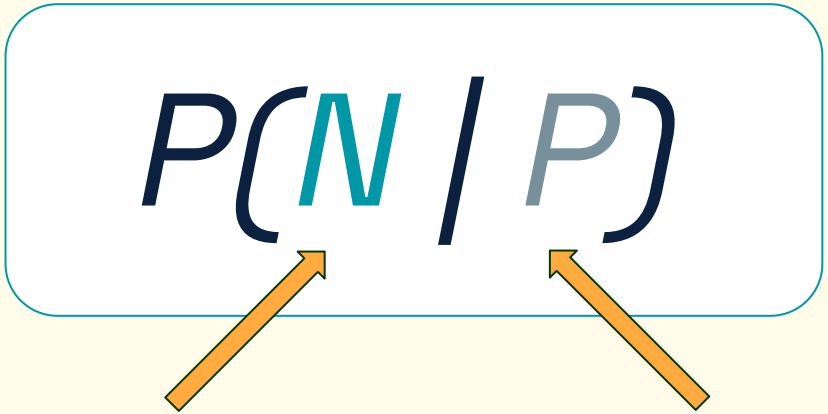

Gödel's Poetry
Recursively Decomposing LLM Based Prover
AI4Math Workshop 2026
July 2026

Section 0

Background

LLM: Conditional Probability Model



Lean: Programming Language

Constructive Logic

- Logical Proposition
- Proof of a proposition
- Logical implication
- ...

Curry-Howard Isomorphism

Programming Language

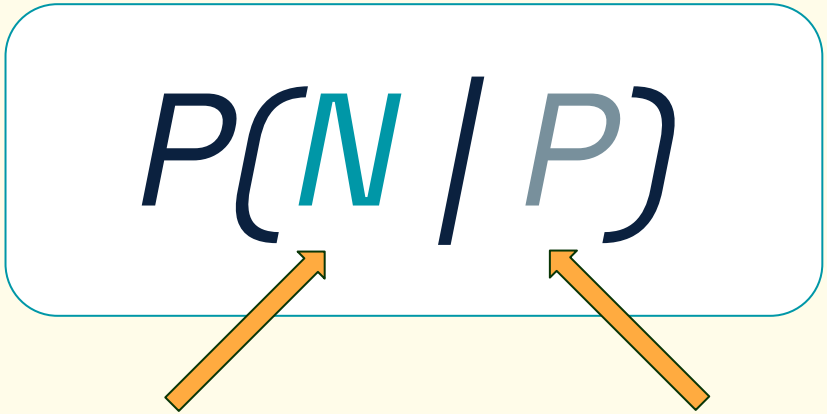
- Type
- Term of proposition's type
- Function type
- ...

Example

```
-- As a logic theorem:
theorem identity_proof (A : Prop) : A → A :=
  fun (h : A) => h

-- As a programming function:
def identity_program (α : Type) : α → α :=
  fun (x : α) => x
```

Theorem Prover: LLM + Lean



Section 1

Foreground

Motivation

Goals

Lean theorem prover that is:

- Open
- Extensible
- Performant
- Future proof

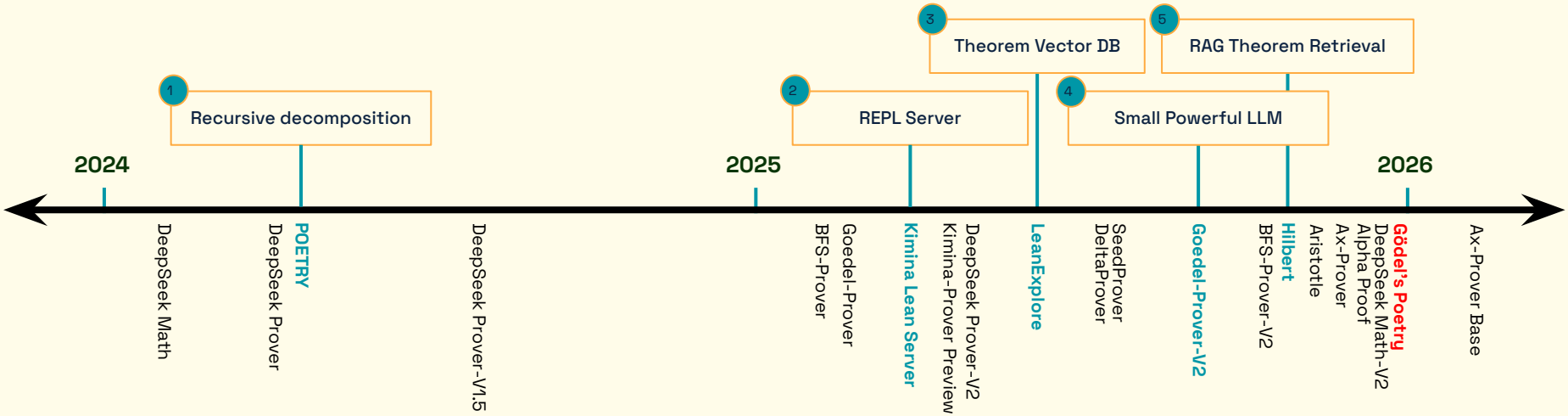
Design Constraints

- Single developer
- No training costs
- Rapid model releases
- Limited model contexts
- Model running costs
- LLM hallucinations

