Final Report

From Readings About LLM Reasoning and some research experiments

A blog style semester reflection based on chain of thought, planning, arithmetic, monitorability, and generalization

By Radiath Kamal Patwary

Github Repository: https://github.com/RadiathKamalPatwary/comp560-RadiathKamalPatwary/tree/main

Summary

I am most proud of this semester is the research foundation I built by reading deeply about large language model reasoning. I spent huge amount of time this semester doing some research experiments and reading papers; learning how different papers explain the promise and weakness of chain of thought, long chain of thought, tree-based search, symbolic planning, arithmetic generalization, and chain of thought monitorability. This report turns that reading into a research direction: small controlled LLM experiments can test whether visible scratch space actually improves generalization, or whether it only gives the appearance of reasoning when the examples remain close to training data.

Reading as a Research Achievement

At first, it felt strange to call reading an achievement. A semester achievement usually sounds like a completed project, a polished poster, a working model, a public repository, or a result that can be summarized in one sentence.I spent a large part of the semester reading papers about LLM reasoning and trying to understand not only what each paper claimed, but why the claims mattered. Over time, that reading became a framework for thinking about research. I started with the broad question of how large language models reason, and I ended with a much sharper question about scratch space, distribution shift, and small model generalization.

The most important change was that I stopped treating chain of thought as a single method. Before reading these papers, it was easy to think of chain of thought as a prompt style: ask the model to think step by step, get a longer answer, and hope that the answer is better. The papers made that view feel too simple. Chain of thought can be a prompting technique, a training target, a search

structure, a planning trace, a safety signal, or a misleading explanation. Sometimes it helps because it creates intermediate computations. Sometimes it helps because it gives the model examples that match the task. Sometimes it hurts because it makes the model over elaborate or follow surface patterns. Sometimes it is useful for monitoring, and sometimes it creates a false sense of transparency.

If someone wants to study small LLMs, arithmetic tasks, scratchpads, or chain of thought, they need to ask what the model is actually learning. Is it learning an algorithm, a heuristic, a format, a shortcut, or a distributional pattern? Auto CoT helps thinking about reasoning demonstrations as data. Tree based reasoning helps thinking about search and backtracking. Long CoT helps thinking about depth, exploration, and reflection. Symbolic planning helps thinking think about verification. Arithmetic failure helps thinking think about structural mismatch. The distribution lens helped me think about training and test shifts. CoT controllability helps think about reasoning traces as safety signals.

The Central Question: Is the Trace Doing Work?

The deepest question I took from the readings is whether a reasoning trace is doing computational work or simply describing work that the model has already implicitly done. This distinction matters because the surface form of chain of thought can be misleading. When a model writes several sentences before answering, the human reader naturally interprets those sentences as the cause of the answer. In human writing, a solution path often represents the actual reasoning path. With LLMs, the relationship is less clear. The trace might help the model organize information. It might also be a style learned from examples. It might be partially causal in some cases and mostly decorative in others.

The papers show that this cannot be answered with a single yes or no. In some settings, reasoning traces clearly improve performance. Automatic demonstrations can match or exceed manually designed chain of thought examples when selected with diversity, showing that generated intermediate steps can be useful in prompting (Zhang et al., 2022). Tree based exploration can strongly outperform a single linear chain on tasks requiring planning or search, such as Game of 24, where the reported GPT-4 success rate rises from 4 percent under ordinary chain of thought to 74 percent under the tree based method (Yao et al., 2023). Logical instruction tuning with verified planning feedback can improve plan validity, reaching up to 94 percent in the Blocksworld setting for the strongest reported variant (Verma et al., 2025). These findings make it hard to dismiss chain of thought as useless.

At the same time, other findings make it hard to treat chain of thought as genuine reasoning by default. In arithmetic, models can appear competent while relying on a shallow one digit lookahead heuristic that breaks when cascading carries or multiple operands require deeper propagation (Baeumel et al., 2025). Under distribution shifts in task, length, or format, chain of thought can become brittle, producing fluent steps that do not transfer to genuinely novel transformations (Zhao et al., 2026). Modern reasoning models can also struggle to control the contents of their own chains of thought, even when they can control their final outputs much more successfully (Chen Yueh-Han et al., 2026). These papers show that the visible trace is not automatically a faithful window into robust internal computation.

