

APPENDIX

A USE OF LLMs

The ideation and underlying idea for using the Curry-Howard correspondence for CoT and interpretability was that of the authors. LLMs were used to implement different coding strategies, though we found them to be a bit haphazard and often a false economy. LLMs were also used to refine and edit, but again we found them useful for scaffolding but for actual prose and design choices for the paper, they often were cumbersome. The OpenAI GPT5 API was used during the experiments. We decided against open-source models given space limitations and the need for fast reliable API calls for reasoning on a budget. We are exploring open source models these in ongoing work.

B EXAMPLE: CERTIFICATION METRICS

To illustrate how the certification metrics operate in practice, we provide a full example drawn from the GSM8K benchmark.

Problem. “A raspberry bush has 6 clusters of 20 fruit each and 67 individual fruit scattered across the bush. How many raspberries are there total?”

Step 1: Program emission. The emitter produces a structured JSON program:

```
{
  "program": {
    "premises": [
      {"id": "v1", "value": 6, "unit": "count"},
      {"id": "v2", "value": 20, "unit": "count"},
      {"id": "v3", "value": 67, "unit": "count"}
    ],
    "ops": [
      {"id": "t1", "op": "mul", "inputs": ["v1", "v2"], "out": "t1"},
      {"id": "t2", "op": "add", "inputs": ["t1", "v3"], "out": "t2"}
    ],
    "answer": {"value": 187, "unit": "count", "therefore_id": "therefore::1"}
  }
}
```

Step 2: Typed rendering. We render the JSON program into a deterministic typed derivation:

```
Premise v1 : 6 [count]
Premise v2 : 20 [count]
Premise v3 : 67 [count]
t1 : 6 × 20 = 120 [count]
t2 : 120 + 67 = 187 [count]
Therefore : 187 [count]
```

This textualisation is what we call a *typed proof sketch*: it can be directly audited by a human and checked mechanically by the evaluator.

Step 3: Certification metrics. For this run, the metrics are:

- **Coverage.** Two operations were generated, and both are successfully typed and integrated into the Typed Reasoning Graph (TRG). Hence

$$\text{Coverage} = \frac{2}{2} = 1.0.$$

- **EVR.** Each operation satisfies its rule preconditions: `mul` takes two numeric inputs, and `add` takes two numeric inputs. Thus

$$\text{EVR} = \frac{2}{2} = 1.0.$$

- **UVR.** Unit propagation is dimensionally valid throughout:

$$\begin{aligned} \text{count} \times \text{count} &\rightarrow \text{count}, \\ \text{count} + \text{count} &\rightarrow \text{count}. \end{aligned}$$

Both operations respect unit typing, so

$$\text{UVR} = \frac{2}{2} = 1.0.$$

- **PE (Path Exists).** The TRG contains a directed path from the premises $\{v1, v2, v3\}$ through $t1$ and $t2$ to the “Therefore” node. Hence

$$\text{PE} = 1.$$

- **MPS (Minimal Path Size).** The shortest typed path to the conclusion involves two inference steps ($t1, t2$), so

$$\text{MPS} = 2.$$

Step 4: Gate application. Under the *relaxed* gate, which requires $\text{EVR} \geq 0.3$ and $\text{PE} = 1$, this chain is accepted. Under the *strict* gate, which requires $\text{EVR} \geq 0.8$, $\text{UVR} \geq 0.8$, $\text{PE} = 1$, and numeric consistency, the chain is also accepted. The answer 187 matches the gold label, so the run is both correct and certified.

Step 5: Interpretation. This example demonstrates how the metrics operationalise the Curry–Howard view of “proofs as programs.” The chain-of-thought is not just a sequence of plausible natural-language steps; it is a typed program whose typed dataflow is well-formed under the rules of the type system. UVR plays a critical role here: it ensures that the reasoning respects dimensional validity, ruling out degenerate chains that arrive at the correct number through ill-typed operations (e.g. adding dollars and counts). Certification is thus stricter than accuracy alone: it requires that the derivation be both correct and well-typed.