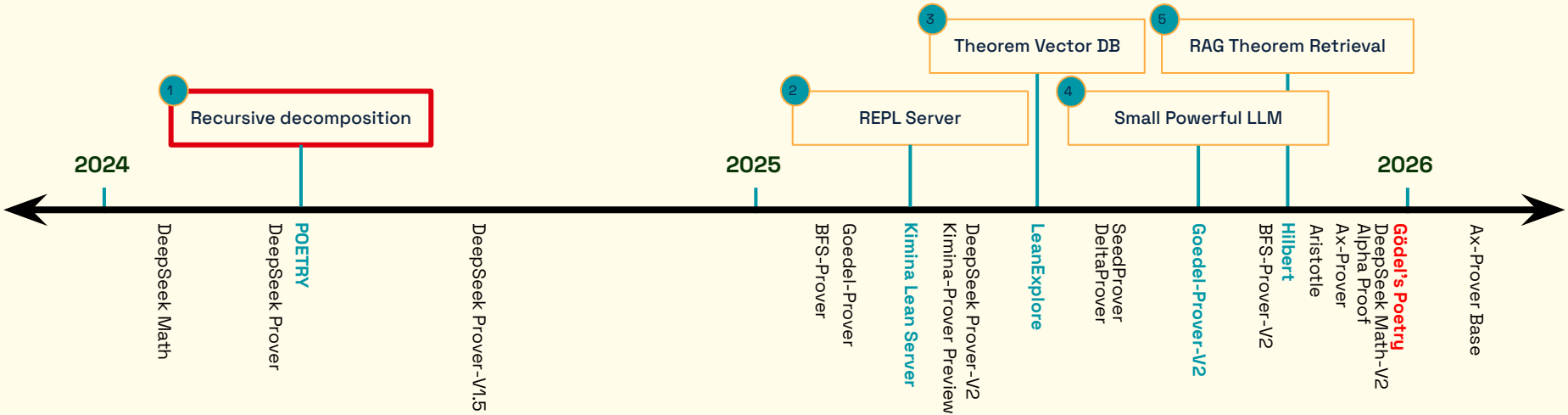
Section 2

Past Work

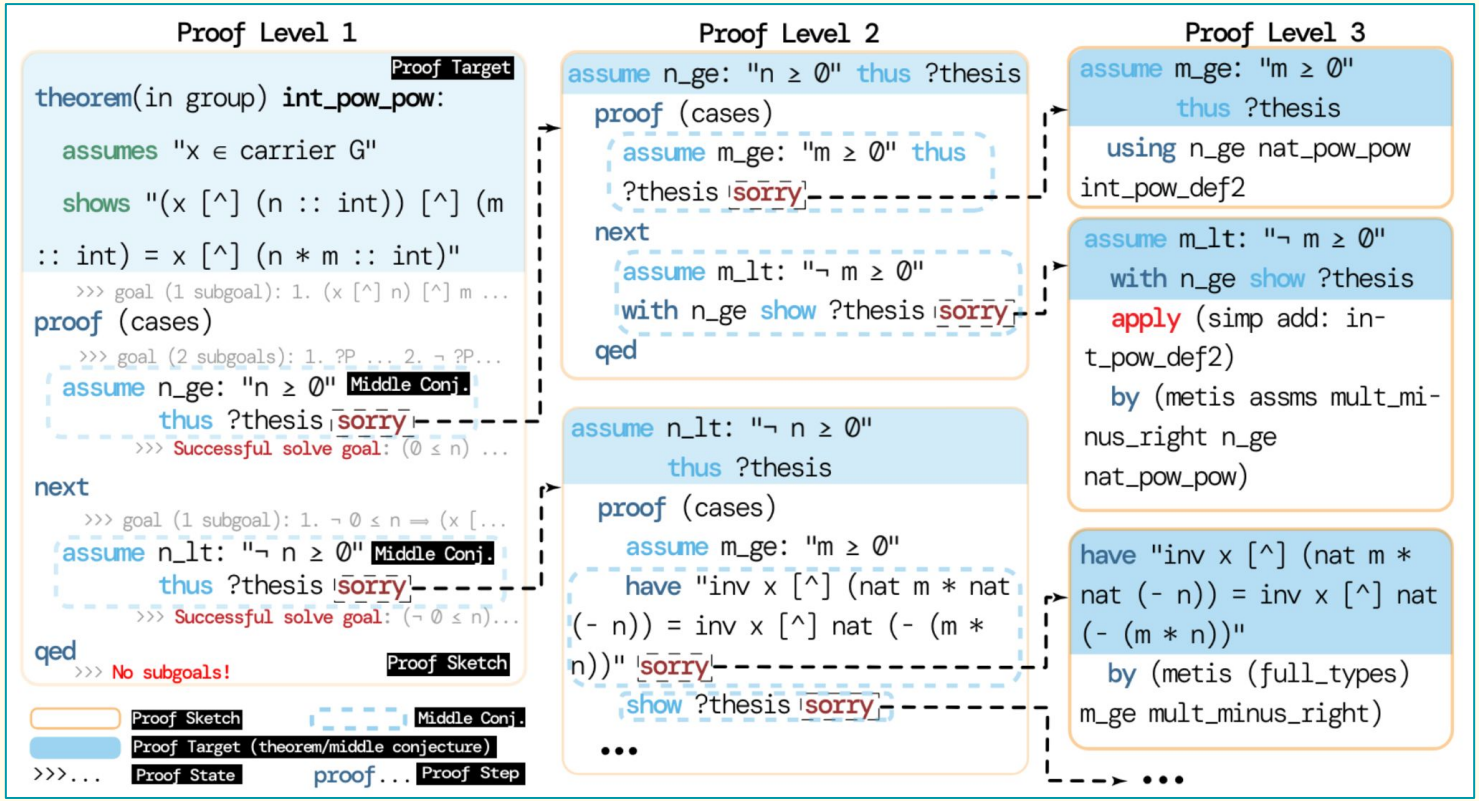
Related Work



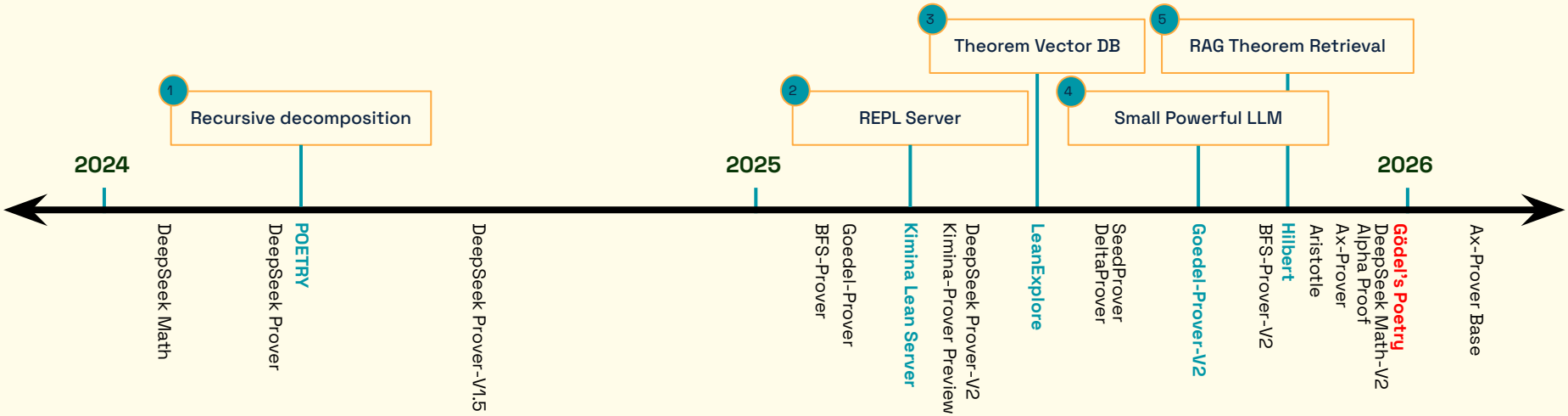
Related Work: Recursive Decomposition



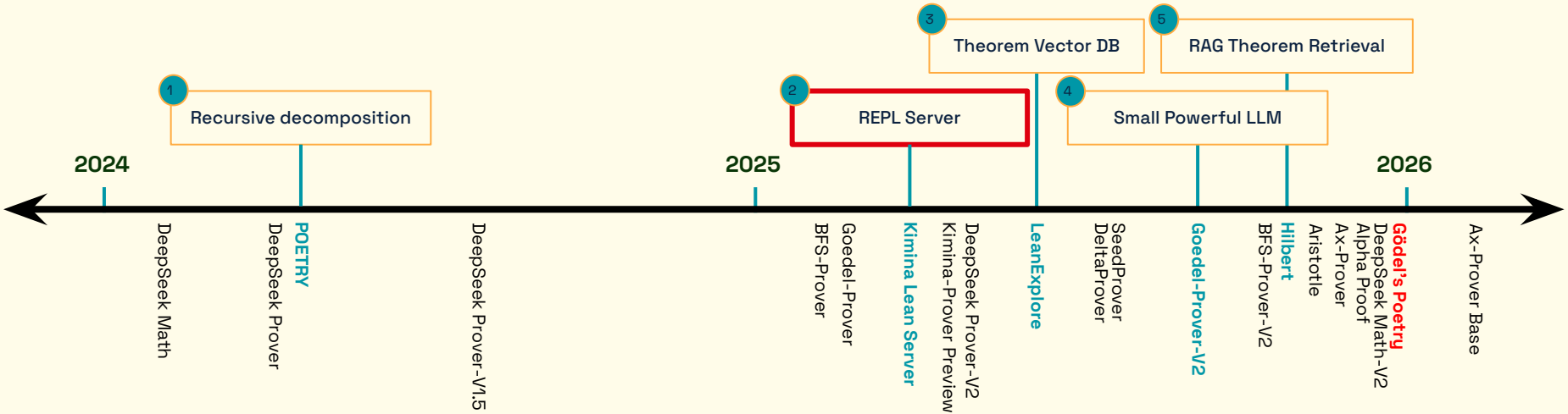
Related Work: Recursive Decomposition



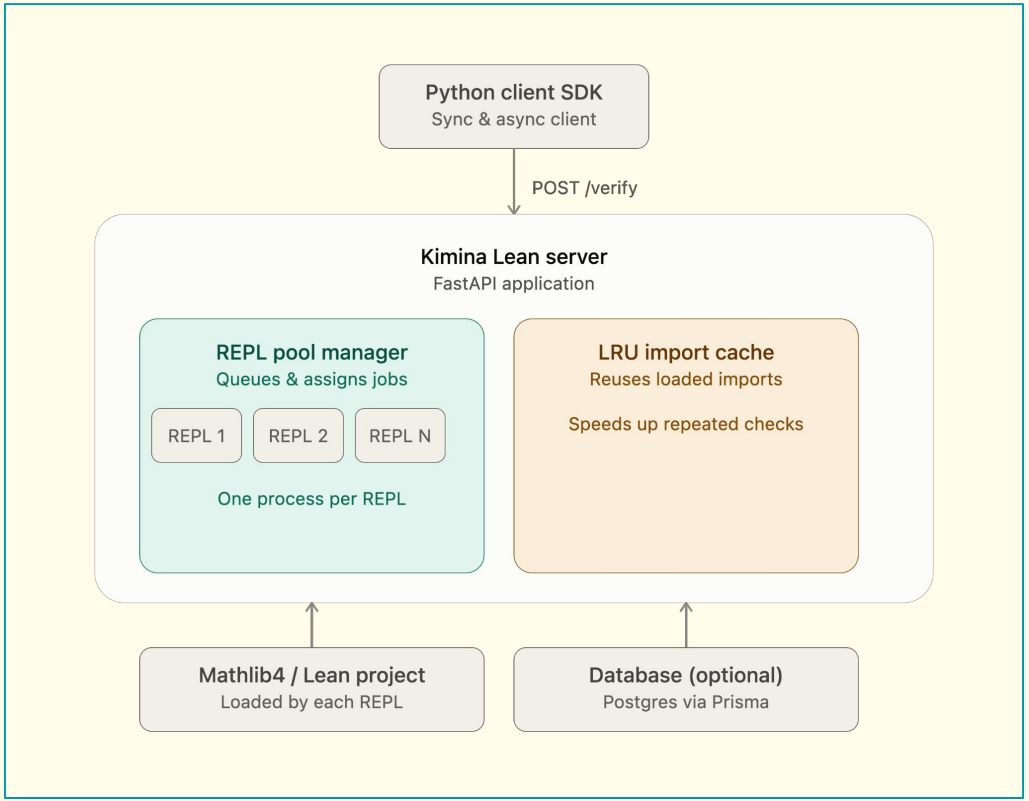
Related Work



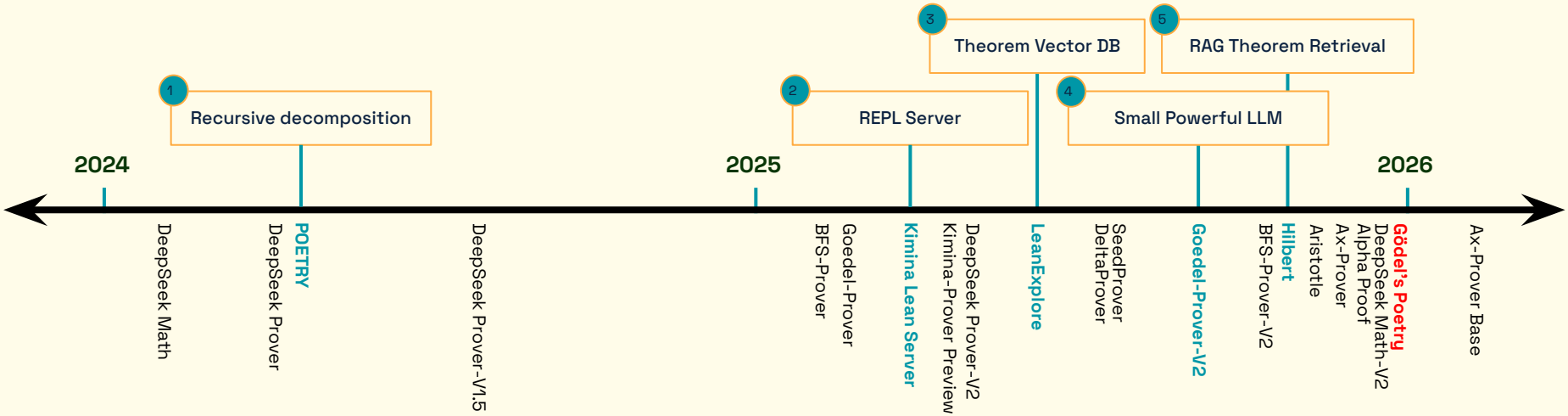
Related Work: REPL Server



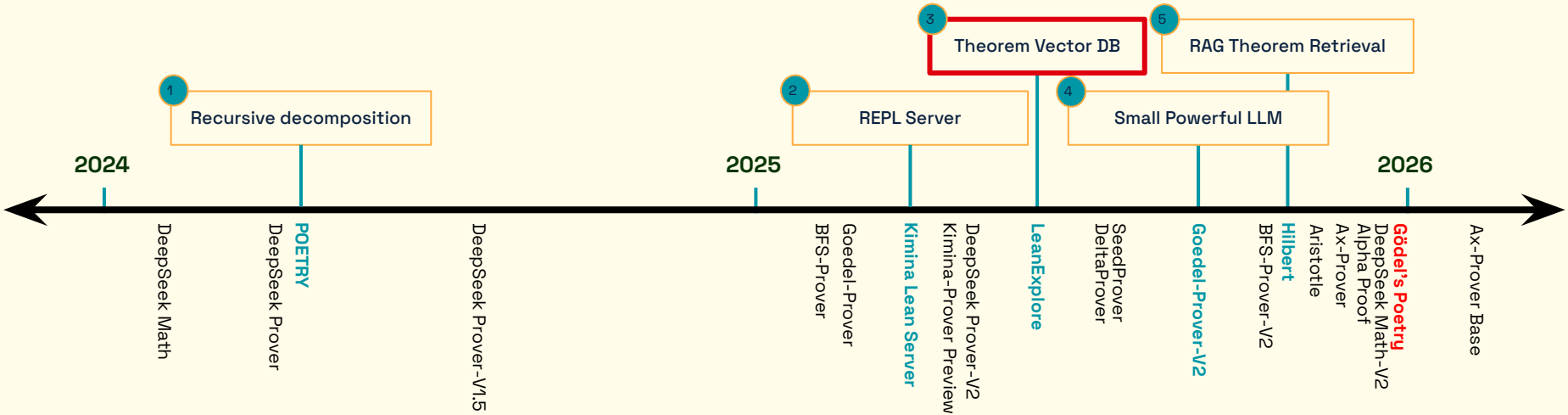
Related Work: REPL Server



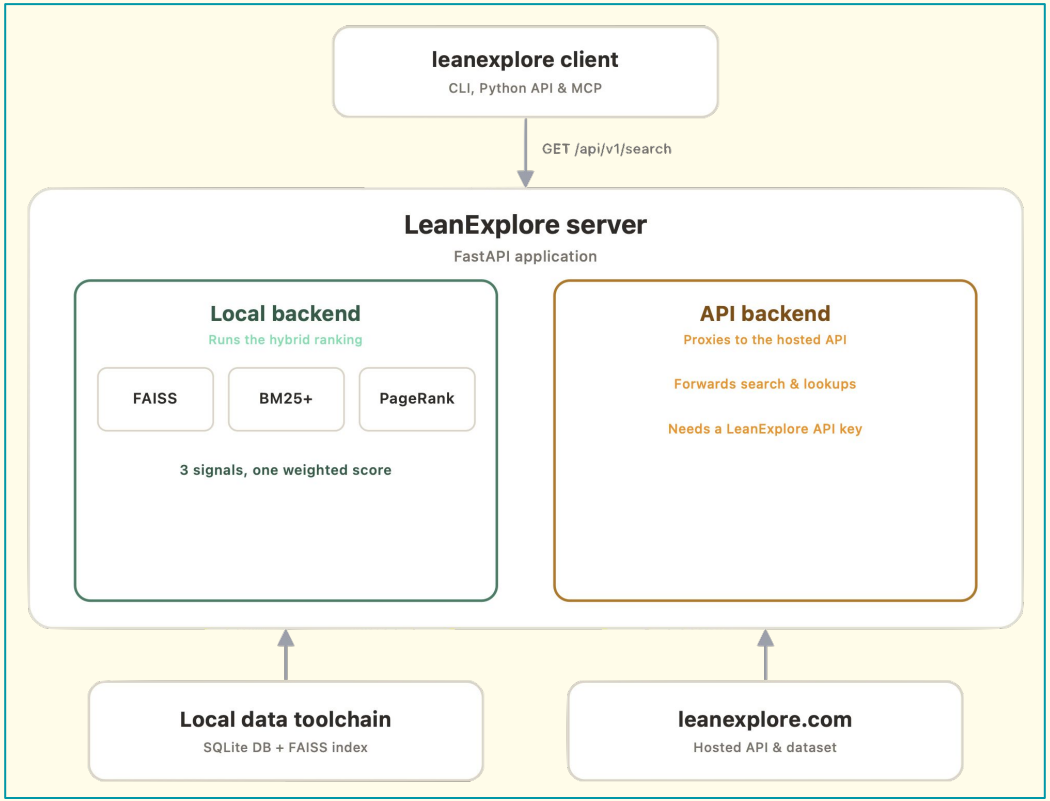