The result is a more careful research question: when does the trace actually help the model compute something it could not otherwise compute? This is the question that now feels most important to me. A scratchpad should not be evaluated only by whether it makes outputs longer or more readable. It should be evaluated by whether it changes the model's failure pattern. If a direct answer

model fails on cascading carries but a structured scratchpad model succeeds, then scratch space may be functioning as an external memory for intermediate values. If both models fail in the same places, then the scratchpad may only be imitating reasoning language. If the scratchpad works in familiar formats but collapses under format shifts, then it may be distribution dependent rather than algorithmic.

Reasoning Demonstrations as Data

One of the first big insights from the readings is that demonstrations matter because they are data. When a model is prompted with examples that include intermediate reasoning, it does not receive pure logic in an abstract form. It receives patterns of questions, formats, intermediate steps, and final answers. The model then conditions on those patterns. This means the quality, diversity, and representativeness of demonstrations can shape performance in ways that look like reasoning but are also deeply connected to data distribution.

The automatic demonstration approach is especially important because it challenges the idea that humans must hand write every good reasoning example. The method uses the model itself to generate rationales and then uses those rationales as demonstrations. The key detail is that simple similarity based retrieval is not enough. If a model generates mistaken reasoning chains, choosing highly similar examples can repeat the same type of error. The proposed solution is to cluster questions and choose representative examples from different clusters, so the prompt covers a more diverse set of reasoning patterns (Zhang et al., 2022). I found this very important because it makes prompt design look less like word choice and more like dataset construction.

This directly connects to small model research. If I train a tiny transformer on arithmetic with scratchpads, I should not assume that adding scratchpads automatically teaches the algorithm. The

scratchpads themselves become training data. If they are all written in one narrow style, the model may learn that style instead of the underlying procedure. If the examples cover only easy cases, the model may produce plausible traces on harder cases but fail exactly where the missing examples matter. If the scratchpads are diverse and explicitly mark the variables that matter, such as carry or borrow, the model may have a better chance of learning a reusable computation.

The deeper lesson is that the boundary between prompting and training is not as sharp as it first seems. In few shot prompting, examples are placed in context. In supervised training, examples update weights. In both cases, the model is exposed to a distribution of reasoning traces. That distribution can encourage or hide specific strategies. A prompt with several similar reasoning examples can become a small temporary dataset. A scratchpad training corpus can become a large permanent prompt. This is why the distribution lens later feels so powerful. It says that chain of thought is not only about asking for steps. It is about what kinds of steps the model has learned to imitate and where those steps still match the test setting.

This insight also changes how I would evaluate my own experiments. I would not simply compare direct answer versus scratchpad accuracy. I would compare different scratchpad data designs. One dataset might contain only final answers. One might contain natural language explanations. One might contain rigid digit level operations. One might contain explicit state variables. One might mix formats to test robustness. The question would be whether a model trained on these different reasoning distributions learns different solution strategies.

Linear Reasoning Is Often Too Narrow

A second major insight is that ordinary chain of thought is linear, but many problems are not. A single chain forces the model to pick one next step, then another, then another. If an early step is

wrong, the rest of the chain may be built on a bad path. Humans often solve hard problems differently. We try an approach, notice it is not working, return to an earlier point, compare alternatives, or search through possibilities. The tree based framework makes this difference explicit by treating thoughts as nodes in a search space rather than one continuous stream of tokens. The important conceptual move is that a thought is not just a token. It is a meaningful intermediate unit. In different tasks, a thought might be a line of arithmetic, a partial crossword answer, or a paragraph plan for writing. The method asks four design questions: how to decompose the process into thought steps, how to generate possible next thoughts, how to evaluate states, and what search algorithm to use (Yao et al., 2023). I found this useful because it turns vague advice like 'think harder' into a concrete structure. The model is not merely producing more text. It is generating candidate states, judging them, pruning them, and sometimes backtracking.

This helps explain why the method performs well on tasks that require search. In Game of 24, one wrong operation early can make the goal impossible, so exploring alternatives matters. In mini crosswords, local answers interact with each other, so the model must consider whether one choice makes other clues easier or impossible. In creative writing, choosing a writing plan shapes coherence across the final passage. In each case, a linear chain can lock the model into one path too early. Tree based reasoning creates a way to keep alternatives alive long enough to compare them.

This perspective raises an important question about arithmetic and symbolic tasks. Some arithmetic problems do not require branching search if the algorithm is known. Standard addition or subtraction is deterministic. But a small model may not have learned the algorithm. In that situation, scratch space might act like a limited search process. The model may generate intermediate steps and implicitly test whether they are consistent. If we add a verifier, the process

becomes closer to tree reasoning or planning feedback. This suggests that scratchpad experiments should not only compare answer accuracy. They should also examine whether models can revise intermediate states or recover from a wrong partial step.

