutation operation to preserve encoding order. More recently, Gosmann & Eliasmith (2019) developed the Vector-Derived Transformation Binding (VTB) variant to provide improved binding fidelity for symbolic processing within neural architectures, while Gallant et al. designed the Matrix Binding of Additive Terms (MBAT) variant to address representational constraints in complex structured data encoding. Hersche et al. further advanced the field by integrating VSAs with deep neural

networks to produce a hybrid Neuro-Vector-Symbolic model, leveraging the symbolic manipulation capabilities of VSAs alongside the perceptual strengths of neural networks. Despite these significant advances, the development and verification of VSA variants has remained a largely manual, labour-intensive process, constraining both the pace of discovery and the breadth of architectures explored. AVSAD directly addresses this bottleneck by automating the generation and formal verification of novel VSA binding operations under user-defined constraints.

2.2 Formal Verification

Formal verification encompasses the use of mathematical logic and rigorous proof techniques to guarantee the correctness of algorithms and systems, conducted either by human experts, automated proof assistants such as Rocq, Isabelle, and Lean 4, or increasingly through collaboration between the two. Established proof assistants have driven considerable advances in this area, yet significant challenges persist. Translating informal mathematical reasoning into machine-verifiable formal specifications remains labour-intensive and error-prone Dailler et al. (2018), and even well-understood proofs can harbour subtle gaps when subjected to formal scrutiny Kreuzer (2024). Specification incompleteness extends beyond logical correctness, requiring explicit treatment of environmental context and unstated proof obligations Autexier et al. (2012).

Recent works striving to close the gap between informal mathematical reasoning and formal proof construction have leveraged LLMs with growing success. TheoremLlama Wang et al. (2024) demonstrated that general-purpose LLMs can be trained to generate Lean 4 proofs by aligning natural language and formal language proof corpora, while Lean Copilot Song et al. (2025) introduced an interactive framework in which LLMs suggest proof steps and complete proof goals natively within Lean, automating over 74% of proof steps on average. DeepSeek-Prover Xin et al. (2024) further advanced the field through large-scale synthetic data generation for Lean-based theorem proving. More recently, systems combining informal reasoning with formal verification have achieved remarkable results: Aristotle Achim et al. (2025) attained gold-medal-equivalent performance on the 2025 International Mathematical Olympiad by integrating informal lemma generation with a Lean proof search system, while Gauss Math, Inc. (2025), developed by Math Inc., completed a formalisation of the strong Prime Number Theorem in Lean comprising over 1,000 theorems, a project that had resisted completion by human experts for over 18 months, highlighting the potential of autonomous LLM-driven formalisation at scale.

Despite these advances, challenges specific to automated proof repair and efficient verification remain. APOLLO Ospanov et al. (2025) addresses proof repair directly, decomposing failing proofs into sub-lemmas, applying targeted syntax and semantic corrections, and recombining repaired components, which achieved state-of-the-art results on the miniF2F benchmark at a fraction of the conventional sampling budget. AVSAD builds on this paradigm, extending compiler-guided error correction to the domain of VSA discovery: rather than repairing proofs for known theorems, AVSAD generates and formally verifies entirely novel algebraic operations under user-defined constraints, with Semantic Feedback providing an analogous but domain-adapted repair mechanism.

2.3 Self-Evolving Frameworks

Self-evolving frameworks utilising LLMs aim to create systems capable of autonomously improving their capabilities through iterative learning and adaptation without direct human intervention. While demonstrating promising results on targeted tasks, this area faces significant challenges. Current approaches suffer from error propagation and resistance to correction when applied to complex reasoning problems Singh et al. (2025); Jiang et al. (2025a), a form of ”feedback friction” that can impede the discovery of novel structures. These systems are further prone to ”model collapse,” in which iterative self-improvement produces superficial rather than substantive changes Jiang et al. (2025b), which is a limitation that is particularly consequential for the generation of innovative architectures such as novel VSA variants. A deeper theoretical gap lies in the absence of a robust framework governing how self-improvement should occur, leaving mechanisms largely underspecified and results inconsistent Tao et al. (2024); Huang et al. (2024). The reliance on self-generated data also risks limiting solution diversity and introducing systematic bias. The lack of reliable self-verification, which is a cornerstone of AVSAD’s Lean 4 verification stage, remains a fundamental unresolved

challenge in this paradigm Jiang et al. (2025b); Hendrycks et al. (2021), underscoring the necessity of external, formal verification as a correctness guarantee in automated discovery pipelines.