Related Work



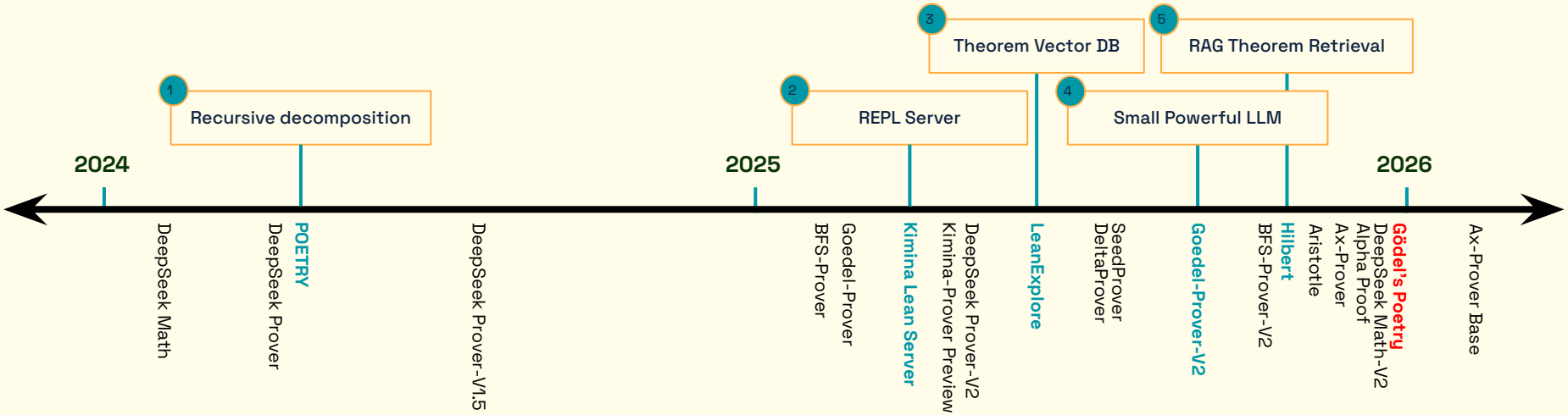
Related Work: Theorem Vector DB



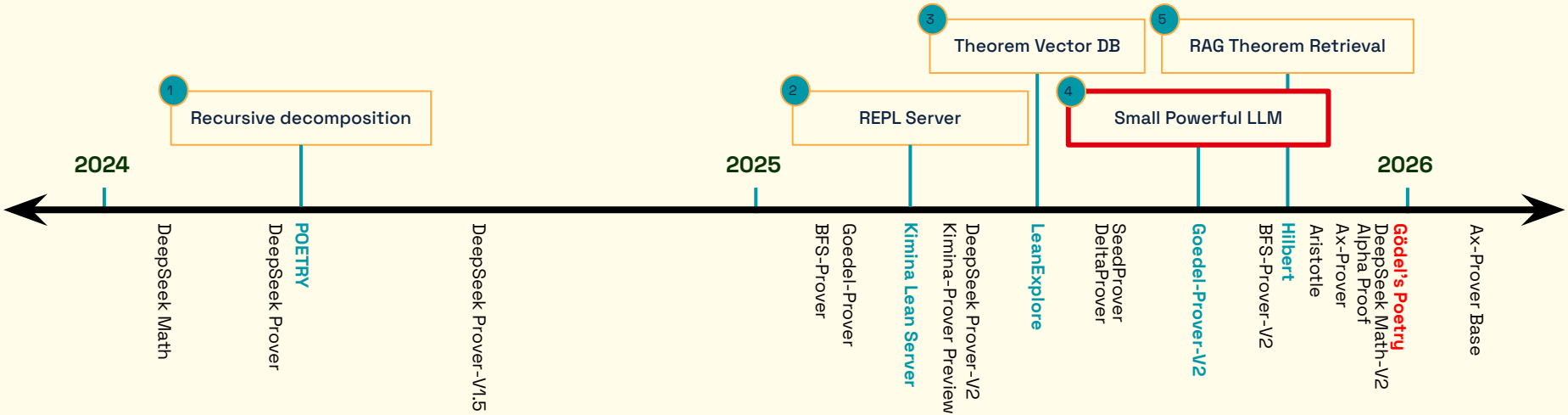
Related Work: Theorem Vector DB



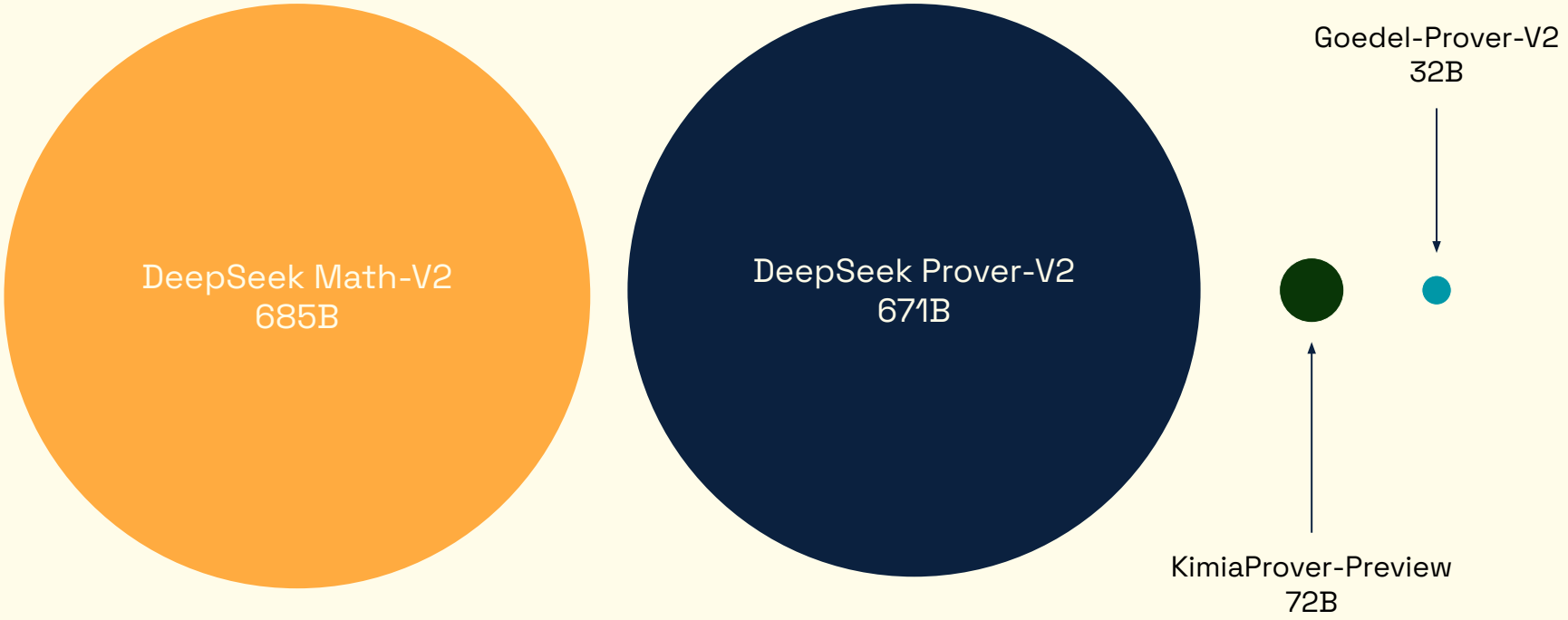
Related Work



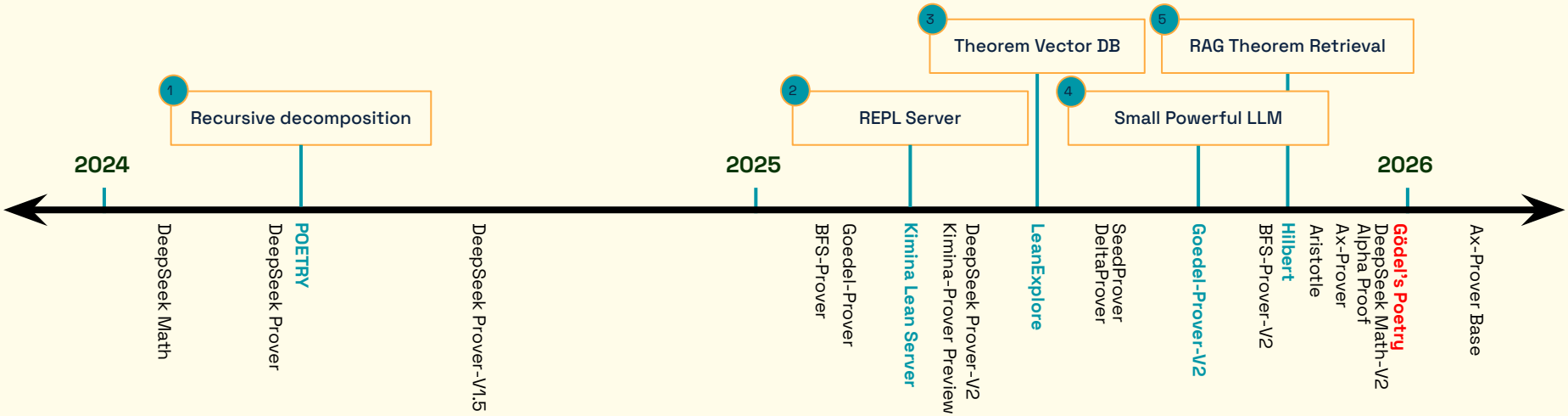
Related Work: Small Powerful LLM



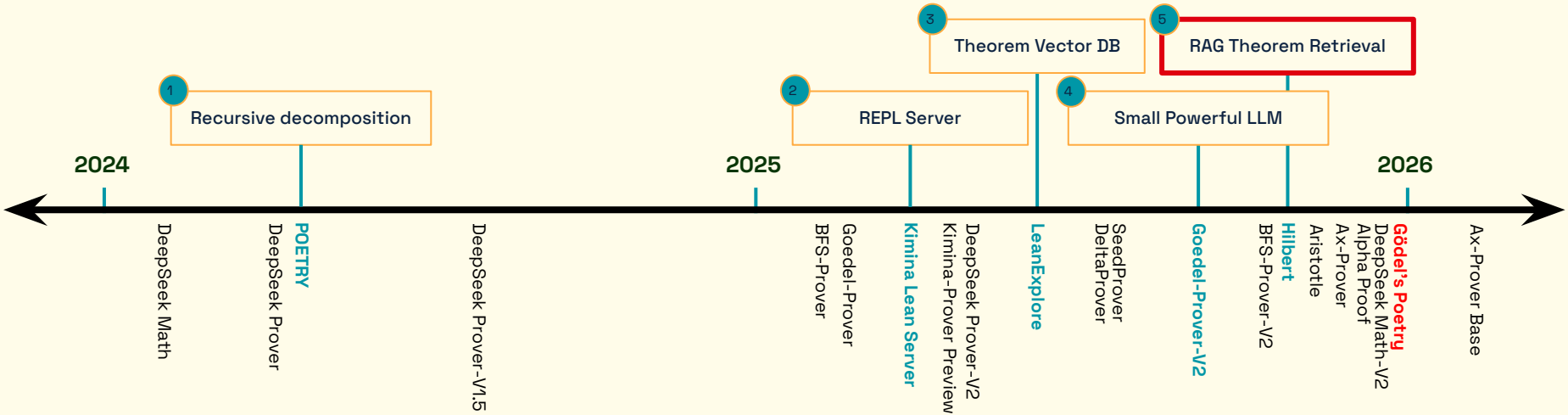
Related Work: Small Powerful LLM



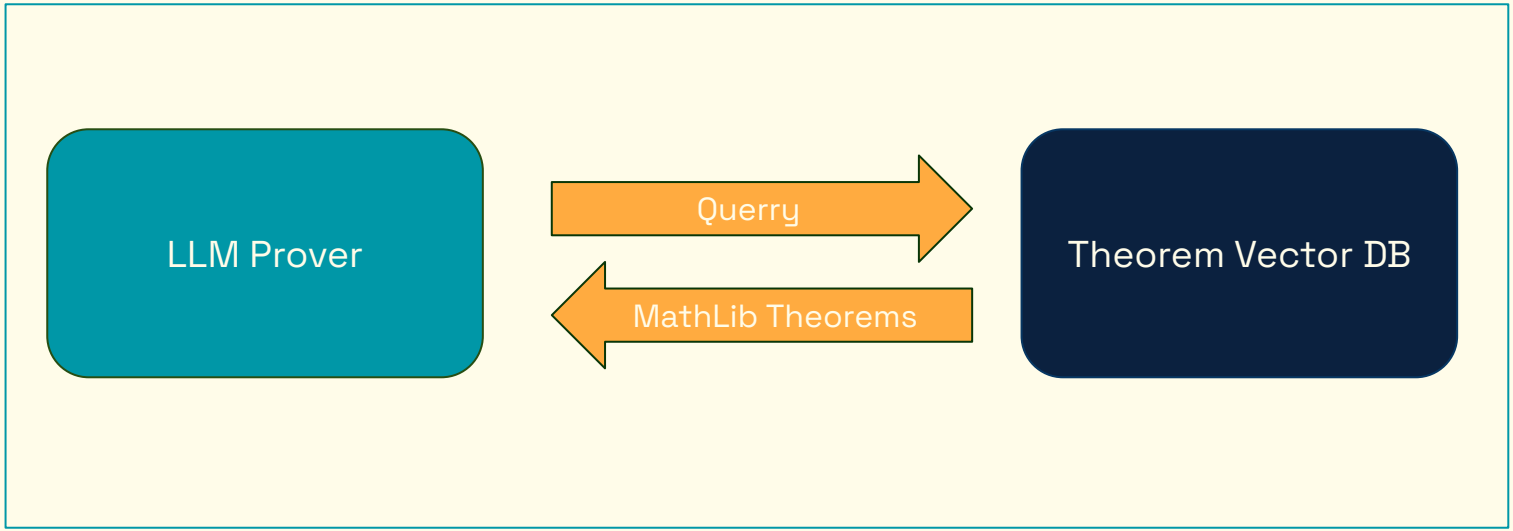
Related Work



Related Work: RAG Theorem Retrieval



Related Work: RAG Theorem Retrieval



