META-ADAPTIVE PRIME GENERATION: TOWARD A HARDWARE-AWARE AND INPUT-SCALE CONDITIONAL CONSTRUCTIVE SIEVE

PU JUSTIN SCARFY YANG

Abstract. We introduce the Meta-Prime Engine (M-PE), a dynamic, hardware-aware, and input-scale adaptive framework for prime number generation. Built upon the Yang Dynamic Generative Sieve (YDGS), M-PE selects optimal prime-generating subalgorithms in real time based on detected CPU architecture, cache profile, SIMD support, and numerical input scale. It unifies classical sieve methods, priority-queue structures, segmented memory filters, and future algorithmic innovations under a single epistemically complete dispatch framework. The design includes a registrable algorithmic layer and extensible dispatch interface, allowing seamless integration of any future prime-generating method while preserving logical consistency. This work presents the first formal foundation for a forever-extensible, constructively justified, and performance-optimized prime generation system—capable of learning, adapting, and evolving as computing paradigms change.

Contents

Date: May 15, 2025.

1. Introduction

Prime generation lies at the heart of computational number theory and cryptography. Traditional algorithms optimize either for bounded sieving (e.g., Sieve of Eratosthenes) or incremental logic (e.g., priority queue sieves), but rarely adapt dynamically to machine context or problem scale. This paper proposes a *meta-adaptive* approach—formally defining an architecture-aware and input-sensitive algorithm that chooses optimal submethods based on environment detection.

2. The Yang Dynamic Generative Sieve (YDGS)

We begin with the base logic:

Definition 1 (Yang Dynamic Constructive Sieve)**.** *Let $\mathcal{P}_n = \{p_1 = 2, \ldots, p_n\}$ be the known primes. Define*

$$S_n := \left\{ m > p_n \mid \forall p_i \in \mathcal{P}_n,\ p_i^2 \leq m \Rightarrow p_i \nmid m \right\},$$

and set

$$p_{n+1} := \min S_n.$$

Theorem 1. *The sequence $\{p_n\}$ generated by YDGS is the full ordered list of all primes.*

Proof. See [1]. Each candidate m not divisible by any $p_i \leq \sqrt{m}$ must be prime. Since all such p_i are tracked, completeness and soundness follow. □

3. Meta-Adaptive Algorithm Architecture

3.1. **Indicator Functions for Dispatch.** Let C denote CPU context and I denote input scale. Define two indicator functions:

$$\texttt{CPU\_ID}() \to \mathcal{C}, \quad \texttt{Input\_Size}(N) \to \mathcal{I},$$

where $\mathcal{C}, \mathcal{I}$ are finite selector sets (e.g., cache tiers, SIMD flags, size brackets).

Definition 2 (Meta-Selector)**.** *Define a dispatch map:*

$$\Phi : \mathcal{C} \times \mathcal{I} \to \mathcal{A},$$

where $\mathcal{A}$ is the set of subalgorithms $\{A_1, A_2, \ldots, A_k\}$, including:

- *A_1: YDGS base logic*
- *A_2: Wheel-enhanced sieve*
- *A_3: Priority Queue Sieve*
- *A_4: GPU-parallel segmented sieve*

- A_5*: Lazy filter pipeline*

3.2. **Execution Engine.** The algorithm begins by calling:

$$(C, I) := (\texttt{CPU\_ID}(), \texttt{Input\_Size}(N)),$$
$$A := \Phi(C, I), \quad \texttt{run}(A, N).$$

Remark 1. *Each A_i is optimized for a specific (C, I) zone. For instance, if C has AVX512 and $I > 10^8$, A_4 may be optimal. If C is cache-constrained and $I < 10^3$, A_1 may be preferable.*

4. The Meta-Prime Engine (M-PE)

We define:

Definition 3 (Meta-Prime Engine)**.** *The set of all context-sensitive mappings over YDGS-compatible subalgorithms:*

$$\textit{M-PE} := \{\Phi(C, I) \mapsto A_i \mid A_i \textit{ is logically consistent with YDGS}\}.$$

M-PE acts as a meta-algorithm controller, adapting to any known or future CPU input, allowing hardware-aware prime generation at optimal performance.

5. Theoretical Implications and Outlook

- We generalize sieve theory into **environmental sieve systems**.
- The YDGS logic serves as the invariant backbone, extendable to distributed systems or Yang_n number fields.
- AI-based prediction modules can estimate prime density in a range to further enhance heuristic performance.