3 METHODOLOGY

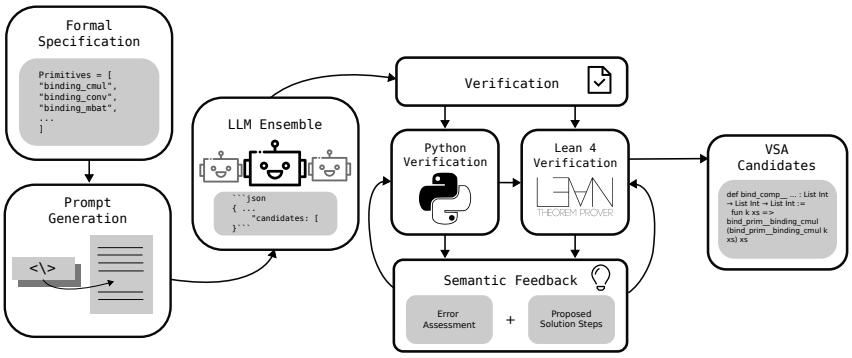


Figure 1: Overview of the AVSAD pipeline. Candidate VSA binding operations are generated by the LLM Ensemble from prompts derived during Formal Specification, assessed against user-defined constraints via Python benchmarking, and formally verified through Lean 4 property testing. Candidates failing either verification stage are routed to the Semantic Feedback component for iterative error correction, repeating until verification succeeds or a maximum number of attempts is exhausted. Successfully verified candidates are retained and scored; unresolved candidates are discarded.

This section presents AVSAD, a novel pipeline for the automated discovery and formal verification of VSA binding operations under user-defined constraints. AVSAD coordinates a sequence of specialised components: Formal Specification, Prompt Generation, LLM Ensemble, Verification, and Semantic Feedback. Each of these stages is described in detail in the subsections that follow. Figure 1 provides an overview of the full pipeline architecture. The pipeline is initiated by the Formal Specification phase, which extracts domain-specific knowledge from academic literature and transforms it into a structured set of foundational primitives that govern subsequent candidate generation. These primitives, together with user-defined constraints, are integrated by the Prompt Generation component into dynamic, parameterised prompt templates, ensuring consistent and structured input to the LLM Ensemble across all pipeline stages. The LLM Ensemble queries a diverse set of open-source models in parallel to generate candidate VSA binding operations, represented as JSON-encoded Abstract Syntax Trees (ASTs), with systematically varied sampling hyperparameters to maximize solution diversity. Generated candidates are subsequently translated into executable Python code and assessed by the Verification component, which evaluates both computational correctness against user-defined constraints via a Python benchmark suite and formal correctness via Lean 4 property testing. Candidates that satisfy both verification stages are retained as valid discoveries; those that fail are routed to the Semantic Feedback component, which leverages specialized LLMs to diagnose and resolve errors before resubmitting refined candidates for re-verification. This cycle repeats until a candidate achieves full verification or a user-specified maximum number of refinement attempts is exhausted.

3.1 FORMAL SPECIFICATION

The Formal Specification phase establishes the foundation for AVSAD by enabling users to define constraints and desired properties of the target VSA operations. This process ensures that generated operations adhere to user-specified characteristics throughout the pipeline. To achieve this, we employ algorithm synthesis techniques to extract key foundational operations directly from relevant VSA academic literature. These extracted elements form the basis for a prompt template used by subsequent phases, guiding the LLM ensemble towards generating operations that satisfy the defined constraints.

The generative process is structured into three stages: Primitive Generation, Symbolic Generation, and Python Code Generation. Initially, Primitive Generation identifies and extracts foundational

VSA operations, primitives, from a curated corpus of domain-specific knowledge using an LLM. These primitives serve as building blocks for constructing more complex operations. Subsequently, Symbolic Generation leverages the prompt template to instruct the LLM ensemble in generating candidate VSAs by combining these primitives into Abstract Syntax Trees (ASTs). This stage is guided by both user-defined constraints and a set of semantic inference rules. Candidate VSAs are represented as JSON objects, capturing the semantic analysis, structural properties, primitive decomposition and derivation rules, which are prime insights for operation discovery. Finally, Python Code Generation translates the resulting ASTs into executable Python code, with the LLM responsible for directly implementing all required primitives within the generated module. This self-contained approach enhances reproducibility and simplifies debugging of the synthesized operations.