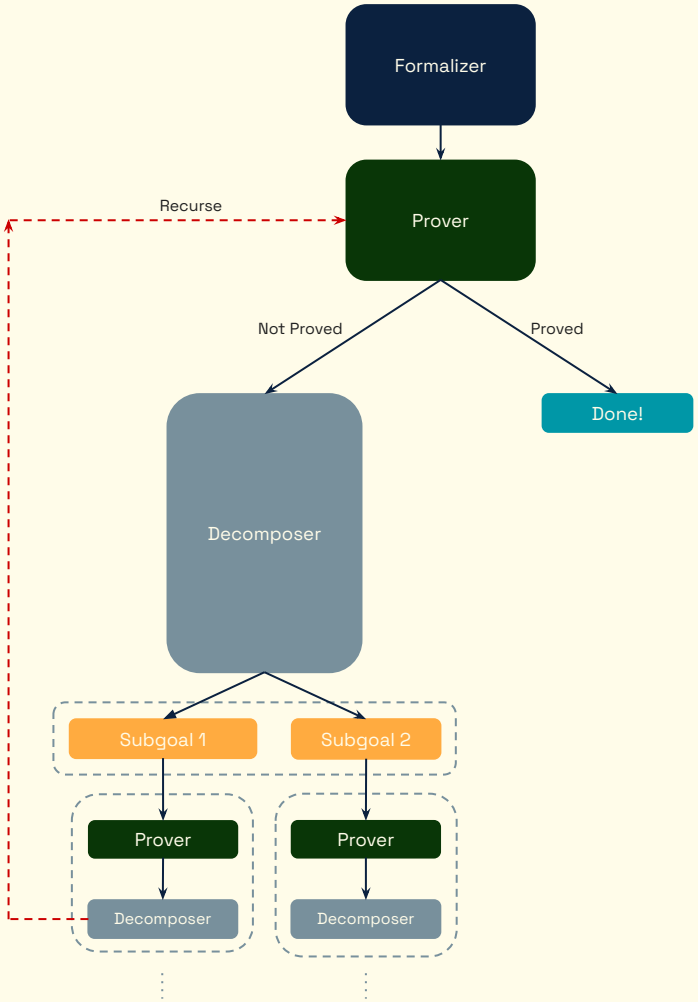
Section 3

Methods

Section 3.1

Architecture

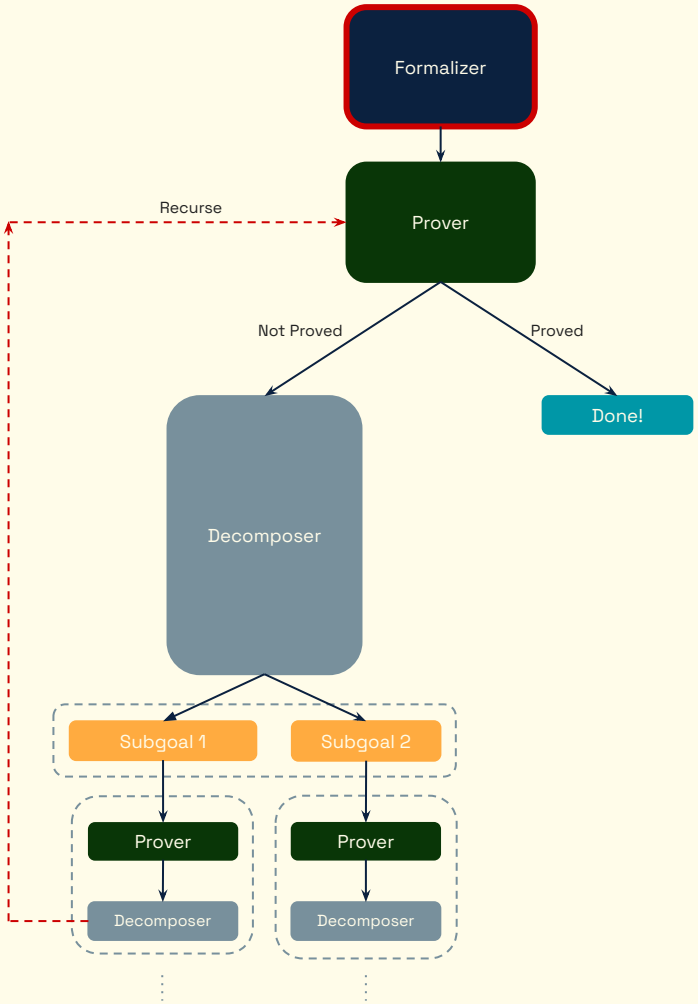
Formalizer	Formalizes informal theorems
Prover	Proves formal theorems
Decomposer	Decomposes formal theorems



Section 3.1

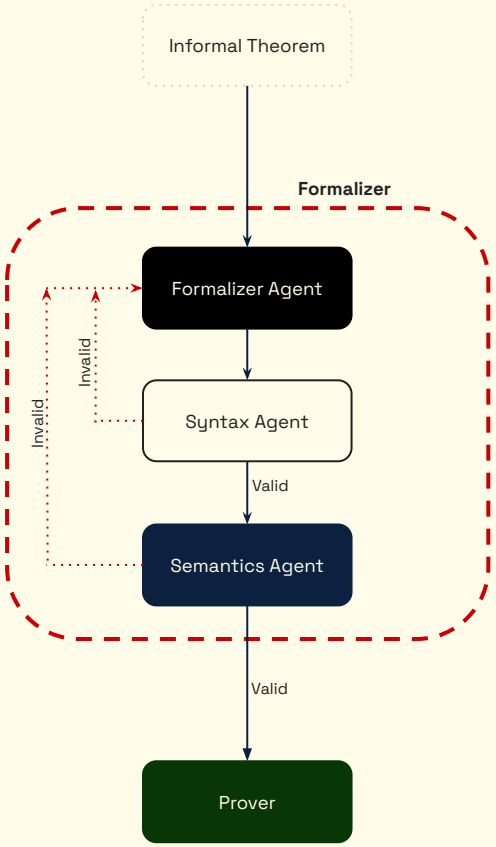
Architecture

Formalizer	Formalizes informal theorems
Prover	Proves formal theorems
Decomposer	Decomposes formal theorems



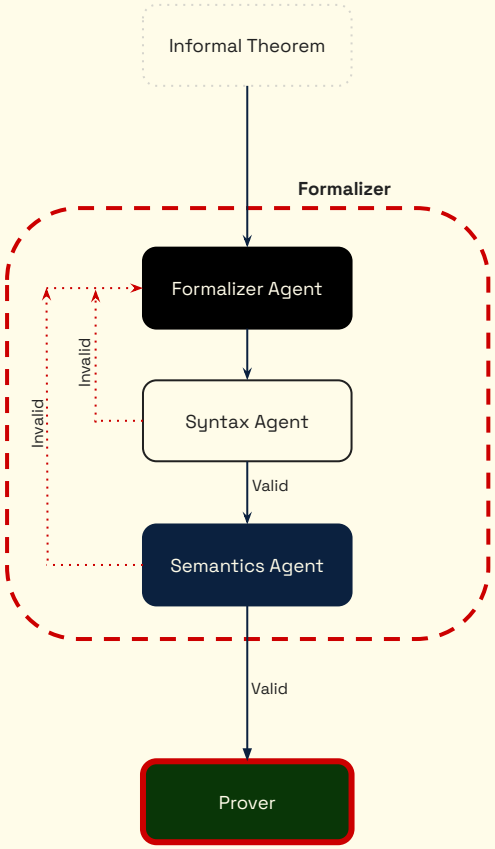
Architecture: Formalizer