EXAMPLE: UNIT-TYPE FAILURE

To demonstrate the role of unit types (UVR), consider the following GSM8K-style problem:

Problem. “The expenditure of Joseph in May was \$500. In June, his expenditure was \$60 less. How much was his total expenditure for those two months?”

Step 1: Program emission. The emitter produces this JSON program:

```
{
  "program": {
    "premises": [
      {"id": "v1", "value": 500, "unit": "usd"},
      {"id": "v2", "value": 60, "unit": "count"}
    ],
    "ops": [
      {"id": "t1", "op": "sub", "inputs": ["v1", "v2"], "out": "t1"},
      {"id": "t2", "op": "add", "inputs": ["v1", "t1"], "out": "t2"}
    ],
    "answer": {"value": 940, "unit": "usd", "therefore_id": "therefore::1"}
  }
}
```

Step 2: Typed rendering.

Premise $v1$: 500 [usd]
 Premise $v2$: 60 [count]
 $t1$: $500 - 60 = 440$ [invalid]
 $t2$: $500 + 440 = 940$ [usd?]
 Therefore : 940 [usd]

Here the subtraction step $500 - 60$ mismatches the units: dollars minus counts is not type-valid. Although the numeric result (440) happens to be correct, the typing is invalid.

Step 3: Certification metrics.

- **Coverage:** Both operations are included in the TRG, so Coverage = 1.0.
- **EVR:** The operations meet structural preconditions (correct number of inputs), so EVR = 1.0.
- **UVR:** The first operation fails unit propagation (usd – count). Only 1 of 2 ops is valid, so
$$\text{UVR} = \frac{1}{2} = 0.5.$$
- **PE:** A path exists from premises to the conclusion, so PE = 1.
- **MPS:** The minimal path uses both $t1$ and $t2$, so MPS = 2.

Step 4: Gate application.

- Under the *relaxed* gate ($\text{EVR} \geq 0.3$, $\text{PE} = 1$), the chain is *accepted* because UVR is not enforced.
- Under the *strict* gate ($\text{EVR} \geq 0.8$, $\text{UVR} \geq 0.8$, $\text{PE} = 1$, consistency required), the chain is *rejected*, because $\text{UVR} = 0.5$.

Step 5: Interpretation. This example highlights why unit typing is crucial. The final answer 940 is numerically correct and would pass a naïve accuracy check. Yet the reasoning chain contains a semantically invalid operation (subtracting counts from dollars). Without UVR, such ill-typed chains inflate accuracy metrics. By contrast, the strict gate with UVR ensures that only dimensionally valid programs are accepted, making the certified chains more faithful to the Curry–Howard ideal that well-typed programs are proofs.

C APPENDIX: CODEBASE SEQUENCING AND IMPLEMENTATION NOTES**C.1 CELL-BY-CELL SEQUENCING**

Table 5 summarises the main sequence of cells in the notebook, their dependencies, and the artefacts they produce. This provides a map for reproducibility and for new contributors wishing to extend the framework. We ran all experiments in a Colab notebook using an A100-GPU runtime. Costs for the overall experiment were in the order of around USD100.00. The codebase with Jupyter notebook for reproducibility is available via Anonymous (2025). The table below is LLM-generated as was a lot (but not all) of the code following incremental instructions from our design process.

C.2 CHALLENGES AND VARIATIONS DURING DEVELOPMENT

Several challenges and variations arose during the development of PC-CoT:

API quirks. The GPT-5 API required careful handling of parameters. For example, only `max_completion_tokens` (not `max_tokens`) is valid, and some variants reject `temperature=0`. Truncation of outputs was common, requiring the implementation of token escalation (+1000 tokens on retries).