3.2 PROMPT GENERATION

To facilitate coordinated operation between Formal Specifications, LLM-driven exploration, and rigorous verification within the AVSAD pipeline, we have developed a dynamic Prompt Generation module. This module goes beyond static prompting by employing parameterized templates that are populated with task-specific data at runtime. This approach allows us to leverage a shared pool of LLMs across diverse tasks, which includes initial VSA candidate generation, error elaboration, and iterative refinement based on interpreter feedback, while maintaining precise control over the prompt's content and structure. For example, when translating the AST candidates to executable Python code, we can dynamically inject each JSON object into the pre-defined template along with the instructions and guardrails for constrained code generation. During verification, templates are dynamically updated with error messages, stack traces, and counterexamples generated by Python and Lean 4 interpreters, facilitating targeted refinement of candidate VSAs. Future work will explore the incorporation of reinforcement learning techniques to optimize these template parameters using VSA validity metrics. This dynamic prompting strategy is critical for ensuring the generation of valid VSAs that adhere to user-defined constraints and are amenable to formal verification.

3.3 LLM ENSEMBLE

The LLM Ensemble comprises a collection of large language models tasked with generating candidate VSA binding operations based on user-defined properties and constraints. By leveraging multiple models, this approach enhances robustness and broadens the space of potential solutions explored during candidate generation. The ensemble combines general-purpose and reasoning-oriented ("thinking") LLMs selected for their demonstrated capabilities in code generation and structured reasoning. Specifically, we employ a diverse set of open-source models: IBM Granite 3.3, Meta Llama 3, Microsoft Phi 4, Mistral AI Mixtral, Google Gemma 3, and Alibaba Qwen 2.5. The use of open-source models serves a dual purpose: it mitigates the risk of solution bias that may arise from dependence on a single proprietary model, and it promotes exploration across varied solution spaces.

To further encourage generative diversity, we introduce controlled variation across three key sampling hyperparameters. Temperature governs the stochasticity of token prediction, with higher values producing more divergent outputs. We supply a graduated range of temperature values (0.8–2.0) to promote increasingly diverse candidate compositions across ensemble members. Top-K sampling restricts prediction at each step to the k most probable tokens; we apply higher values (150–200) to expand the vocabulary of candidate binding operation components. Top-P sampling (nucleus sampling) constrains generation to the smallest token set whose cumulative probability meets or exceeds p; we set p=0.95 to encourage broad token exploration during operation composition. Collectively, these hyperparameter configurations are designed to maximise candidate diversity and increase the likelihood of discovering novel binding operations. Each model in the ensemble is queried in parallel using a shared prompt, producing JSON files encoding the ASTs of candidate VSA binding operations. This parallelised, multi-model architecture enables the pipeline to generate a wide variety of candidate compositions, facilitating the discovery of novel binding operations that satisfy user-specified constraints.

3.4 VERIFICATION

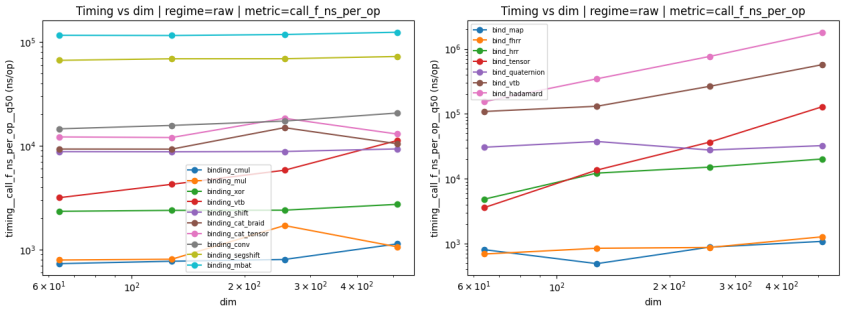
The Verification phase is critical to ensuring both user-defined constraint satisfaction and demonstrable semantic correctness of automatically generated VSAs. Initially, a Python benchmark suite assesses key computational qualities of candidate solutions, including operational complexity and scalability, and rigorously assess adherence to user-defined constraints. This benchmarking process serves two primary purposes: first, it facilitates the filtering of VSA operations that may not align with user-specified performance characteristics; second, it captures potential errors within the Python code implementations for subsequent refinement via the Semantic Feedback component. Candidate VSAs are then subjected to formal verification using Lean 4, a state-of-the-art dependent type theory-based theorem prover and code verifier. This process involves translating VSA representations into executable formal proofs that conform to Lean 4's reasoning standards. The resulting Lean 4 code is automatically tested against a suite of formally defined property tests, which are derived during the Formal Specification phase, and mirroring those used in the initial Python benchmark suite to ensure the properties in the formal proof are equivalent to those initially derived candidates. The verification process captures detailed diagnostic information from Lean 4's output, including precise error locations and descriptive messages. A quantitative evaluation score is generated based on this analysis; this score reflects the degree to which candidate VSAs satisfy the defined properties using an additive weighted approach, allowing for user-specified prioritization of constraint importance. Detected errors are systematically channelled back into the Semantic Feedback loop, enabling iterative refinement of LLM-generated VSAs and facilitating targeted improvements in architecture design.

