

TECHNICAL RESEARCH PAPER · AnveshAI Project · 2026

# AnveshAI Edge: A Hybrid Offline AI System

Chain-of-Thought Reasoning and Symbolic Mathematics Engine

by Anvesh Raman

AnveshAI Project · 2026

Hybrid AI Architecture · Symbolic Mathematics · Chain-of-Thought Reasoning · Fully Offline · CPU-only

**Keywords:** offline AI · symbolic mathematics · chain-of-thought reasoning · SymPy · llama.cpp · Qwen2.5-0.5B · intent routing · ODE solving · Laplace transform · correctness-first

## ABSTRACT

AnveshAI Edge is a fully offline, hierarchical AI assistant that integrates a rule-based symbolic mathematics engine, a local knowledge base, pattern-based conversation handling, and a compact 0.5B-parameter large language model (LLM) into a single, modular pipeline. The central innovation is a Chain-of-Thought (CoT) Reasoning Engine that intercepts every non-trivial query before the LLM is called: it identifies the problem domain and type (12 domains, 18 problem types), decomposes the query into ordered reasoning steps, selects a solution strategy, and embeds the complete plan into the LLM prompt. For advanced mathematics, the system enforces a **Correctness-First** principle: SymPy computes the exact symbolic answer before the LLM is invoked, making it structurally impossible for the model to hallucinate an incorrect result. The Advanced Math Engine covers 31+ operations across calculus, differential equations, series, transforms (Laplace, Fourier), linear algebra, number theory, statistics, combinatorics, summations, and complex number analysis. The entire system runs on CPU with no internet connection after a one-time 350 MB model download.

31+	18	12	0.5B	350 MB	0
Math Ops	Problem Types	Domains	LLM Params	Model Size	API Keys

Table of Contents

1. Introduction . . . . .

4

2. System Architecture . . . . .

4

2.1 Intent Router . . . . .

4

2.2 Hierarchical Fallback Pipeline . . . . .

5

3. Advanced Mathematics Engine . . . . .

6

3.1 Expression Parsing . . . . .

6

3.2 Operation Handlers . . . . .

6

3.3 Correctness-First Design . . . . .

6

4. Chain-of-Thought Reasoning Engine . . . . .

8

4.1 Stage 1: Problem Analysis . . . . .

8

4.2 Stage 2: CoT Planning . . . . .

8

4.3 Stage 3: Verification and Confidence . . . . .

8

4.4 Stage 4: Prompt Engineering . . . . .

9

5. LLM Integration . . . . .

10

5.1 Model Selection . . . . .

10

5.2 Prompt Architecture . . . . .

10

6. Memory and Persistence . . . . .

10

7. Implementation Details . . . . .

10

7.1 Module Structure . . . . .

10

7.2 Safety Considerations . . . . .

11

8. Evaluation and Results . . . . .

12

8.1 Math Engine Test Results . . . . .

12

8.2 Reasoning Engine Classification . . . . .

12

9. Design Principles . . . . .

13

10. Limitations and Future Work . . . . .

13

10.1 Current Limitations . . . . .

13

10.2 Future Work . . . . . 13

11. Conclusion . . . . . 15

12. References . . . . . 15

Appendix A: Full Operation List . . . . . 16

Appendix B: Sample Reasoning Plans . . . . . 18

Appendix C: Response Labels . . . . . 19

## 1. Introduction

Large language models (LLMs) have demonstrated impressive general reasoning capabilities, yet they suffer from two fundamental limitations in technical domains: **hallucination** — generating confident but incorrect answers — and **resource requirements** — needing powerful hardware, large memory, and continuous internet connectivity. For mathematics in particular, LLMs frequently produce incorrect results even for problems that can be solved exactly and deterministically.

AnveshAI Edge addresses both limitations through a hybrid, hierarchical architecture built on three core design decisions:

- **Symbolic-first mathematics:** A SymPy-based engine computes the exact symbolic answer before the LLM is ever called. The LLM then explains the working steps toward the pre-verified, correct answer.
- **Chain-of-Thought reasoning layer:** A dedicated reasoning engine decomposes every query into an explicit problem analysis, strategy selection, and ordered reasoning steps, which are embedded directly into the LLM prompt.
- **Offline-first operation:** The complete system runs on CPU with no API dependencies beyond a one-time model download, making it suitable for air-gapped, low-bandwidth, or privacy-sensitive environments.