Cell	Name	Purpose and Functionality	Key Outputs / Artefacts
1	Runtime / Setup	Install packages, mount Drive, create directory tree, verify GPU	Directory structure under Drive
2	Config / Utilities	Load API keys, seeds, JSON/CSV helpers	Config object, I/O functions
3	Type System	Define numeric, tuple, and unit types with coercions	Type-checker functions, unit rules
4	Rule Schemas	Define inference primitives (Add, Mul, Div, Assume, Therefore)	Rule definitions, unit tests
5	Proof Memory (Γ)	Store typed statements with confidences, prune stale items	In-memory store, pruning policy
6	TRG Core	Construct Typed Reasoning Graph, implement metrics (Coverage, EVR, PE, MPS)	TRG objects, metric functions
7	Segmentation / Labeler	Segment text into candidate steps, bootstrap rule labels	LabeledStep objects
8	TRG Builder	Build TRG from segmented CoT, attach equations and Therefore node	TRG JSON, initial graphs
9	Synthetic Programs	(Optional) Generate gold proofs, compute graph distances	CSVs, faithfulness plots
10	Statistics Utilities	Bootstrap CI, ROC/AUC, calibration metrics	Diagnostic statistics
11	HF Loader	Load Hugging Face models for ablations	Model configs
12	GSM8K Loader (baseline)	Sample dataset, generate vanilla CoT	Raw CoTs, JSON logs
13	TRG on GSM8K	Apply TRG to vanilla CoTs	Coverage, EVR, PE scores
14	GPT-5 Labeler	Label steps with GPT-5, stabilise rule classification	Cached labels (gpt5_label_cache/)
15	PC-CoT (L3)	Main decoder: schema prompts, typed checks, soft constraints	Typed Faithfulness Certificates (TFCs)
16	Baselines	Run CoT, SC, PAL with budget matching	Comparison outputs
17	Certified SC	Apply TRG/TFC gates, aggregate certified runs	CSC outputs, logs
18	Robustness	Paraphrases, distractors, unit traps	CSVs, robustness figures
19	Significance Tests	Aggregate runs, compute bootstrap CIs, calibration	Summary stats, diagnostic plots
20	Ablations	Sweep thresholds, compare L3 vs L4, coarse/fine rules	CSVs, ablation plots
21	Pilot (n=5)	End-to-end pipeline on 5 GSM8K items	JSONL, diagnostics, figures
22a	Baseline Answer-only	Generate answer-only runs for alignment with 22b	Raw text, CSVs
22b	JSON Program Runs	Generate JSON programs, typed renderings, save side-by-side	runX_program.pretty.json, typed_program.txt, json-vs-typed.md
22	Merge / Visualise	Align 22a+22b, plot accuracy, CoTProgram correspondence	CSVs, bar plots, side-by-side panels
23	Pilot (n=50)	Larger run for Series-I reporting	Aggregated results, figures

Table 5: Sequencing of cells in the PC-CoT notebook.

Classifier instability. Early versions of the labeler misclassified steps or failed to stabilise rule heads. We introduced heuristics that forced “Therefore” steps to be labelled correctly and equations to be mapped to the corresponding compute rules. This increased EVR and PE significantly.

Unit validity. Introducing UVR (Unit Validity Ratio) was a late addition motivated by failures where numerically correct answers were produced via dimensionally invalid operations (e.g. subtracting counts from dollars). Implementing unit propagation rules improved strict precision.

Soft vs. hard constraints. We experimented with rigid grammar-constrained decoding (L4) versus soft constraints (L3). Hard constraints reduced coverage drastically, while soft constraints (logit masking, hint injection) preserved diversity and yielded higher overall accuracy.

Run size and cost. While full runs over GSM8K are possible, we found that subsets of 200–300 examples were cost-effective for prototyping. Larger pilots (n=50, n=199) were staged once the pipeline was stable.

Visualisation complexity. Cell 22 Viz was iteratively expanded to align baseline and PC-CoT runs, compute agreement rates, and render side-by-side comparisons. Matching CoT lines to program operations required robust heuristics for number matching and operator detection.

Abstention behaviour. Designing Certified Self-Consistency required deciding what to do when no certified runs exist. Our choice was abstention, rather than fallback to an uncertified answer, to prioritise precision in safety-critical settings.

Variations explored. We explored multiple ablations:

- Removing type checks entirely (accuracy fell to $\sim 41\%$)
- Dropping path requirements (accuracy fell to $\sim 52\%$)
- Aggregating all runs without certification (accuracy fell to $\sim 36\%$)
- Enforcing L4 hard constraints (accuracy $\sim 49\%$)

Each ablation underscored the value of typing, path enforcement, and soft constraints in achieving strong performance.