The tree based view also creates a useful contrast with chain of thought. Chain of thought can be seen as one sampled path through a reasoning space. Self consistency samples many complete paths and votes on final answers, but it still does not explore alternatives locally inside each path. Tree based reasoning explores local alternatives and evaluates partial progress. This is a deeper kind of inference time computation. The cost is higher, but the mechanism is clearer. More computation is spent not simply on longer explanation, but on comparing possible routes. That distinction is central to my research direction: longer traces are not the same as structured exploration.

Long Reasoning Needs Depth, Exploration, and Reflection

The survey of long chain of thought helped me see modern reasoning models as part of a larger shift. Short chain of thought is usually shallow and linear. Long chain of thought is described as involving deeper reasoning, broader exploration, and feasible reflection (Chen et al., 2025). This taxonomy is useful because it separates different reasons why longer reasoning might help. A model might improve because it can make more logical deductions. It might improve because it explores alternatives. It might improve because it reflects on mistakes and revises. These are not the same mechanism, even though all of them can produce a longer answer.

This distinction matters because many discussions of reasoning focus too much on length. It is tempting to think that more tokens mean more intelligence, but the survey makes the debate more

careful. Inference time scaling can improve results when extra reasoning tokens are used productively. It can also create overthinking, inefficiency, or unnecessary complexity when the model spends tokens without making real progress (Chen et al., 2025). For a research project, this means I should not measure scratchpad success only by whether the scratchpad is longer. I should ask whether longer reasoning is carrying useful state, exploring new possibilities, or correcting errors.

The survey's future directions also helped me think beyond math tasks. Long reasoning is expanding toward multimodal reasoning, multilingual reasoning, agentic and embodied systems, efficient reasoning, knowledge augmented reasoning, and safety (Chen et al., 2025). Even though my own project would start with small arithmetic or symbolic tasks, the bigger field is moving toward systems that reason across images, languages, tools, and environments. Starting small is valuable precisely because the larger systems are hard to understand. A clean synthetic task can reveal mechanisms that become hidden inside complex real world agents.

The most useful part of this survey for me was its vocabulary. Deep reasoning is about the number and structure of logical steps. Exploration is about trying multiple paths or uncertain nodes. Reflection is about using feedback to refine earlier reasoning. Once I had these categories, I could look back at the other papers differently. Tree based reasoning emphasizes exploration. PDDL instruction tuning emphasizes verification and reflection. Auto CoT emphasizes the generation of useful reasoning examples. The arithmetic limitation paper exposes where shallow reasoning fails. The distribution paper asks whether these patterns transfer. The controllability paper asks whether reasoning traces remain observable and reliable for safety.

For small LLM experiments, this vocabulary suggests a clear design. A direct answer baseline has little visible depth, exploration, or reflection. A simple scratchpad may add depth but not

exploration. A tree or multi sample method may add exploration. A verifier guided training loop may add reflection. By separating these components, I can ask which one actually improves generalization. Maybe arithmetic mainly needs structured depth because carries and borrows must be tracked. Maybe planning needs reflection because actions must be checked against preconditions and effects. Maybe puzzle solving needs exploration because there are many possible paths. This turns a broad topic into a set of controlled experimental variables.

The Arithmetic Case Shows Why Small Tasks Matter

Arithmetic may look too simple for research on LLM reasoning, but the arithmetic paper made me see why simple tasks can be powerful. Addition has exact rules, automatic evaluation, and clear subcases. If a model fails, the failure can often be traced to a specific structural reason. The paper argues that autoregressive LLMs struggle with addition because they generate left to right, while the standard addition algorithm processes digits right to left and propagates carries (Baeumel et al., 2025). This mismatch creates a situation where the model must predict earlier result digits before knowing whether later positions will produce carries.

The paper's central claim is that models rely on a one digit lookahead heuristic. This can work fairly well for many two operand additions, because one nearby digit position often gives enough information to guess whether a carry will matter. But it fails when carries cascade or when multiple operands make the carry structure more complex. The paper uses probing and digit wise accuracy to show that failures occur exactly where the one digit lookahead is insufficient (Baeumel et al., 2025). This is a strong example of how a model can appear to perform arithmetic while not implementing the full algorithm.