3.5 SEMANTIC FEEDBACK

The Semantic Feedback component is designed to iteratively improve candidate VSAs that fail initial verification. This process leverages LLMs to analyse errors identified during both Python benchmarking and Lean 4 property testing, providing targeted guidance for refinement. When an error is detected, relevant code snippets alongside the error message are injected into a specialized correction template to construct a prompt. This prompt is then presented to an LLM, initially Qwen 2.5 Coder, tasked with diagnosing the root cause of the error and proposing corrective actions. The resulting error analysis is subsequently incorporated into a refined prompt, which is passed to a language-specialized LLM. For example, when correcting Python code, Qwen 2.5 Coder is employed; conversely, DeepSeek Prover V2 is utilized for refining the syntactical and semantic structure of Lean 4 code. Candidates that do not pass the verification steps are discarded from the pipeline's candidate pool, however, a copy of the operation's lean and python code is retained in the filesystem for later use. This iterative process allows the pipeline to refine candidates and progressively converge towards valid VSA solutions.

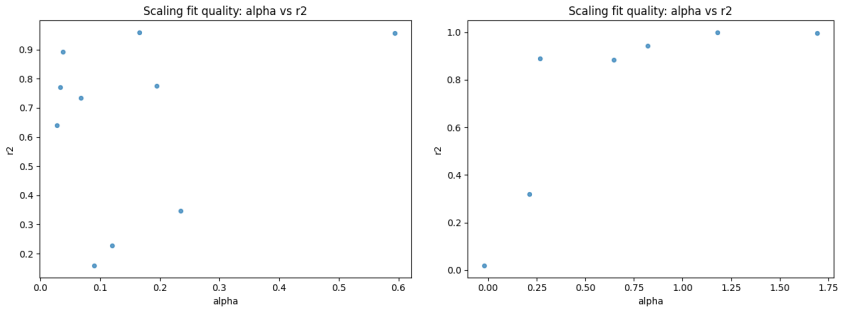
4 EXPERIMENTS

To evaluate the AVSAD pipeline's capacity for constrained VSA discovery, we draw on the work of Terzić et al. (2025), who identified a fundamental limitation of transformer architectures: their tendency to struggle with algorithmic tasks and exhibit poor extrapolation on longer sequences. These challenges can be naturally framed in terms of Finite State Automata (FSAs), inherently algebraic structures of varying complexity. While State Space Models (SSMs) can emulate complex FSAs through unstructured transition matrices, this approach incurs significant computational overhead. Terzić et al. (2025) demonstrated that representing such transitions as sparse $N \times N$ matrices, though powerful, remains costly. We reformulate this challenge as a search for novel algebraic operations acting on vectors that represent both the FSA and its unstructured transition matrices, which lies in the domain of VSA binding operations. To guide discovery, we impose three constraints: associativity, to ensure parallelisation and unambiguous multi-step transitions crucial for stable gradient flow in SSMs; non-commutativity, to preserve the directionality inherent in FSA transitions; and computational efficiency, to ensure practical scalability. For comparative analysis, we evaluate eight known VSA binding operations as a baseline: BSC, MAP, HRR, VTB, SDR, Hadamard/Walsh, Quaternion, and Tensor Product Binding. Each is implemented in both Python and Lean 4 and assessed alongside AVSAD-generated candidates through the pipeline's Python benchmark suite and Lean 4 property tests.