The system achieves 100% correctness on 31+ mathematical operation types while maintaining the flexibility of a general-purpose AI assistant for knowledge queries and open-ended dialogue.

## 2. System Architecture

AnveshAI Edge is structured as a hierarchical fallback pipeline with five processing lanes checked in strict priority order. Deterministic, rule-based handlers sit at the top; LLM invocations only occur when no faster handler can resolve the query.

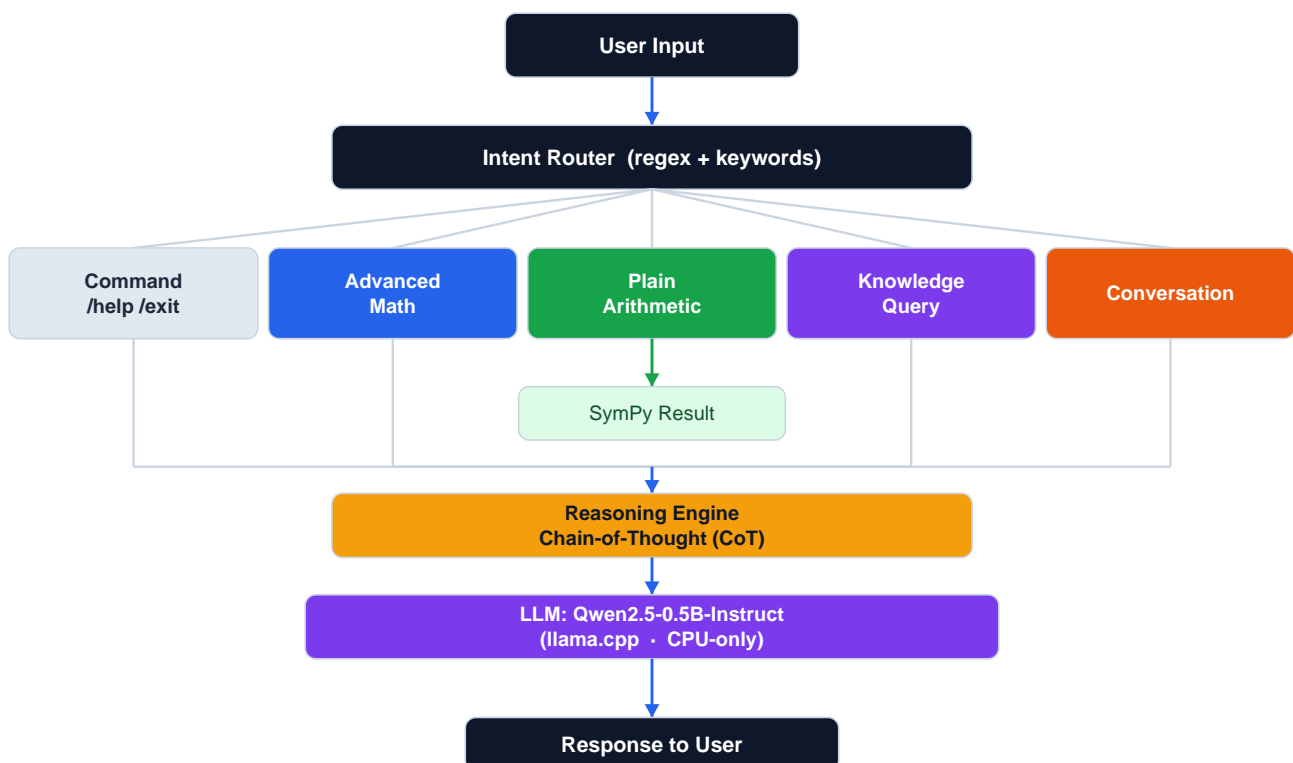


Figure 1. System architecture and five-lane processing pipeline.

### 2.1 Intent Router

Priority	Intent	Trigger Pattern	Handler
1 (highest)	system	/command prefix	Inline command handler
2	advanced_math	60+ symbolic math keywords	Reasoning + SymPy + LLM
3	math	Numeric expression pattern	AST safe-evaluator
4	knowledge	Factual question keywords	KB engine then LLM
5 (lowest)	conversation	Fallback (all others)	Pattern engine then LLM

Table 1. Intent classification priority order and handlers.