Lessons. The development process highlighted the fragility of untyped CoT reasoning and the importance of type-driven structure. Program-text dual representations (JSON + typed rendering) were crucial for both machine verification and human interpretability, and incremental saving (per-run JSON, per-question directories, checkpoints) ensured robustness against API failures and runtime interruptions.

D TYPED REASONING GRADIENT

In the main paper we introduced the idea of a *typed reasoning gradient*: the observation that chain-of-thought (CoT) traces do not fall neatly into categories of “faithful” or “unfaithful,” but instead occupy a graded spectrum of typed structure. Here we expand this concept and describe the levels in more detail.

- **Level 0: Unstructured narrative.** At this level the model produces free-form natural language with no extractable operations. The text may contain intuitive reasoning or explanatory language, but from the perspective of the type system there is no structured content to check. These traces provide little beyond surface plausibility and cannot be verified; they correspond to the “post-hoc rationalization” failure mode highlighted in prior work.
- **Level 1: Identifiable operations without valid paths.** Here, the model emits some steps that can be mapped to primitive operations (e.g., additions or multiplications), but the resulting Typed Reasoning Graph (TRG) fails to form a valid path from premises to conclusion. This can occur because intermediate values are unused, because the “Therefore” node is disconnected, or because typing constraints are violated. Such traces demonstrate partial structure but remain logically incomplete.
- **Level 2: Partial typed paths.** In this category, the TRG contains one or more valid typed paths that cover a portion of the reasoning, but not all relevant premises or intermediate steps are included. For example, a multiplication might be well-typed and propagate correctly, but subsequent additions or unit checks fail, leaving the path incomplete. These runs are often rejected by strict certification, but they provide evidence that the model is gesturing toward proof-like structure even if it cannot fully sustain it.
- **Level 3: Complete typed proofs.** At the highest level, the reasoning trace yields a complete typed program from premises to conclusion, with all operations type-checked, units propagated, and a valid path to the “Therefore” node. These runs correspond to strict-certified outputs under our gate criteria and demonstrate the full operationalisation of the Curry–Howard correspondence: the chain-of-thought is literally a program that computes the answer.

Discussion. Our experiments on GSM8K show that only around 40–60% of correct answers reach Level 3 structure (strict-certified runs), yet these achieve precision above 90%. By contrast, Levels 1–2 are far more common among rejected runs, with many containing fragments of typed structure (e.g., an addition with correct numeric inputs) but failing unit or path checks. This distribution provides empirical support for the gradient view: correctness is not random, but strongly correlated with the amount of typed structure present. The gradient is thus both descriptive and predictive. It highlights that interpretability is not binary but exists along a spectrum, where increasing amounts of typed structure provide progressively stronger evidence of computational faithfulness. This framing also suggests directions for future work: strengthening models by converting partial structure into complete or partially-typed proofs.

E PRACTICAL DEPLOYMENT CONSIDERATIONS

PC-CoT offers different tradeoffs under different certification gates. Under the *relaxed gate* ($\text{EVR} \geq 0.30$), the system achieves high coverage, with 70% of questions yielding at least one certified run, and maintains moderate precision at 87%. This setting is therefore suitable for assisted reasoning scenarios, where the goal is to maximize usable outputs and some level of error tolerance is acceptable. In contrast, the *strict gate* ($\text{EVR} \geq 0.80$, $\text{UVR} \geq 0.80$, consistency required) produces lower coverage, with only 54% of questions admitting certified runs, but achieves very high precision at 92%. This setting is well suited for high-stakes applications where reliability and verifiability are paramount, even at the cost of abstaining more often.

The abstention mechanism is an important property in its own right. When no certified path exists, the system refrains from producing an answer rather than offering an unverified guess. In domains such as finance, healthcare, or scientific reasoning, this selective silence is preferable to overconfident but potentially invalid explanations. More broadly, the gate design provides a tunable control over the coverage–precision frontier: practitioners can adjust the thresholds to suit the tolerance for error in their target application.