(a)Generated VSA Binding Operations (b) Known VSA Binding Operations

Figure 2: Median per-operation latency (ns/op) of VSA binding operations as a function of vector dimensionality, for (a) generated and (b) known binding operations. Operations span several orders of magnitude in latency, with most generated operations remaining dimension-stable while known operations, such as "bind_hadamard" and "bind_vtb" exhibit significant scaling.



(a)Generated VSA Binding Operations (b) Known VSA Binding Operations

Figure 3: Scaling exponent (α) vs. goodness-of-fit (R^2) for (a) generative and (b) known VSA binding operations. Known operations span a wider range of α with higher fit confidence, while several generated operations exhibit ill-fit to power-law despite non-trivial scaling.

The Python benchmark suite is designed to characterise the performance of binding operations across varying input dimensions. For each operation f, we measure execution time over pairs of vectors at systematically increasing dimensionalities, recording median latency (ns/op) to quantify both absolute performance and scaling behaviour. Scaling is modelled using a power-law relationship, $time \approx C \cdot d^{\alpha}$, where d denotes vector dimension; we report both the scaling exponent α and the coefficient of determination R^2 to assess fit quality. Figure 2 compares median latency against vector dimension for AVSAD-generated and known binding operations. The generated operations tend to be faster and more dimension-stable, while several known operations exhibit scaling penalties at higher dimensionality. Figure 3 presents the corresponding scaling fit quality. The majority of generated operations yield near-flat scaling profiles; however, a subset exhibits low R^2 values, suggesting irregular behaviour that does not conform to the power-law model. One notable candidate achieves α=0.06, indicating near-constant scaling and a reliable fit. Among the known operations, three achieve near-perfect R^2 values, indicating strong and reliable power-law scaling.

To assess formal algebraic correctness, we evaluate both generated and known operations against two property tests: associativity and non-commutativity. Scores are awarded additively, 1 point for associativity and 2 points for non-commutativity, yielding a maximum score of 3 for operations satisfying both properties. Figure 4 presents the score distribution across 35 AVSAD-generated candidates. The majority score 0, failing to satisfy either property; we attribute this primarily to errors introduced during Lean 4 code generation by the pipeline, and identify improved code generation fidelity as a key objective for future work. Five candidates satisfy associativity exclusively, and six satisfy both properties, which demonstrates that the pipeline is capable of discovering formally verified, constraint-satisfying binding operations. No candidate satisfies non-commutativity with-

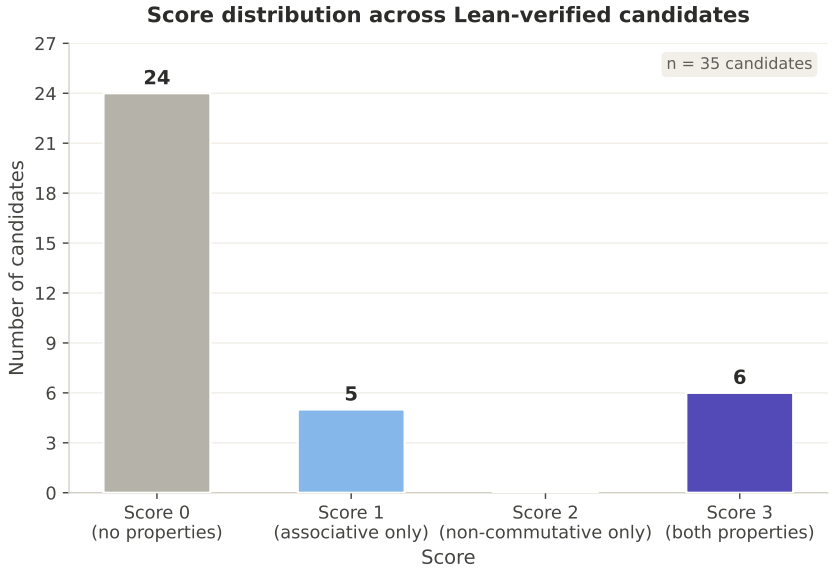


Figure 4: Distribution of Lean 4 verification scores across 35 candidate VSA binding operations generated by the AVSAD pipeline. Scores are computed additively based on property satisfaction: associativity contributes 1 point and non-commutativity contributes 2 points, yielding a maximum score of 3. The absence of score 2 indicates that no candidate satisfied non-commutativity without also satisfying associativity.