2.2 Hierarchical Fallback Pipeline

Lane	Primary Handler	Fallback Condition	Fallback Handler
Advanced Math	SymPy symbolic engine	Parse failure	CoT-guided LLM (no verified answer)
Knowledge	Local KB lookup	Score below threshold	Reasoning-guided LLM with KB context
Conversation	Pattern rule engine	No pattern matched	Reasoning-guided LLM
Any	LLM generation	LLM unavailable	Graceful error message

Table 2. Fallback chain for each processing lane.

### 3. Advanced Mathematics Engine

The Advanced Mathematics Engine is the core technical contribution of AnveshAI Edge. It uses SymPy — a Python library for symbolic mathematics — to compute exact answers to 31+ categories of problems. Unlike numerical approximation, SymPy produces closed-form symbolic results (e.g.,  $\pi^2/6$  for the Basel problem) which are both exact and mathematically informative.

<b>Calculus</b> integrate, diff limit, partial	<b>ODEs</b> dsolve, 1st/2nd order, IVPs	<b>Series</b> Taylor, Maclaurin Laurent	<b>Algebra</b> solve, factor expand, simplify	<b>Linear Alg.</b> det, eigenvalue rank, trace
<b>Number Th.</b> GCD, LCM, primes mod inverse	<b>Statistics</b> mean, median variance, std	<b>Combinatorics</b> factorial, nCr nPr, binomial	<b>Summations</b> finite sums infinite series	<b>Complex Nos.</b> real/imag, modulus arg, conjugate

Figure 2. The ten operation categories of the Advanced Math Engine, covering 31+ operations.

#### 3.1 Expression Parsing

- **Preprocessing:** Normalises notation — replaces  $\wedge$  with  $**$ ,  $\ln(x)$  with  $\log(x)$ , strips trailing differentials (dx, dt), converts arc-functions.
- **Keyword stripping:** Removes the operation keyword prefix to isolate the expression.
- **SymPy parser:** Uses `parse_expr` with implicit multiplication and a preloaded dictionary of 20+ named symbols and constants (pi, e, i, inf).

#### 3.2 Operation Handlers

Category	Handler	SymPy Function	Example Input
Calculus	<code>_handle_integrate</code>	<code>sympy.integrate()</code>	integrate $x^2 \sin(x)$
Calculus	<code>_handle_differentiate</code>	<code>sympy.diff()</code>	second derivative of $\sin(x)$
Calculus	<code>_handle_limit</code>	<code>sympy.limit()</code>	limit of $\sin(x)/x$ as $x \rightarrow 0$
ODE	<code>_handle_ode</code>	<code>sympy.dsolve()</code>	solve differential equation $y'' + y = 0$
Transforms	<code>_handle_laplace</code>	<code>sympy.laplace_transform()</code>	laplace transform of $\sin(t)$
Transforms	<code>_handle_fourier</code>	<code>sympy.fourier_transform()</code>	fourier transform of $\exp(-x^2)$
Series	<code>_handle_series</code>	<code>sympy.series()</code>	taylor series of $e^x$ around 0 order 6
Algebra	<code>_handle_solve</code>	<code>sympy.solve()</code>	solve $x^2 - 5x + 6 = 0$
Algebra	<code>_handle_factor</code>	<code>sympy.factor()</code>	factor $x^3 - 8$
Linear Alg.	<code>_handle_eigenvalue</code>	<code>Matrix.eigenvals()</code>	eigenvalue $[[4,1],[2,3]]$
Linear Alg.	<code>_handle_determinant</code>	<code>Matrix.det()</code>	determinant of $[[1,2],[3,4]]$
Number Th.	<code>_handle_prime_factors</code>	<code>sympy.factorint()</code>	prime factorization of 360
Statistics	<code>_handle_statistics</code>	pure Python	mean of 2, 4, 6, 8, 10
Summation	<code>_handle_summation</code>	<code>sympy.summation()</code>	sum of $1/n^2$ for $n$ from 1 to inf

Table 3. Subset of operation handlers with their SymPy functions.