This insight connects directly to my interest in small LLMs. If a small model learns addition from examples, high overall accuracy may hide a fragile heuristic. It may solve common cases while failing on carefully chosen hard cases. Therefore, evaluation must be broken down by case type. For addition, I would separate no carry, single carry, cascading carry, and multi operand cases. For subtraction, I would separate no borrow, single borrow, and cascading borrow. If a model performs well overall but fails badly on cascading cases, that says something important about its internal strategy.

The paper also suggests a specific reason scratch space might help. If the model is forced to compute from right to left inside the scratchpad, then the visible reasoning trace could align the generation process with the arithmetic algorithm. For example, the scratchpad could first compute the ones column, write the carry, then compute the tens column, and so on. This might reduce the need for hidden lookahead because the relevant intermediate state becomes explicit in the generated text. But this is only a hypothesis. The distribution paper warns that the model might learn the scratchpad format without learning the algorithm. That is exactly why this is a good experiment.

The broader lesson is that simple tasks can reveal deep limitations. A model that writes essays or solves some math word problems may still use shortcuts on basic arithmetic. A failure on addition is not just a failure of arithmetic. It reveals the difficulty of aligning a sequential token generator with an algorithm that requires different information flow. This makes arithmetic a useful microscope for reasoning research. It is not important because addition itself is hard for humans. It is important because it exposes how a model handles intermediate state, directionality, and generalization.

Distribution Shift Is the Main Stress Test

The distribution lens became one of the most important ideas in my reading. It argues that chain of thought succeeds or fails depending on how far the test query is from the training distribution. The paper studies this through task, length, and format shifts in a controlled environment called DataAlchemy, which abstracts tasks into atoms, elements, and transformations (Zhao et al., 2026). This setup matters because many evaluations of reasoning are messy. Real world benchmarks can contain data leakage, uncontrolled complexity, or hidden correlations. A synthetic environment makes it possible to isolate what changed and observe whether reasoning still works.

The most important claim is that chain of thought can look effective near the training distribution but become brittle under even moderate shifts. The paper reports that models can generate fluent yet logically inconsistent reasoning traces outside familiar distributions (Zhao et al., 2026). This is exactly the kind of warning that matters for scratchpad research. A model might learn to produce correct looking steps on familiar arithmetic formats, then fail when numbers become longer, operands increase, or the prompt format changes. If we only test in distribution, we may mistake interpolation for reasoning.

Task shift asks whether the model can handle a new operation or transformation. Length shift asks whether it can scale to longer inputs or longer reasoning chains. Format shift asks whether it can handle the same underlying problem when the surface presentation changes. These three axes give me a practical evaluation plan. In addition or subtraction, task shift might mean training on addition and testing related subtraction, or training on no carry and testing carry. Length shift might mean training on one digit and testing two or three digits. Format shift might mean changing from '23+58=' to 'What is 23 plus 58?' or changing the order of scratchpad labels.

The paper's finding about novel transformations is especially important. In some cases, models may appear to generalize because the new task shares superficial structure with old tasks. But when non commutative transformations remove easy shortcuts, performance can collapse (Zhao et al., 2026). This teaches us that evaluation examples must be designed to eliminate accidental success. If a model gets the right final answer through a shortcut, the trace may still be unfaithful. In arithmetic, this means we should include cases where surface patterns are misleading, such as examples that require cascading carries or borrows despite looking simple at the first digit.

The distribution lens also gives a balanced interpretation of positive chain of thought results. It does not say chain of thought never works. It says chain of thought may be a structured inductive bias learned from training data. That means it can be useful within the right distribution while still failing outside it. This is a more realistic view than calling CoT either real reasoning or a complete mirage: where does scratch space help, where does it fail, and which changes cause the failure?

The planning paper shifted my attention from generating reasoning to verifying reasoning. Planning is difficult because it requires actions to be valid in a sequence. An action may look reasonable in natural language but still be invalid if a precondition is not satisfied. A later state may look plausible but fail to follow from the effects of earlier actions. This makes symbolic planning a strong test of whether a model can maintain logical consistency over time. The PDDL based framework focuses on teaching models to reason about action applicability, state transitions, and plan validity through logical chain of thought instruction tuning (Verma et al., 2025).

The part I found most valuable is the use of external verification. Instead of trusting the model's own explanation, the framework uses VAL feedback to check whether state action state transitions

are valid. The paper distinguishes binary feedback from detailed feedback. Binary feedback only says valid or invalid. Detailed feedback gives more specific information about which preconditions failed or which effects were misapplied (Verma et al., 2025). This matters because detailed feedback can teach the model what went wrong, not just that something went wrong. It turns reasoning into a trainable verification process.