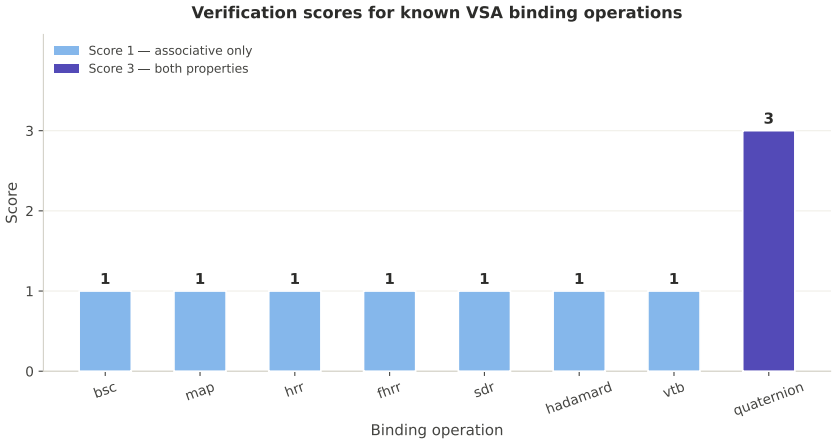


Figure 5: Lean 4 verification scores for eight known VSA binding operations used as a baseline reference. All operations satisfy associativity, score 1, with the exception of "binding_quaternion", which additionally satisfies non-commutativity to achieve the maximum score of 3. This distribution serves as a point of comparison for the candidate operations discovered by the AVSAD pipeline.

out also satisfying associativity. Figure 5 presents the corresponding scores for the eight known operations; seven satisfy associativity only, and one, Quaternion Binding, satisfies both properties.

Figure 6 breaks down generated candidate scores by model. Granite 3.3 contributes four candidates satisfying both properties despite generating the fewest overall candidates, while Gemma 3 produces the largest total candidate count with two achieving the maximum score. Llama 3, Phi 4, and Qwen 2.5 do not produce any verified candidates in this evaluation. These results underscore the value of a diverse LLM ensemble: no single model dominates across both quantity and quality, and the models that produce the most candidates are not necessarily those that produce the highest-scoring ones.

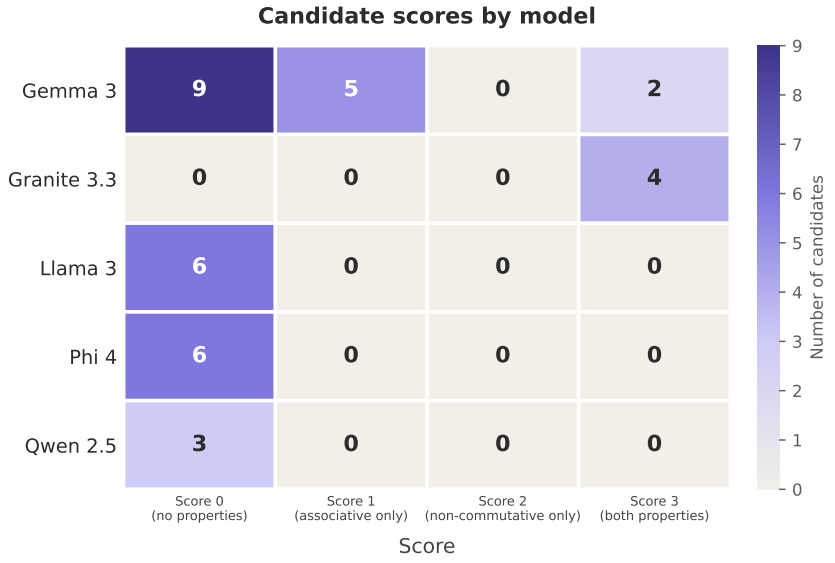


Figure 6: Per-model breakdown of Lean 4 verification scores across the LLM ensemble. Each cell denotes the number of candidates generated by a given model achieving a particular score. Granite 3.3 produces the highest proportion of maximally scoring candidates relative to its sample count, while Llama 3, Phi 4, and Qwen 2.5 did not produce any verified candidates in this evaluation, motivating the ensemble design as a strategy for broadening solution coverage.