Formalizer Agent	LLM that formalizes informal theorems
Syntax Agent	Kimina Lean Server variant that checks the syntax of formal theorems
Semantics Agent	LLM checks formal and informal theorems are semantically equivalent



Architecture: Formalizer

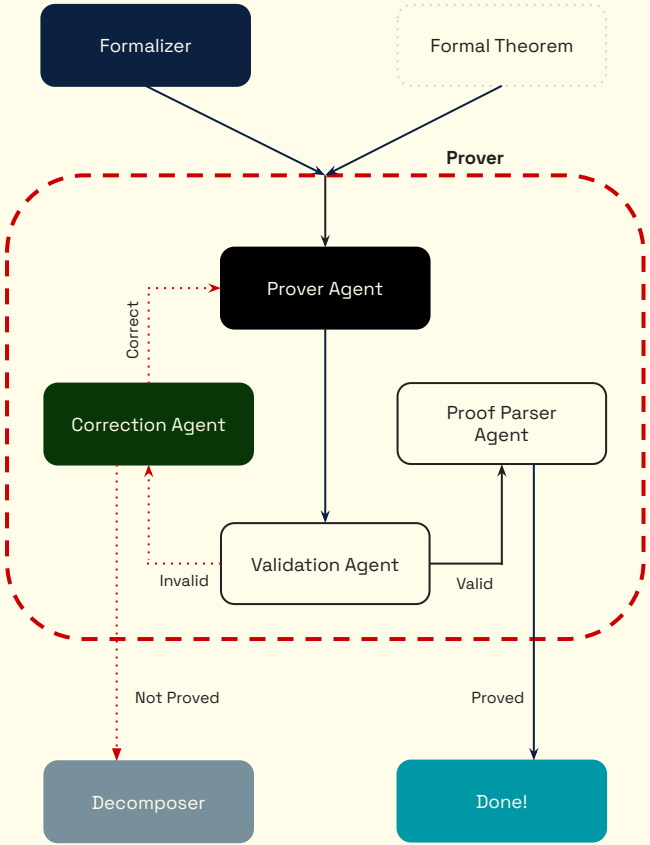
Formalizer Agent	LLM that formalizes informal theorems
Syntax Agent	Kimina Lean Server variant that checks the syntax of formal theorems
Semantics Agent	LLM checks formal and informal theorems are semantically equivalent