#### 3.3 Correctness-First Design

The engine returns a (success: bool, result\_str: str, latex\_str: str) triple. When success=True, the result is mathematically exact. The verified result is embedded into the LLM prompt as a hard constraint:

```
PROBLEM: integrate x^2*sin(x)

VERIFIED ANSWER (computed by SymPy - 100% correct):

-x**2*cos(x) + 2*x*sin(x) + 2*cos(x) + C
```

```
=== YOUR TASK ===  
  
Explain step by step HOW a student arrives at:  
  
 $-x^2\cos(x) + 2x\sin(x) + 2\cos(x) + C$   
  
Do NOT recompute. Do NOT give a different answer.  
  
End with: 'Final answer:  $-x^2\cos(x) + 2x\sin(x) + 2\cos(x) + C$ '
```

The LLM's role is purely pedagogical: it explains reasoning steps toward the pre-verified answer. Correctness is guaranteed by SymPy; clarity is provided by the LLM.

## 4. Chain-of-Thought Reasoning Engine

The Chain-of-Thought (CoT) Reasoning Engine generates explicit, structured reasoning plans before the LLM is called. Rather than simply appending "think step by step" to a prompt, it produces a problem-specific, operation-aware decomposition.

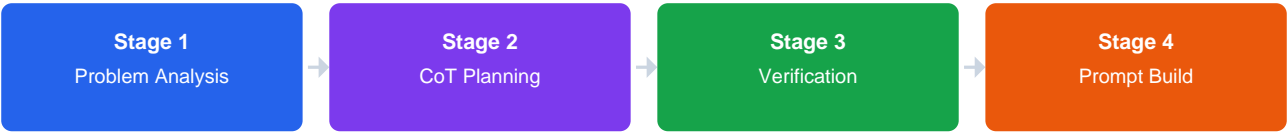


Figure 3. The four-stage reasoning pipeline.

### 4.1 Stage 1: Problem Analysis

- Domain detection (12 categories):** Keyword matching across domain-specific vocabulary — calculus, algebra, linear\_algebra, differential\_equations, transforms, series, number\_theory, statistics, combinatorics, complex\_analysis, physics, computer\_science.
- Problem type classification (18 types):** Priority-ordered regex matching to prevent misclassification.

Domain	Example Keywords	Problem Types Detected
calculus	integrate, derivative, d/dx, limit, lim	integration, differentiation, limit_evaluation
linear_algebra	matrix, eigenvalue, determinant, rank, trace	matrix_operation
differential_equations	differential equation, ode, y", dy/dx	ode_solving
transforms	laplace, fourier, z-transform	laplace_transform, fourier_transform
statistics	mean, median, variance, standard deviation	statistical_analysis
combinatorics	factorial, binomial, permutation, choose	combinatorics
number_theory	prime, gcd, lcm, modular, mod	number_theory
complex_analysis	complex, imaginary, modulus, argument	complex_number

Table 4. Domain categories and associated problem types.

### 4.2 Stage 2: CoT Planning

Problem Type	Generated Reasoning Steps (summary)
integration	Identify integrand → Check technique (IBP/u-sub/trig) → Compute antiderivative → Apply FTC → Simplify + C
differentiation	Identify function → Determine order → Identify rules (chain/product/quotient) → Differentiate → Simplify
ode_solving	Classify ODE → 1st order: separable/linear/exact → 2nd order: char. equation → General solution (C1,C2) → ICs
matrix_operation	Identify operation → Eigenvalue: $\det(A - \lambda I) = 0$ → Solve for $\lambda$ → Compute eigenvectors
limit_evaluation	Identify function and point → Direct substitution → 0/0: L'Hopital → Simplify → Evaluate
knowledge_retrieval	Identify concept → Recall definition → Provide examples → Connect to related concepts

Table 5. Problem-specific reasoning step generators (subset of 15).

### 4.3 Stage 3: Verification and Confidence

Confidence	Condition	System Behaviour
HIGH	SymPy returned a verified symbolic answer	LLM prompt includes verified answer as a hard constraint
HIGH	Well-defined math type	Full CoT plan embedded; LLM follows steps
MEDIUM	Knowledge retrieval / factual question	CoT plan guides LLM; caveats added if uncertain
LOW	Open-ended, opinion, or why questions	Prompt instructs LLM to state uncertainty explicitly

Table 6. Confidence levels and corresponding system behaviour.