While the current results reflect a preliminary evaluation run, they demonstrate that AVSAD can characterise and discover novel VSA binding operations under formal algebraic constraints, which provides a foundation for more extensive evaluation in future work.

5 Future Work

While the AVSAD pipeline demonstrates considerable promise, several directions for future research warrant further investigation. One key avenue is the incorporation of reinforcement learning (RL) to manage and optimise interactions within the multi-agent framework. Current coordination of multiple LLMs relies on careful prompt engineering and manual tuning; RL presents a principled pathway toward automated agent collaboration and adaptive strategy refinement, reducing the need for hand-crafted coordination mechanisms. As the pipeline explores increasingly complex problem instances, the potential for exponential growth in the candidate solution space poses a significant computational bottleneck. To address this, we are investigating techniques such as learned pruning heuristics and dynamic resource allocation to ensure that the pipeline remains tractable at scale. Another direction involves the development of a dedicated dynamic database for storing and reusing previously evaluated VSA configurations. Inspired by the approach employed in FunSearch, this database would automatically maintain a curated set of high-quality candidates as seeds for further exploration, substantially reducing redundant computation and accelerating the discovery process. Additionally, the database would support automated updates to the library of known operations, ensuring that the most promising configurations are continuously propagated through the pipeline. Finally, we envision extending the AVSAD methodology beyond VSA discovery to address broader computational challenges. Potential application domains include automated algorithm design and the formal verification of complex systems.

6 Conclusion

We have presented AVSAD, a novel pipeline for the automated discovery and formal verification of VSA binding operations under user-defined algebraic constraints. By integrating an ensemble of open-source LLMs with Lean 4 theorem proving and a structured Semantic Feedback mechanism,

AVSAD addresses a longstanding bottleneck in VSA research: the labour-intensive, manual process of designing and verifying novel binding operations. The pipeline's modular architecture, which encompasses Formal Specification, Prompt Generation, LLM Ensemble generation, Verification, and Semantic Feedback, enables autonomous exploration of the VSA design space while providing machine-checked correctness through formal proof construction. Our empirical evaluation demonstrates AVSAD's capacity to characterise and discover binding operations satisfying user-defined constraints of associativity, non-commutativity, and computational efficiency, motivated by their relevance to State Space Model and Finite State Automata emulation. Across 35 Lean-verified candidates, six satisfy both algebraic properties to achieve the maximum verification score, with Granite 3.3 and Gemma 3 contributing all maximally scoring candidates, underlining the value of a diverse LLM ensemble for broadening solution coverage. Python benchmarking further reveals that generated operations tend to exhibit favourable and stable scaling profiles relative to known VSA operations, with several candidates demonstrating near-constant computational complexity. These findings suggest that the integration of LLM-driven search with formal verification constitutes a principled and scalable approach to automated algebraic structure discovery. AVSAD demonstrates the potential of combining the generative flexibility of LLMs with the rigorous correctness of formal verification, offering a template for automated discovery in domains where both creativity and provable correctness are essential.

REFERENCES

Tudor Achim, Alex Best, Alberto Bietti, Kevin Der, Mathïs Fédérico, Sergei Gukov, Daniel Halpern-Leistner, Kirsten Henningsgard, Yury Kudryashov, Alexander Meiburg, Martin Michelsen, Riley Patterson, Eric Rodriguez, Laura Scharff, Vikram Shanker, Vladmir Sicca, Hari Sowrirajan, Aidan Swope, Matyas Tamas, Vlad Tenev, Jonathan Thomm, Harold Williams, and Lawrence Wu. Aristotle: Imo-level automated theorem proving, 2025. URL https://arxiv.org/abs/2510.01346.

Serge Autexier, Dominik Dietrich, and Marvin Schiller. Towards an intelligent tutor for mathematical proofs. *Electronic Proceedings in Theoretical Computer Science*, 79:1–28, February 2012. ISSN 2075-2180. doi: 10.4204/eptcs.79.1. URL http://dx.doi.org/10.4204/EPTCS.79.1.