Section | 3.3

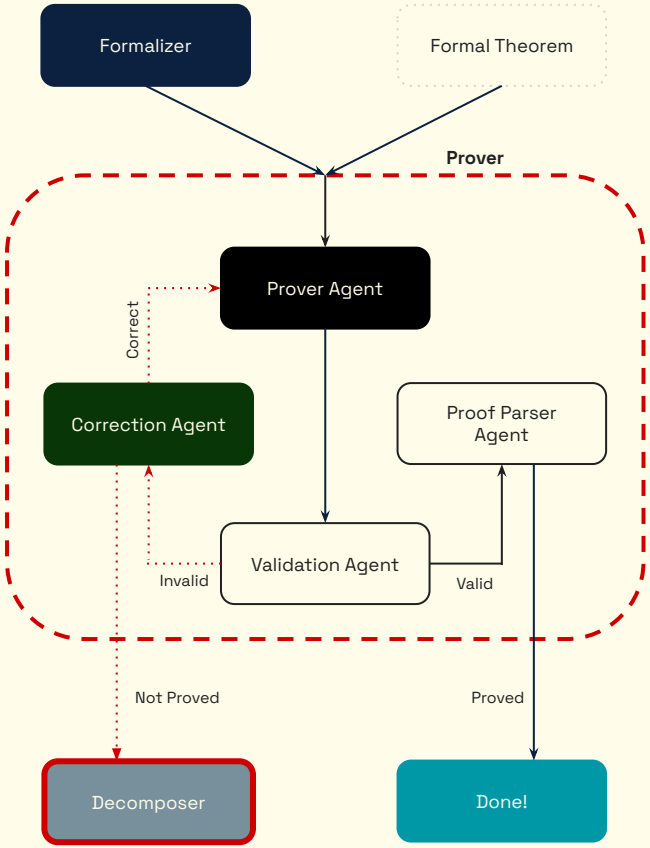
Architecture: Prover

Prover Agent	LLM that proves formal theorems
Validation Agent	Kimina Lean Server variant that checks if a theorem is proven
Correction Agent	Agent takes the Validation Agent output and creates a correction request
Proof Parser Agent	Kimina Lean Server variant that generates a theorem's AST



Architecture: Prover

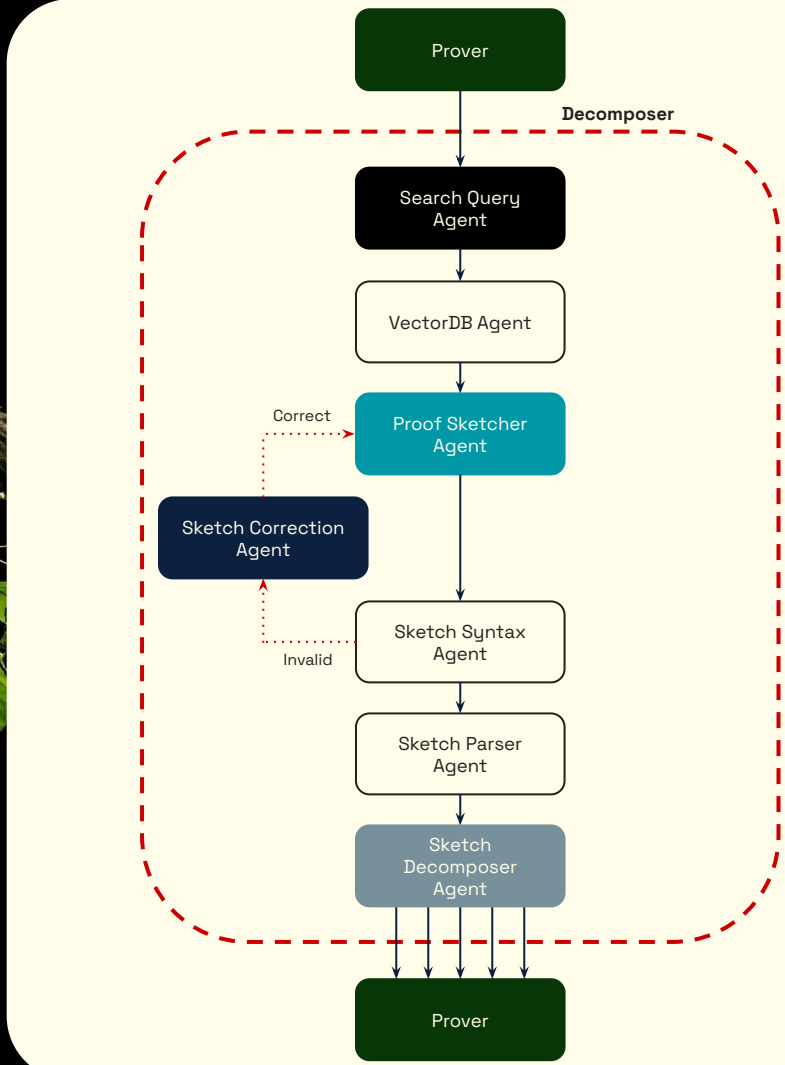
Prover Agent	LLM that proves formal theorems
Validation Agent	Kimina Lean Server variant that checks if a theorem is proven
Correction Agent	Agent takes the Validation Agent output and creates a correction request
Proof Parser Agent	Kimina Lean Server variant that generates a theorem's AST



Section 3.4

Architecture: Decomposer

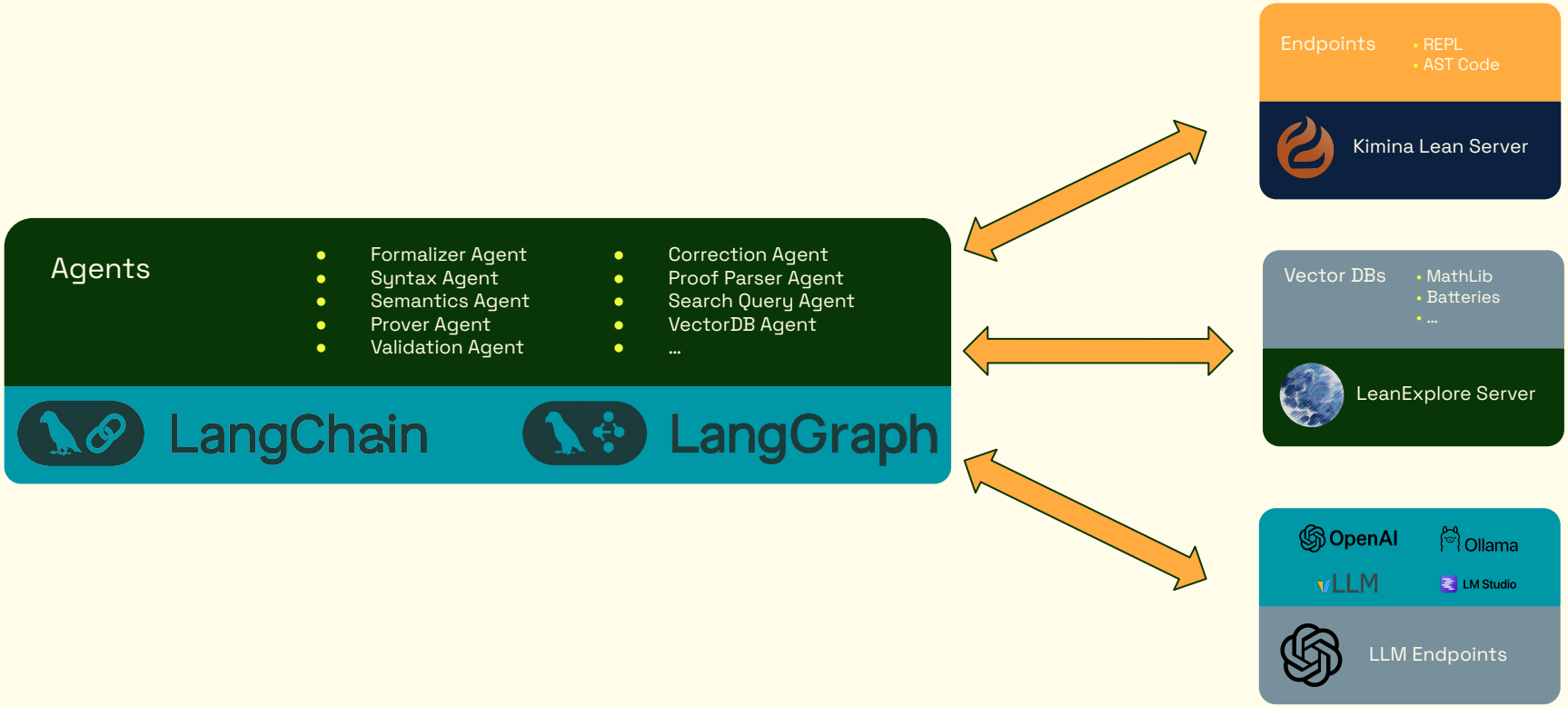
Search Query Agent	LLM that generates vector DB search queries
VectorDB Agent	LeanExplore variant containing MathLib vector DB
Proof Sketcher Agent	LLM that decomposes a theorem into sorry proven have statements
Sketch Syntax Agent	Kimina Lean Server variant that checks a sketch's syntax
Sketch Correction Agent	Takes the last agent's output and creates a correction request
Sketch Parser Agent	Kimina Lean Server variant that generates a sketch's AST
Sketch Decomposer Agent	Agent that decomposes a sketch into independent formal theorems