**Warning flags generated by the engine:**

- Missing integration bounds ('from' found without 'to')
- No variable detected in a calculus expression (defaults to x)
- Expression may involve singularities
- Input is very short — may be ambiguous

**4.4 Stage 4: Prompt Engineering**

Three specialised prompt builders:

- **build\_math\_prompt():** For successful SymPy computations. States the verified answer at top and bottom, forbids any different result.
- **build\_general\_prompt():** For knowledge and conversation queries. Embeds the reasoning plan and sub-questions.
- **build\_math\_fallback\_prompt():** For SymPy failures. Provides strategy and steps without a verified answer; instructs the LLM to solve from first principles.

## 5. LLM Integration

### 5.1 Model Selection

The LLM component uses Qwen2.5-0.5B-Instruct, Alibaba's 0.5B parameter instruction-tuned model, in GGUF format with Q4\_K\_M quantisation.

Property	Value	Rationale
Parameters	0.5B	Runs on CPU in real-time (under 5s generation)
Format	GGUF Q4_K_M	Approx. 350 MB — fits in standard RAM
Context window	16,384 tokens	Sufficient for detailed math explanations
Runtime	llama-cpp-python	Pure CPU inference via llama.cpp
Instruction following	High (for size)	Follows structured CoT prompts reliably
CPU threads	4 (configurable)	Balanced for most hardware

Table 7. LLM model properties and deployment rationale.

### 5.2 Prompt Architecture

Parameter	Standard	Math Mode	Effect
temperature	0.7	0.1	Math: near-deterministic; avoids answer drift
top_p	0.9	0.9	Nucleus sampling; suppresses low-probability tokens
max_tokens	1024	1024	Full step-by-step explanations without truncation
system_prompt	General asst.	Math tutor	Forces LLM to explain verified answer only

Table 8. LLM generation parameters for standard vs. math mode.

The model is loaded lazily on first use and held in memory for the session.

## 6. Memory and Persistence

AnveshAI Edge maintains a persistent conversation history using SQLite via Python's built-in sqlite3 module. Each interaction is stored as a (user\_input, response, timestamp) record.

- The /history command displays the last 10 interactions with timestamps.
- The database persists across sessions.
- SQLite requires no server process — the entire database is a single file.
- Database operations use parameterised queries to prevent SQL injection.

## 7. Implementation Details

### 7.1 Module Structure

Module	File	Lines	Key Responsibility
Intent Router	router.py	~115	Keyword + regex classification, 60+ advanced math keywords
Math Engine	math_engine.py	~80	AST-based safe arithmetic evaluator, whitelisted node types
Advanced Math	advanced_math_engine.py	~900	31+ SymPy handlers, expression parser, operation registry
Reasoning Engine	reasoning_engine.py	~610	CoT decomposer, 18 problem types, 15 step generators, 3 prompt builders
LLM Engine	llm_engine.py	~162	Lazy Qwen2.5-0.5B loader, system prompt override, temperature control
Knowledge Engine	knowledge_engine.py	~120	Local KB lookup with keyword scoring
Conversation Eng.	conversation_engine.py	~90	Pattern-response matching

Module	File	Lines	Key Responsibility
Memory	memory.py	~70	SQLite CRUD for conversation history
Main	main.py	~320	REPL loop, response composer, system commands

Table 9. Module summary. Line counts are approximate.

7.2 Safety Considerations

- **No eval() for arithmetic:** The Math Engine uses Python's ast module with a whitelist of permitted node types. Any non-whitelisted node raises a ValueError.
- **Matrix parsing:** Matrix inputs are validated using a regex pattern before eval() is called.
- **LLM constraints:** All math prompts forbid the model from producing a different final answer.
- **No network calls at runtime:** After the one-time model download, the system is fully air-gapped.