6. Future-Proof Design and Algorithmic Integrability

A major goal of the Meta-Prime Engine (M-PE) is to not only optimize for current architectures and input scales, but to be *eternally extensible*—designed from the outset to seamlessly incorporate any future prime-generating subalgorithm.

6.1. **Formal Design of Future Integrability.**

Definition 4 (Algorithm Registry Layer)**.** *Let $\mathcal{A}_{reg}$ be the registered set of known subalgorithms:*

$$\mathcal{A}_{reg} = \{A_i \mid A_i : \textit{prime generator},\ i \in \mathbb{N}\}.$$

For each new algorithm A_{new}, if it satisfies logical compatibility with the YDGS base and is verifiably sound, then:

$$\mathcal{A}_{reg} \leftarrow \mathcal{A}_{reg} \cup \{A_{new}\}.$$

Definition 5 (Extensible Selector Map)**.** *Let Φ be promoted to a dynamic dispatch map:*

$$\Phi : \mathcal{C} \times \mathcal{I} \times \mathcal{F} \to \mathcal{A}_{reg},$$

where $\mathcal{F}$ represents any external algorithm flag space (e.g., future heuristic IDs, AI-metadata tags).

Remark 2. *M-PE contains a standing interface for future registration:*

$$\texttt{register\_new\_prime\_algorithm}(A_{new}, \textit{conditions}, \textit{meta-data})$$

which ensures compatibility with the sieve engine and incorporates dispatch information into Φ*.*

6.2. **Philosophical Position.** By allowing any future logical, AI-assisted, or hybrid quantum prime algorithms to be registered and executed via conditionally defined logical selectors, M-PE becomes a *permanently expandable epistemic structure*—a universal framework for the construction of primes beyond all known and unknown present methods.

Definition 6 (Epistemic Prime Engine)**.** *A Prime Engine* E *is called* epistemic *if it is designed such that:*

$$\forall A_{future},\ A_{future} \models \texttt{is\_prime\_generator} \Rightarrow A_{future} \hookrightarrow E$$

without re-architecting the core of E*.*

7. Future-Proofing and Algorithmic Longevity

A critical feature of the Meta-Prime Engine (M-PE) is its ability to adapt not only to present architectures and input domains, but also to unforeseen developments in algorithmic theory and computing environments. We formalize this in terms of registrability, dispatch generality, and epistemic openness.

7.1. **Registrable Algorithmic Layer.** To accommodate future algorithms A_{future} not yet discovered or published, we define a formal structure that allows dynamic integration.

Definition 7 (Algorithmic Registry)**.** *Let* $\mathcal{A}_{\text{reg}}$ *be the dynamic registry of all algorithmic modules recognized by the engine. The registry is defined inductively as*

$$\mathcal{A}_{\text{reg}} := \{A_i \mid A_i \textit{ satisfies correctness and constructivity criteria}\},$$

with future additions permitted via:

$$A_{future} \in \mathcal{A}_{\text{reg}} \iff \texttt{verify}(A_{future}) = \texttt{true}.$$

Remark 3. *The verification function may include formal proof of correctness, benchmarking signatures, or AI-assisted meta-validation of functional equivalence to known primality logic.*

7.2. **Dynamic Dispatch Mapping.** We expand the selector function to allow future insertion points:

Definition 8 (Extended Dispatch Map)**.** *Let* $\Phi : \mathcal{C} \times \mathcal{I} \times \mathcal{F} \to \mathcal{A}_{\text{reg}}$ *be a dynamic dispatch function, where* $\mathcal{F}$ *is a metadata space of external indicators (e.g., algorithm type ID, AI model tag, version key).*

Given a future subalgorithm A_{new}, the M-PE dispatcher assigns it to appropriate (C, I) zones via an extensible rule table:

$$\Phi(C, I, f_{\text{new}}) = A_{\text{new}}, \quad \text{with } f_{\text{new}} \in \mathcal{F}_{\text{future}}.$$

7.3. **Philosophical Motivation.** We postulate that prime generation is not a closed domain of optimization, but an open-ended frontier, requiring an epistemically complete system:

Definition 9 (Epistemically Complete Algorithmic Framework)**.** *A prime-generating framework E is* epistemically complete *if:*

$$\forall A, \text{ if } A \models \texttt{is\_prime\_generator} \Rightarrow A \hookrightarrow E,$$

with no loss of logical consistency or structural integrity.