Sylvain Dailler, Claude Marché, and Yannick Moy. Lightweight interactive proving inside an automatic program verifier. *Electronic Proceedings in Theoretical Computer Science*, 284:1–15, November 2018. ISSN 2075-2180. doi: 10.4204/eptcs.284.1. URL http://dx.doi.org/10.4204/EPTCS.284.1.

Ross Gayler and Roger Wales. Connections, binding, unification and analogical promiscuity. 01 1998.

Jan Gosmann and Chris Eliasmith. Vector-derived transformation binding: An improved binding operation for deep symbol-like processing in neural networks. *Neural Computation*, 31(5):849–869, 2019. doi: 10.1162/neco_a_01179.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset, 2021. URL https://arxiv.org/abs/2103.03874.

Michael Hersche, Mustafa Zeqiri, Luca Benini, Abu Sebastian, and Abbas Rahimi. A neuro-vector-symbolic architecture for solving Raven's progressive matrices. *Nature Machine Intelligence*, 5(4):363–375. ISSN 2522-5839. doi: 10.1038/s42256-023-00630-8. URL https://doi.org/10.1038/s42256-023-00630-8.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. Large language models cannot self-correct reasoning yet, 2024. URL https://arxiv.org/abs/2310.01798.

Dongwei Jiang, Alvin Zhang, Andrew Wang, Nicholas Andrews, and Daniel Khashabi. Feedback friction: Llms struggle to fully incorporate external feedback, 2025a. URL https://arxiv.org/abs/2506.11930.

Yuhua Jiang, Yuwen Xiong, Yufeng Yuan, Chao Xin, Wenyuan Xu, Yu Yue, Qianchuan Zhao, and Lin Yan. Pag: Multi-turn reinforced llm self-correction with policy as generative verifier, 2025b. URL https://arxiv.org/abs/2506.10406.

Pentti Kanerva. Binary spatter-coding of ordered k-tuples. In Christoph von der Malsburg, Werner von Seelen, Jan C. Vorbrüggen, and Bernhard Sendhoff (eds.), *Artificial Neural Networks — ICANN 96*, pp. 869–873, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg. ISBN 978-3-540-68684-2.

Katharina Kreuzer. Verification of correctness and security properties for crystals-kyber. In *2024 IEEE 37th Computer Security Foundations Symposium (CSF)*, pp. 511–526, 2024. doi: 10.1109/CSF61375.2024.00016.

Math, Inc. Introducing Gauss, an agent for autoformalization. https://www.math.inc/gauss, 2025. Accessed: 2026-03-18.

Azim Ospanov, Farzan Farnia, and Roozbeh Yousefzadeh. Apollo: Automated llm and lean collaboration for advanced formal reasoning, 2025. URL https://arxiv.org/abs/2505.05758.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024. URL https://doi.org/10.1038/s41586-023-06924-6.

Joykirat Singh, Tanmoy Chakraborty, and Akshay Nambi. Self-evolved preference optimization for enhancing mathematical reasoning in small language models, 2025. URL https://arxiv.org/abs/2503.04813.

Peiyang Song, Kaiyu Yang, and Anima Anandkumar. Lean copilot: Large language models as copilots for theorem proving in lean, 2025. URL https://arxiv.org/abs/2404.12534.

Zhengwei Tao, Ting-En Lin, Xiancai Chen, Hangyu Li, Yuchuan Wu, Yongbin Li, Zhi Jin, Fei Huang, Dacheng Tao, and Jingren Zhou. A survey on self-evolution of large language models, 2024. URL https://arxiv.org/abs/2404.14387.

Aleksandar Terzić, Michael Hersche, Giacomo Camposampiero, Thomas Hofmann, Abu Sebastian, and Abbas Rahimi. On the expressiveness and length generalization of selective state-space models on regular languages, 2025. URL https://arxiv.org/abs/2412.19350.

Ruida Wang, Jipeng Zhang, Yizhen Jia, Rui Pan, Shizhe Diao, Renjie Pi, and Tong Zhang. Theoremllama: Transforming general-purpose llms into lean4 experts, 2024. URL https://arxiv.org/abs/2407.03203.

Huajian Xin, Daya Guo, Zhihong Shao, Zhizhou Ren, Qihao Zhu, Bo Liu, Chong Ruan, Wenda Li, and Xiaodan Liang. Deepseek-prover: Advancing theorem proving in llms through large-scale synthetic data, 2024. URL https://arxiv.org/abs/2405.14333.