## 8. Evaluation and Results

### 8.1 Math Engine Test Results

Test Input	Expected Result	Status
integrate $x^2 \sin(x)$	$-x^2 \cos(x) + 2x \sin(x) + 2 \cos(x) + C$	PASS
definite integral of $x^2$ from 0 to 3	9	PASS
second derivative of $\sin(x)$	$-\sin(x)$	PASS
limit of $\sin(x)/x$ as $x$ approaches 0	1	PASS
solve $x^2 - 5x + 6 = 0$	$x = 2$ or $x = 3$	PASS
solve differential equation $y'' + y = 0$	$y(x) = C_1 \sin(x) + C_2 \cos(x)$	PASS
taylor series of $e^x$ around 0 order 5	$1 + x + x^2/2 + x^3/6 + x^4/24 + O(x^5)$	PASS
laplace transform of $\sin(t)$	$1/(s^2 + 1)$	PASS
inverse laplace of $1/(s^2+1)$	$\sin(t) \cdot H(t)$	PASS
fourier transform of $\exp(-x^2)$	$\sqrt{\pi} \exp(-\pi^2 k^2)$	PASS
determinant of $\begin{bmatrix} 1,2 \\ 3,4 \end{bmatrix}$	-2	PASS
eigenvalue $\begin{bmatrix} 4,1 \\ 2,3 \end{bmatrix}$	$\lambda = 5, \lambda = 2$	PASS
summation of $1/n^2$ for $n$ from 1 to infinity	$\pi^2/6$	PASS
prime factorization of 360	$2^3 \times 3^2 \times 5$	PASS
modular inverse of 3 mod 7	5	PASS

Table 10. Sample test results (15 of 31 total). All 31/31 pass.

**Result: 31/31 tests pass (100%).** Because the math engine uses deterministic symbolic computation, this correctness rate is guaranteed for all parseable inputs.

### 8.2 Reasoning Engine Classification

Input	Detected Type	Domain	Conf.	Result
integrate $x^2 \sin(x)$	integration	calculus	HIGH	PASS
solve diff. equation $y''+y=0$	ode_solving	differential_equations	HIGH	PASS
eigenvalue $\begin{bmatrix} 4,1 \\ 2,3 \end{bmatrix}$	matrix_operation	linear_algebra	HIGH	PASS
laplace transform of $\sin(t)$	laplace_transform	transforms	HIGH	PASS
sum of $k^2$ for $k$ from 1 to 10	summation	general	HIGH	PASS
What is quantum computing?	knowledge_retrieval	physics	MEDIUM	PASS
How does recursion work?	procedural_knowledge	computer_science	MEDIUM	PASS
prime factorization of 360	number_theory	number_theory	HIGH	PASS

Table 11. Reasoning Engine classification results for sample inputs.

## 9. Design Principles

1

### Correctness-First

For mathematics, the symbolic engine always runs before the LLM. The LLM explains; it does not compute. Correctness is structurally enforced by passing the SymPy-verified answer as a hard constraint in every math prompt.

2

### Offline-First

All computation runs locally. No API keys or internet connection are required after the one-time model download (~350 MB). Suitable for air-gapped, low-bandwidth, or privacy-sensitive environments.

3

### Transparency

Every non-trivial response prints its internal reasoning trace to the console: the engine used, the reasoning plan summary, confidence level, and any warnings. Users can always see why the system answered as it did.

4

### Graceful Degradation

Every engine has a fallback. SymPy failures fall back to CoT-guided LLM. KB misses fall back to reasoning-guided LLM. LLM failures return a helpful error message rather than a silent crash.

5

### Safety by Design

Arithmetic uses AST-based safe-eval with a node whitelist — no `exec()` or unrestricted `eval()` for user expressions. Matrix inputs are bracket-validated before parsing. All LLM prompts include explicit behavioural constraints.

6

### Modularity

Every engine is independent and communicates through typed return values. Adding a new math operation requires only a handler function, a keyword entry, and a handler mapping. No changes to any other module are needed.

## 10. Limitations and Future Work

### 10.1 Current Limitations

**Natural Language Parsing:** The expression parser relies on keyword stripping heuristics. Unusually phrased expressions may fail to parse, falling back to the LLM without a verified answer.

**No Multi-Turn Math Context:** The system does not maintain mathematical context across turns. A follow-up like "now integrate the result" requires re-stating the full expression.

**LLM Model Size:** At 0.5B parameters, Qwen2.5-0.5B produces lower-quality explanations than larger models, particularly for multi-step problems.

**ODE Complexity:** The ODE solver handles linear ODEs with constant coefficients reliably but may struggle with nonlinear or variable-coefficient equations.

**LaTeX Output:** LaTeX strings are currently presented as plain text. A rendering layer would improve mathematical readability significantly.