Theorem 2 (Universal Integrability Theorem)**.** *Let E be the Meta-Prime Engine. If A is any future algorithm such that:*

1. *A produces the correct set of primes,*
2. *A is expressible in finite logical or symbolic rules,*

then A can be integrated into E via registry insertion and dispatch mapping without modifying the core structure of E.

Proof. Since E's registry and dispatch are defined inductively and dynamically, it suffices that A be correctly specified and verifiable within existing or extended rules. Therefore, it is formally composable within E. □

7.4. **Operational Interface Specification.** To support integration in practice, the following meta-interface is envisioned:

```
function register_new_prime_algorithm(A_new, meta):
    verify A_new satisfies YDGS compatibility
    compute dispatch zone (C*, I*, F*)
    insert A_new into A_reg
    update dispatch map Phi(C*, I*, F*) <- A_new
```

This interface ensures long-term expandability of M-PE without redesign.

8. Self-Adaptive Learning Layer (Future Module)

To extend M-PE's capabilities, we propose a future layer of AI-assisted meta-reasoning.

Definition 10 (Self-Adaptive Prime Engine)**.** *A future-enhanced prime engine E^+ includes a learning module Λ such that:*

$$\Lambda : \textit{observed runtime behavior} \mapsto \textit{dispatch preference updates},$$

optimizing algorithm selection, pre-filter tuning, and parallel resource allocation.

Remark 4. *The learning layer Λ can employ reinforcement learning or meta-gradient tuning over $\mathcal{A}_{\text{reg}}$ to optimize prime discovery rates over various domains.*

8.1. **Examples of Future Dispatch Domains.** Let us illustrate possible future dispatch scenarios:

- **Quantum SIMD CPU with 2048 threads**: dispatch to A_{quant}
- **Low-power edge device**: dispatch to A_{frugal} (minimal cache footprint)
- **AI-assisted prime-density predictor**: dispatch to $A_{\text{ML-heuristic}}$
- **Newly discovered recursive sieve method (2027)**: dispatch to $A_{\text{recursive-27}}$

Definition 11 (Time-Indexed Dispatch Readiness)**.** *The M-PE is said to be future-ready at time t if:*

$$\forall \textit{ prime generator } A_{t+1},\ \exists\ \Phi_t \textit{ such that } A_{t+1} \in \textit{image}(\Phi_t).$$

9. System Architecture and Module Overview

The Meta-Prime Engine (M-PE) is structured as a multi-layered architecture, where each component encapsulates a logically coherent subsystem.

9.1. **Module Layers.**

- **Layer 0: Base Constructive Logic** – YDGS (Yang Dynamic Generative Sieve)
- **Layer 1: Algorithm Registry** – Verifies, stores, and tracks all registered subalgorithms
- **Layer 2: Hardware and Input Profilers** – Detects CPU, memory, instruction set, input scale
- **Layer 3: Dispatch Layer** – Uses indicator functions to select optimal subalgorithm
- **Layer 4: Future Learning Interface (optional)** – Accepts AI or human-guided insertion of novel prime generators

9.2. **Recommended TikZ Diagram (verbal description).** To visually represent the system:

- Use a horizontal layered diagram with arrows flowing from: - Input (N) $\rightarrow$ Profiler (C, I) $\rightarrow$ Dispatch Selector $\rightarrow$ Subalgorithm Executor $\rightarrow$ Prime Stream Output - The Algorithm Registry is drawn vertically adjacent to the Dispatch Layer - The Learning Interface connects into the Dispatch Layer via a dotted line, showing its optional nature

9.3. **Component Interaction Logic.**

Definition 12 (Dynamic Execution Flow)**.** *Let N be the requested upper bound or number of primes. Then the evaluation path proceeds as:*

$$\begin{aligned} N &\longrightarrow (\texttt{CPU\_ID}(), \texttt{Input\_Size}(N)) = (C, I) \\ (C, I, F) &\longrightarrow \Phi(C, I, F) = A \in \mathcal{A}_{\text{reg}} \\ \texttt{run}(A, N) &\longrightarrow \{p_1, p_2, \ldots, p_k\} \end{aligned}$$

Remark 5. *This structure generalizes runtime behavior to a symbolic dynamic system. Every registered subalgorithm A is responsible for satisfying YDGS-equivalent correctness.*

10. Concluding Reflections: Toward a Unified Prime Theory

The M-PE is more than a performance optimization; it is a reconceptualization of what a "prime generator" can be. The classical vision of sieving has always been static—bounded, predetermined, and algorithmically inert. M-PE redefines this vision by embracing:

- **Logical Constructivism:** Every prime is produced explicitly by filtration logic from prior primes.
- **Structural Extensibility:** Any valid method may join the generative process if verifiably sound.
- **Computation-Environment Awareness:** Prime generation is no longer independent of its computing substrate.
- **Meta-Epistemic Generality:** The system is designed to remain open to all future knowledge.