Section 4

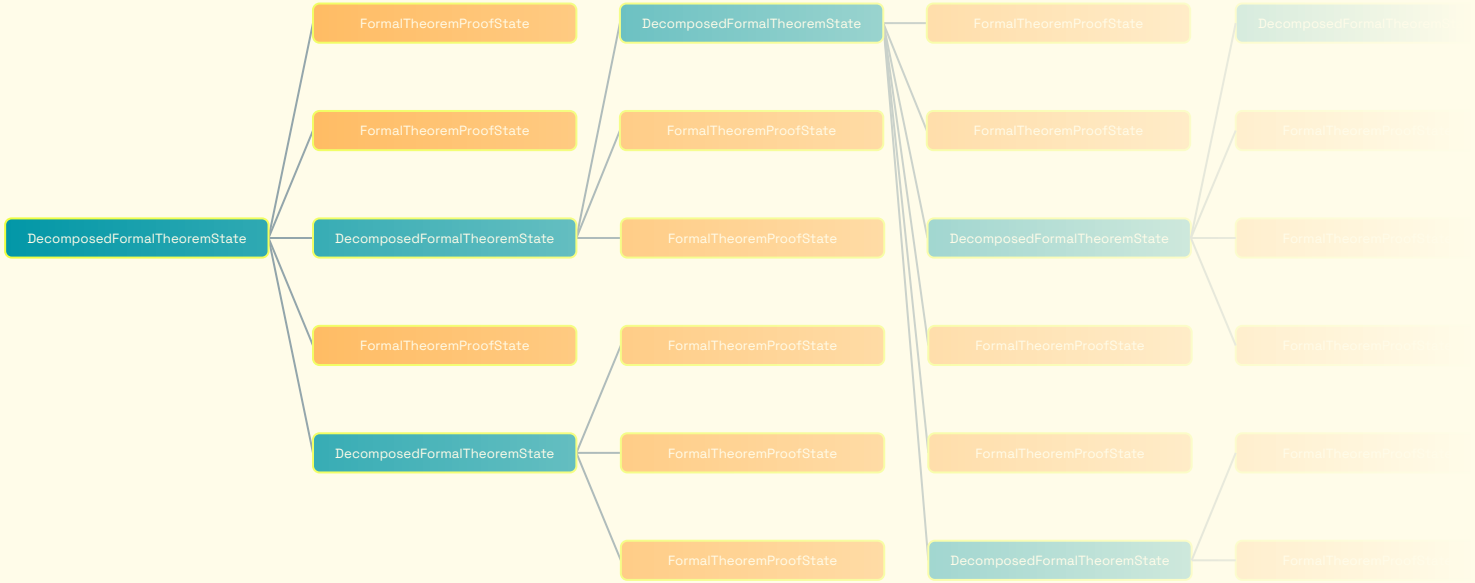
Implementation

Implementation: Technologies



Section 4.2

Implementation: Limited Model Context



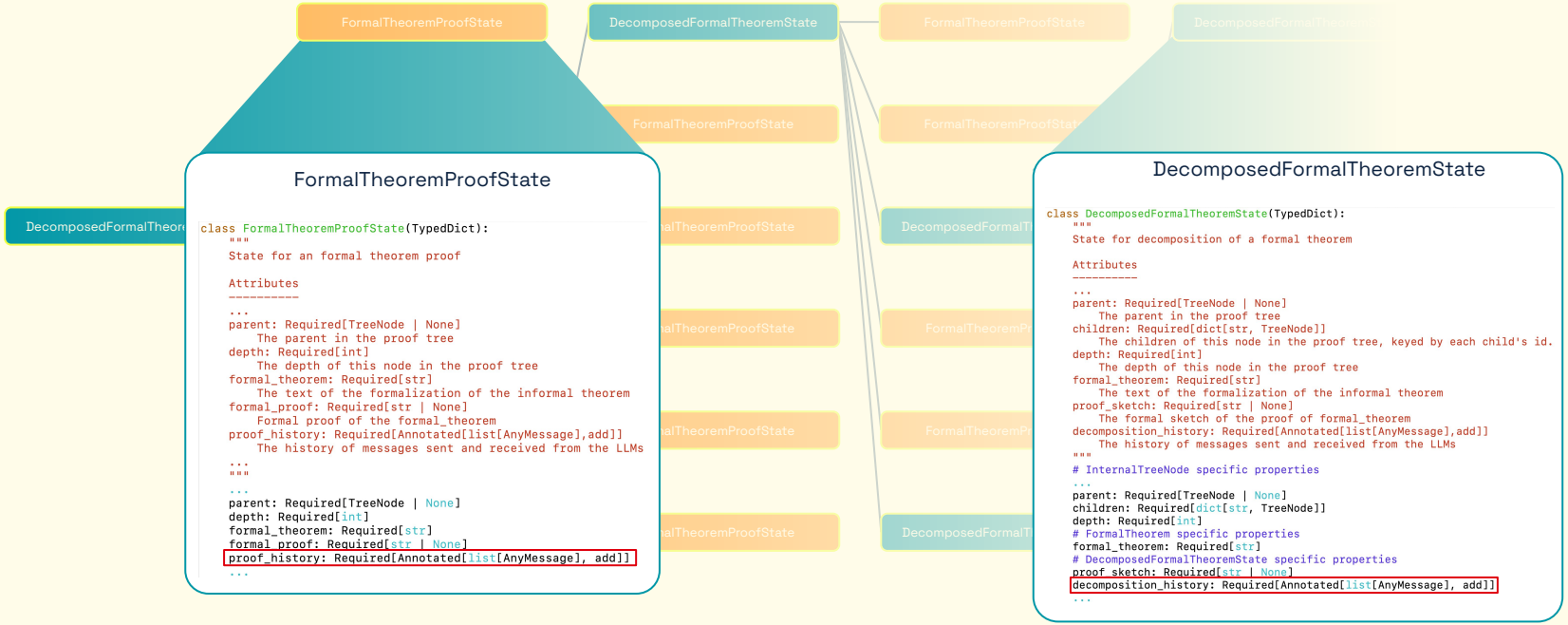
Section 4.2

Implementation: Limited Model Context


```python
class FormalTheoremProofState(TypedDict):
    """
    State for an formal theorem proof

    Attributes
    ----------
    ...
    parent: Required[TreeNode | None]
        The parent in the proof tree
    depth: Required[int]
        The depth of this node in the proof tree
    formal_theorem: Required[str]
        The text of the formalization of the informal theorem
    formal_proof: Required[str | None]
        Formal proof of the formal_theorem
    proof_history: Required[Annotated[list[AnyMessage],add]]
        The history of messages sent and received from the LLMs
    ...
    """
    ...
    parent: Required[TreeNode | None]
    depth: Required[int]
    formal_theorem: Required[str]
    formal_proof: Required[str | None]
    proof_history: Required[Annotated[list[AnyMessage], add]]
    ...
```

Implementation: Limited Model Context



Implementation: Extensibility

config.ini

```
[FORMALIZER_AGENT_LLM]
model = kdavis/goedel-formalizer-v2:32b
# provider can be: ollama, vllm, lmstudio, or openai
provider = ollama
url = http://localhost:11434/v1
max_tokens = 50000
num_ctx = 40960
max_retries = 10
max_remote_retries = 5
# Optional vLLM-specific parameters (ignored by Ollama/LM Studio/OpenAI)
# use_beam_search = false
# best_of = 1
# top_k = -1
# repetition_penalty = 1.0
# length_penalty = 1.0
# Optional LM Studio parameters (ignored by Ollama/vLLM/OpenAI)
# ttl = 300

[PROVER_AGENT_LLM]
model = kdavis/Goedel-Prover-V2:32b
# provider can be: ollama, vllm, lmstudio, or openai
provider = ollama
url = http://localhost:11434/v1
max_tokens = 50000
num_ctx = 40960
max_self_correction_attempts = 2
max_depth = 20
max_pass = 32
max_remote_retries = 5
# Optional vLLM-specific parameters (ignored by Ollama/LM Studio/OpenAI)
# use_beam_search = false
# best_of = 1
# top_k = -1
```

Implementation: Extensibility

config.ini

```
[FORMALIZER_AGENT_LLM]
model = kdavis/goedel-formalizer-v2:32b
# provider can be: ollama, vllm, lmstudio, or openai
provider = ollama
url = http://localhost:11434/v1
max_tokens = 50000
num_ctx = 40960
max_retries = 10
max_remote_retries = 5
# Optional vLLM-specific parameters (ignored by Ollama/LM Studio/OpenAI)
# use_beam_search = false
# best_of = 1
# top_k = -1
# repetition_penalty = 1.0
# length_penalty = 1.0
# Optional LM Studio parameters (ignored by Ollama/vLLM/OpenAI)
# ttl = 300

[PROVER_AGENT_LLM]
model = kdavis/Goedel-Prover-V2:32b
# provider can be: ollama, vllm, lmstudio, or openai
provider = ollama
url = http://localhost:11434/v1
max_tokens = 50000
num_ctx = 40960
max_self_correction_attempts = 2
max_depth = 20
max_pass = 32
max_remote_retries = 5
# Optional vLLM-specific parameters (ignored by Ollama/LM Studio/OpenAI)
# use_beam_search = false
# best_of = 1
# top_k = -1
```

Implementation: Extensibility

config.ini

```
[FORMALIZER_AGENT_LLM]
model = kdavis/goedel-formalizer-v2:32b
# provider can be: ollama, vllm, lmstudio, or openai
provider = ollama
url = http://localhost:11434/v1
max_tokens = 50000
num_ctx = 40960
max_retries = 10
max_remote_retries = 5
# Optional vLLM-specific parameters (ignored by Ollama/LM Studio/OpenAI)
# use_beam_search = false
# best_of = 1
# top_k = -1
# repetition_penalty = 1.0
# length_penalty = 1.0
# Optional LM Studio parameters (ignored by Ollama/vLLM/OpenAI)
# ttl = 300

[PROVER_AGENT_LLM]
model = kdavis/Goedel-Prover-V2:32b
# provider can be: ollama, vllm, lmstudio, or openai
provider = ollama
url = http://localhost:11434/v1
max_tokens = 50000
num_ctx = 40960
max_self_correction_attempts = 2
max_depth = 20
max_pass = 32
max_remote_retries = 5
# Optional vLLM-specific parameters (ignored by Ollama/LM Studio/OpenAI)
# use_beam_search = false
# best_of = 1
# top_k = -1
```