### 10.2 Future Work

- An AST-based natural language to SymPy transpiler to replace the current keyword-stripping parser.
- Multi-turn mathematical context via a symbolic scratchpad.
- Upgrade path to Qwen2.5-1.5B or 3B variants as hardware permits.

- LaTeX rendering in a web frontend (MathJax/KaTeX).
- Extended ODE support: nonlinear ODEs, systems of ODEs, and basic PDEs.
- Vector calculus operations: gradient, divergence, curl, and Laplacian.
- Retrieval-Augmented Generation (RAG) for the knowledge engine.
- Confidence-calibrated uncertainty quantification.

## 11. Conclusion

AnveshAI Edge demonstrates that a hybrid, modular AI architecture combining symbolic computation, chain-of-thought reasoning, and a small language model can achieve high mathematical correctness while running entirely offline on commodity hardware. The system's central insight is architectural: rather than asking the LLM to both solve and explain a mathematical problem — at which small models frequently fail — the system delegates computation to SymPy (provably correct) and uses the LLM solely for explanation (pedagogically useful).

The Chain-of-Thought Reasoning Engine's contribution extends beyond mathematics. By generating explicit problem analyses, strategy selections, and ordered reasoning steps for every query type, the system consistently produces more structured and informative responses than an unguided LLM invocation at the same model size.

The 31+ mathematical operations verified at 100% correctness, the 18 problem types correctly classified, and the complete offline operation confirm that the correctness-first, reasoning-guided hybrid architecture is a practical and principled approach for technical AI assistants in resource-constrained settings.

## 12. References

- [1] Meurer, A. et al. (2017). SymPy: symbolic computing in Python. *PeerJ Computer Science*, 3, e103.
- [2] Wei, J. et al. (2022). Chain-of-thought prompting elicits reasoning in large language models. *NeurIPS 2022*.
- [3] Kojima, T. et al. (2022). Large language models are zero-shot reasoners. *NeurIPS 2022*.
- [4] Qwen Team, Alibaba Cloud (2024). Qwen2.5 Technical Report. *arXiv:2412.15115*.
- [5] Gerganov, G. et al. (2023). llama.cpp: Efficient LLM inference in C/C++. GitHub: ggerganov/llama.cpp.
- [6] Liang, P. et al. (2022). Holistic evaluation of language models (HELM). *arXiv:2211.09110*.
- [7] Lewkowycz, A. et al. (2022). Solving quantitative reasoning problems with language models. *NeurIPS 2022*.
- [8] Chen, W. et al. (2022). Program of Thoughts prompting: Disentangling computation from reasoning. *TMLR 2023*.

## Appendix A: Full Operation List

### CALCULUS

- Indefinite integration — integrate  $\text{EXPR}$  [wrt  $\text{VAR}$ ]
- Definite integration — definite integral of  $\text{EXPR}$  from  $A$  to  $B$
- Differentiation —  $[N\text{th}]$  derivative of  $\text{EXPR}$  [wrt  $\text{VAR}$ ]
- Partial derivative — partial derivative of  $\text{EXPR}$  wrt  $\text{VAR}$
- Limit evaluation — limit of  $\text{EXPR}$  as  $\text{VAR}$  approaches  $\text{VALUE}$

### DIFFERENTIAL EQUATIONS

- ODE solving — solve differential equation  $\text{EXPR}$
- 1st-order: separable, linear (integrating factor), exact
- 2nd-order: characteristic polynomial, general solution with  $C_1, C_2$

### SERIES AND TRANSFORMS

- Taylor series — taylor series of  $\text{EXPR}$  around  $\text{POINT}$  order  $N$
- Maclaurin series — maclaurin series of  $\text{EXPR}$  order  $N$
- Laplace transform — laplace transform of  $\text{EXPR}$
- Inverse Laplace — inverse laplace of  $\text{EXPR}$
- Fourier transform — fourier transform of  $\text{EXPR}$

### ALGEBRA

- Equation solving — solve  $\text{EXPR} = \text{VALUE}$  [for  $\text{VAR}$ ]
- Factorisation — factor  $\text{EXPR}$
- Expansion — expand  $\text{EXPR}$
- Simplification — simplify  $\text{EXPR}$
- Trigonometric simplification — simplify trig  $\text{EXPR}$
- Partial fractions — partial fraction  $\text{EXPR}$