Just as Euclid showed that primes never end, the M-PE shows that our ways of constructing them never need to end either.

11. Future Directions

We outline several directions for future inquiry and expansion:

11.1. **Formal Verification and Proof Assistant Integration.** Embedding the YDGS logic and M-PE dispatch framework into systems like Lean 4, Coq, or Agda would provide certified correctness and integration with formal mathematics. Each subalgorithm A_i can be encoded as a `structure` with soundness proofs.

11.2. **$\mathbf{Yang_n}$ Number System Integration.** Extend the notion of prime generation to Yang-type abstract fields and topological algebraic objects, where primality must be reinterpreted structurally. Define analogues of prime ideals or prime filters within $\mathrm{Yang_n}$ models and determine if YDGS logic generalizes.

11.3. **Quantum and Non-Classical Environments.** Adapt the M-PE to quantum computing platforms, where dispatch logic depends on qubit coherence profiles or decoherence-adapted execution models. Create definitions for probabilistic primality pipelines in non-deterministic environments.

11.4. **Self-Modifying Meta-Dispatch Systems.** Develop Φ_t as a learning function that evolves based on feedback:

$$\Phi_{t+1} := \Lambda(\Phi_t, \text{runtime logs}, \text{benchmark success}, \text{future oracle hints})$$

This defines an autonomous epistemic agent of prime generation.

12. Acknowledgments

The author gratefully acknowledges the long tradition of sieve theory and constructive mathematics upon which this system is built. Deep thanks also go to the modern algorithmic community, whose methods inspired the dispatch structure herein. Special recognition to ongoing conversations between logic, category theory, and AI that fuel this interdisciplinary synthesis.

13. Conclusion and Future Research

We propose the first formal definition of a hardware-aware, input-size conditional prime generation framework. By combining constructive sieve logic with adaptive dispatch, the M-PE lays a path toward a universal and future-proof prime algorithm engine.

The Meta-Prime Engine (M-PE) not only adapts to present computation environments and data scales but is built to be future-proof by design. Its dispatch logic, registration interfaces, and epistemic openness make it the first algorithmic framework that can expand to absorb all future prime-generating methods, including those not yet discovered or conceived.

Further work includes:

- Defining formal logic for the auto-verification of subalgorithm soundness.
- Establishing registries for community-defined algorithmic modules.
- Integrating AI-based meta-algorithm advisors for future heuristic evolution.

References

[1] Pu Justin Scarfy Yang, *The Yang Dynamic Generative Sieve: Constructive Prime Generation Revisited*, Preprint, 2025.

[2] Melissa E. O'Neill, *The Genuine Sieve of Eratosthenes*, Journal of Functional Programming, 15(4), 415–433, 2004.

[3] Richard Crandall and Carl Pomerance, *Prime Numbers: A Computational Perspective*, 2nd ed., Springer, 2005.

[4] Richard Bird, *Pearls of Functional Algorithm Design*, Cambridge University Press, 2010.

[5] Donald E. Knuth, *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*, 3rd ed., Addison-Wesley, 1997.

[6] Paul Pritchard, *Linear Prime-number Sieves: A Family Tree*, Science of Computer Programming, 9(1), 17–35, 1987.

[7] Jonathan P. Sorenson, *Two Compact Incremental Prime Sieves*, Mathematics of Computation, 91(334), 1363–1390, 2022.

[8] Hans Riesel, *Prime Numbers and Computer Methods for Factorization*, Birkhäuser, 1994.

[9] Philippe Flajolet and Robert Sedgewick, *Analytic Combinatorics*, Cambridge University Press, 2009.

[10] Thomas H. Cormen et al., *Introduction to Algorithms*, 3rd ed., MIT Press, 2009.

[11] Andrew Granville, *Prime Number Patterns*, The American Mathematical Monthly, 115(4), 279–296, 2008.

[12] Terence Tao and Van H. Vu, *Additive Combinatorics*, Cambridge Studies in Advanced Mathematics, 2006.

[13] H. W. Lenstra Jr., *Primality Testing Algorithms*, Séminaire Bourbaki 1986/1987, no. 669.