Implementation: Extensibility

config.ini

```
[FORMALIZER_AGENT_LLM]
model = kdavis/goedel-formalizer-v2:32b
# provider can be: ollama, vllm, lmstudio, or openai
provider = ollama
url = http://localhost:11434/v1
max_tokens = 50000
num_ctx = 40960
max_retries = 10
max_remote_retries = 5
# Optional vLLM-specific parameters (ignored by Ollama/LM Studio/OpenAI)
# use_beam_search = false
# best_of = 1
# top_k = -1
# repetition_penalty = 1.0
# length_penalty = 1.0
# Optional LM Studio parameters (ignored by Ollama/vLLM/OpenAI)
# ttl = 300

[PROVER_AGENT_LLM]
model = kdavis/Goedel-Prover-V2:32b
# provider can be: ollama, vllm, lmstudio, or openai
provider = ollama
url = http://localhost:11434/v1
max_tokens = 50000
num_ctx = 40960
max_self_correction_attempts = 2
max_depth = 20
max_pass = 32
max_remote_retries = 5
# Optional vLLM-specific parameters (ignored by Ollama/LM Studio/OpenAI)
# use_beam_search = false
# best_of = 1
# top_k = -1
```

Implementation: Extensibility

config.ini

```
[FORMALIZER_AGENT_LLM]
model = kdavis/goedel-formalizer-v2:32b
# provider can be: ollama, vllm, lmstudio, or openai
provider = ollama
url = http://localhost:11434/v1
max_tokens = 50000
num_ctx = 40960
max_retries = 10
max_remote_retries = 5
# Optional vLLM-specific parameters (ignored by Ollama/LM Studio/OpenAI)
# use_beam_search = false
# best_of = 1
# top_k = -1
# repetition_penalty = 1.0
# length_penalty = 1.0
# Optional LM Studio parameters (ignored by Ollama/vLLM/OpenAI)
# ttl = 300

[PROVER_AGENT_LLM]
model = kdavis/Goedel-Prover-V2:32b
# provider can be: ollama, vllm, lmstudio, or openai
provider = ollama
url = http://localhost:11434/v1
max_tokens = 50000
num_ctx = 40960
max_self_correction_attempts = 2
max_depth = 20
max_pass = 32
max_remote_retries = 5
# Optional vLLM-specific parameters (ignored by Ollama/LM Studio/OpenAI)
# use_beam_search = false
# best_of = 1
# top_k = -1
```

Implementation: Extensibility

Environment Variables

```
export FORMALIZER_AGENT_LLM__PROVIDER="lmstudio"
export FORMALIZER_AGENT_LLM__URL="http://localhost:1234/v1"
export FORMALIZER_AGENT_LLM__MODEL="goedel-formalizer-v2-32b-i1"

export PROVER_AGENT_LLM__PROVIDER="lmstudio"
export PROVER_AGENT_LLM__URL="http://localhost:1234/v1"
export PROVER_AGENT_LLM__MODEL="mradermacher/Goedel-Prover-V2-32B-GGUF"

export SEMANTICS_AGENT_LLM__PROVIDER="lmstudio"
export SEMANTICS_AGENT_LLM__URL="http://localhost:1234/v1"
export SEMANTICS_AGENT_LLM__MODEL="lmstudio-community/Qwen3-30B-A3B-Instruct-2507-GGUF"

export SEARCH_QUERY_AGENT_LLM__PROVIDER="lmstudio"
export SEARCH_QUERY_AGENT_LLM__URL="http://localhost:1234/v1"
export SEARCH_QUERY_AGENT_LLM__MODEL="lmstudio-community/Qwen3-30B-A3B-Instruct-2507-GGUF"
```

Implementation: Extensibility

Environment Variables

```
export FORMALIZER_AGENT_LLM__PROVIDER="lmstudio"
export FORMALIZER_AGENT_LLM__URL="http://localhost:1234/v1"
export FORMALIZER_AGENT_LLM__MODEL="goedel-formalizer-v2-32b-i1"

export PROVER_AGENT_LLM__PROVIDER="lmstudio"
export PROVER_AGENT_LLM__URL="http://localhost:1234/v1"
export PROVER_AGENT_LLM__MODEL="mradermacher/Goedel-Prover-V2-32B-GGUF"

export SEMANTICS_AGENT_LLM__PROVIDER="lmstudio"
export SEMANTICS_AGENT_LLM__URL="http://localhost:1234/v1"
export SEMANTICS_AGENT_LLM__MODEL="lmstudio-community/Qwen3-30B-A3B-Instruct-2507-GGUF"

export SEARCH_QUERY_AGENT_LLM__PROVIDER="lmstudio"
export SEARCH_QUERY_AGENT_LLM__URL="http://localhost:1234/v1"
export SEARCH_QUERY_AGENT_LLM__MODEL="lmstudio-community/Qwen3-30B-A3B-Instruct-2507-GGUF"
```

Section 5

Examples

Formalization: Informal to Formal

“Prove that for any natural numbers m and n, if m is even and n is even, then m + n is even.”

```
theorem theorem_b2f45cfb951a : ∀ m n : ℕ, Even m → Even n → Even (m + n) := by
  sorry
```

Proof: Without Decomposition

```
import Mathlib
import Aesop

set_option maxHeartbeats 0

open BigOperators Real Nat Topology Rat

theorem theorem_b2f45cfb951a : ∀ m n : ℕ, Even m → Even n → Even (m + n) := by
  intro m n hm hn
  have h_main : Even (m + n) := by
    -- Extract the witnesses for m and n being even
    cases' hm with a ha
    cases' hn with b hb
    -- Rewrite m and n as 2a and 2b respectively
    rw [ha, hb]
    -- Factor out the 2 from the sum 2a + 2b
    use a + b
    <;> ring
    <;> simp
    <;> ring
  -- The main result follows directly from the above steps
  exact h_main
```

Proof: With Decomposition

```
import Mathlib
import Aesop

set_option maxHeartbeats 0

open BigOperators Real Nat Topology Rat

theorem infinitude_of_primes : ∀ n : ℕ, ∃ p, p > n ∧ Nat.Prime p := by
  -- Step 1: Define product of all primes ≤ n
  have prod_primes_def :
      ∀ n, ∃ P, P = ∏ p ∈ Finset.filter Prime (Finset.range (n + 1)), p := by
    sorry

  -- Step 2: For given n, take P = product of all primes ≤ n
  have choose_P :
      ∀ n, ∃ P, P = ∏ p ∈ Finset.filter Nat.Prime (Finset.range (n + 1)), p := by
    sorry

  -- Step 3: Show that P + 1 has a prime divisor
  have prime_divisor_exists :
      ∀ n P, P = ∏ p ∈ Finset.filter Nat.Prime (Finset.range (n + 1)), p →
        ∃ q, Nat.Prime q ∧ q ∣ (P + 1) := by
    sorry

  -- Step 4: Show that such a prime divisor q is greater than n
  have divisor_gt_n :
      ∀ n P q,
        P = ∏ p ∈ Finset.filter Nat.Prime (Finset.range (n + 1)), p →
          Nat.Prime q → q ∣ (P + 1) → q > n := by
    sorry

  -- Step 5: Combine to conclude existence of a prime > n
  have conclusion : ∀ n, ∃ p, p > n ∧ Nat.Prime p := by
    sorry

  exact conclusion
```

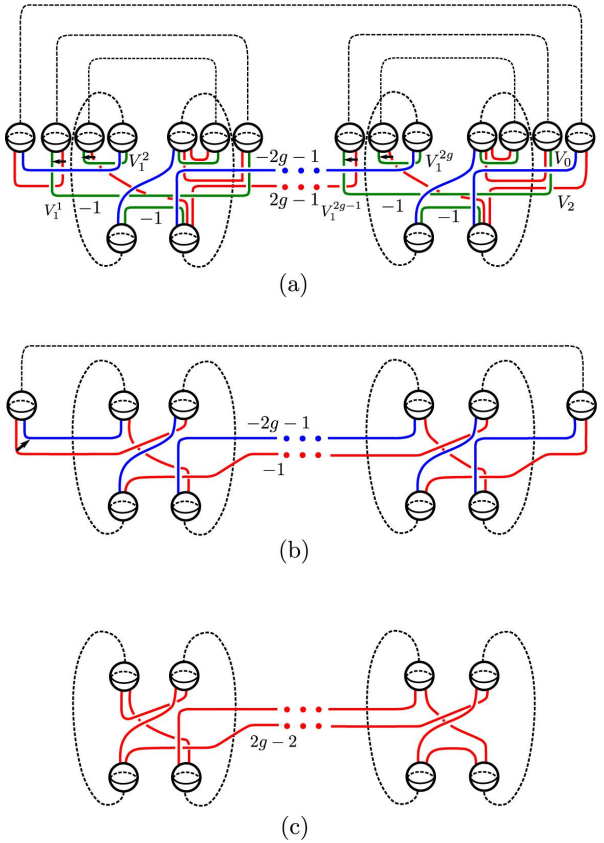
Section 6

Post Mortem

Section 6.1

Post Mortem

Rapid Model Releases	Protected against such, but failed to predict formalization performance improvements.
Model Running Costs	Model running costs seem less of a concern; people spend 100's to 1000s of euros on one run.
LLM Hallucinations	LLM hallucinations are still a problem; so Lean as a "source of truth" is still required.
Limited Model Contexts	Decomposition of LLM contexts across the proof tree allows for essentially infinite model context




Thank
You