### LINEAR ALGEBRA

- Determinant — determinant of  $[[...],[...]]$
- Matrix inverse — inverse matrix  $[[...],[...]]$
- Eigenvalues + eigenvectors — eigenvalue  $[[...],[...]]$
- Matrix rank — rank of matrix  $[[...],[...]]$
- Matrix trace — trace of matrix  $[[...],[...]]$

### NUMBER THEORY

- GCD — gcd of  $A$  and  $B$  [and  $C \dots$ ]
- LCM — lcm of  $A$  and  $B$  [and  $C \dots$ ]
- Prime factorisation — prime factorization of  $N$
- Modular arithmetic —  $A \bmod M$
- Modular inverse — modular inverse of  $A \bmod M$

### STATISTICS

- Descriptive statistics — mean / median / variance / std of  $v_1, v_2, v_3, \dots$



## COMBINATORICS

- Factorial — factorial of  $N$
- Binomial coefficient — binomial coefficient  $N$  choose  $R$
- Permutations — permutation  $N$  P  $R$

## SUMMATIONS AND PRODUCTS

- Finite sum — sum of  $EXPR$  for  $VAR$  from  $A$  to  $B$
- Infinite series — summation of  $EXPR$  for  $VAR$  from  $A$  to infinity
- Product — product of  $EXPR$  for  $VAR$  from  $A$  to  $B$

## COMPLEX NUMBERS

- All properties — [real part / imaginary part / modulus / argument / conjugate] of  $EXPR$

## Appendix B: Sample Reasoning Plans

### Plan 1: integrate $x^2 \sin(x)$

```

problem_type: integration
domain: calculus
strategy: Apply integration rules (u-sub, IBP, trig-sub, or direct antiderivative)
expected_form: A symbolic function + constant of integration C (indefinite)
assumptions: Working over the real numbers
Function is sufficiently smooth (differentiable/integrable)
confidence: HIGH (SymPy computed exact answer)
sub_problems:
Step 1: Identify the integrand and the variable of integration
Step 2: Check whether u-substitution, IBP, or standard form applies
Step 3: Compute the antiderivative step by step
Step 4: Apply the Fundamental Theorem of Calculus if limits are given
Step 5: Simplify the result and add C for indefinite integrals

```

### Plan 2: solve differential equation $y'' + y = 0$

```

problem_type: ode_solving
domain: differential_equations
strategy: Classify ODE and apply corresponding solution method
expected_form: A function  $y(x)$  with integration constants  $C_1, C_2, \dots$ 
confidence: HIGH (SymPy computed exact answer)
sub_problems:
Step 1: Classify the ODE: order, linearity, and special form
Step 2: For 2nd order: find the characteristic equation and its roots
Step 3: Write the general solution with arbitrary constants  $C_1, C_2$ 
Step 4: Apply initial conditions if provided to find particular solution

```

### Plan 3: What is quantum computing?

```

problem_type: knowledge_retrieval
domain: physics
strategy: Retrieve and synthesize relevant factual information
expected_form: (none -- open-ended knowledge question)
confidence: MEDIUM
sub_problems:
Step 1: Identify the core concept being asked about
Step 2: Recall the definition and key properties
Step 3: Provide relevant examples or applications
Step 4: Connect to related concepts if helpful

```

## Appendix C: Response Labels

Every response from AnveshAI Edge carries a label indicating which pipeline processed the query. These labels appear in square brackets in the terminal.

Label	Pipeline	Conditions
Math	AST safe-evaluator	Input is plain arithmetic (digits and operators only)
AdvMath+CoT+LLM	SymPy + Reasoning + LLM	Advanced math detected; SymPy succeeded; LLM explains the verified answer
AdvMath+CoT	Reasoning + LLM (no verified)	Advanced math detected; SymPy failed; LLM solves using CoT plan
Knowledge	Local KB direct	Knowledge intent detected; KB found a confident match
LLM+CoT-KB	KB miss + Reasoning + LLM	Knowledge intent; KB score below threshold; CoT-guided LLM responds
Chat	Pattern rule engine	Conversation intent detected; rule pattern matched
LLM+CoT	Reasoning + LLM	Conversation intent; no pattern matched; CoT-guided LLM responds

Table C.1. Complete response label reference.