The reported results are meaningful because they show that logical reasoning can improve when it is tied to formal constraints. For Llama 3, the detailed feedback variant with more iterations reaches 94 percent in Blocksworld, 64 percent in Mystery Blocksworld, and 79 percent in Logistics (Verma et al., 2025). The different domains are important. Blocksworld tests a classic smaller action space. Mystery Blocksworld obscures predicate meanings, making surface semantic shortcuts less useful. Logistics tests transportation across locations, trucks, and airplanes. This domain variation connects back to distribution shift and generalization.

For my own research direction, the main lesson is that scratchpads should be checkable. If a model writes intermediate arithmetic steps, those steps can be verified automatically. If it writes a carry value, the carry can be checked. If it writes a borrow, the borrow can be checked. This creates a way to train or evaluate not only final answers but also intermediate reasoning. A direct answer model gives only success or failure. A verified scratchpad model gives a trace of where the failure occurred. That is much more useful for understanding what the model learned.

Planning also suggests that self correction should be treated carefully. A model may generate a plausible critique of its own answer without actually fixing the logical issue. External verification avoids that problem by grounding feedback in formal rules. In arithmetic, the verifier is simple: compute the true column values and compare them to the scratchpad. In symbolic tasks, the verifier

may be more complex. But the principle is the same. If we want reasoning traces to be more than fluent stories, we should connect them to constraints that can be tested.

Monitorability Adds a Safety Dimension

The paper on chain of thought controllability adds a safety perspective that I had not fully considered before. If reasoning traces are visible, they can potentially help humans or systems monitor what a model is doing. If a model is planning something problematic, the reasoning trace might reveal intent, intermediate goals, or warning signs. But this safety argument depends on the model not being able to strategically hide or reshape its chain of thought. The paper calls this ability CoT controllability and evaluates whether models can follow instructions about what may or may not appear in their reasoning (Chen Yueh-Han et al., 2026).

The surprising result is that reasoning models often control their final outputs much better than they control their chains of thought. The paper reports examples where CoT controllability is very low while output controllability is much higher, such as models that can obey constraints in the final answer but fail to maintain those constraints throughout the reasoning trace (Chen Yueh-Han et al., 2026). This difference matters because it suggests that the chain of thought is not just another output channel under the same level of deliberate control. It may expose internal tendencies that are harder for the model to suppress.

From a safety perspective, low CoT controllability can be cautiously positive because it may support monitorability. If models struggle to hide or reshape their reasoning, then chain of thought monitoring may remain useful. But the paper is careful not to overclaim. Low controllability is not the same as complete safety, and future models may change. The paper also notes that controllability can vary with model size, reasoning length, reinforcement learning pressure, and

domain (Chen Yueh-Han et al., 2026). This means monitorability should be tracked over time, not assumed permanently.

The controllability paper also changes how I interpret long reasoning. If longer reasoning makes the trace harder to control, that may increase monitorability in some cases but also creates more text to inspect. If reinforcement learning for final answer accuracy reduces controllability of the reasoning trace, then training pressure may change the relationship between internal computation and visible explanation. For a research project, this suggests adding trace level evaluations, not only answer accuracy. The final answer is only one signal. The intermediate text itself is a research object.

Four Ways Reasoning Can Fail

After reading across the papers, I started to see four broad ways reasoning can fail. First, a model can use a shallow heuristic instead of an algorithm. The arithmetic lookahead limitation is the clearest example. The model may solve many additions by approximating carry behavior with one digit lookahead, but fail when the task requires deeper carry propagation (Baeumel et al., 2025). This kind of failure is dangerous because high accuracy on easy cases can hide the absence of the real procedure.

Second, a model can follow a narrow distribution rather than a general rule. The distribution lens shows that chain of thought can work near the training data and collapse under task, length, or format shifts (Zhao et al., 2026). This failure can look like reasoning because the model produces

structured steps, but the structure may not transfer. The model knows how examples usually look. It does not necessarily know the transformation rule in a way that survives changes.

Third, a model can reason linearly when the task requires search. Tree based reasoning addresses this by maintaining multiple possible paths, evaluating them, and backtracking when needed (Yao et al., 2023). A linear chain is not always wrong, but it is fragile when early choices matter. If the first step is bad, the chain can continue confidently in the wrong direction. Search based methods are expensive, but they directly target this failure mode by refusing to commit too early.

Fourth, a model can generate fluent reasoning that is not verified. Planning makes this failure obvious because each action must satisfy preconditions and produce correct effects. Without verification, a plan can sound coherent but be invalid. The PDDL instruction tuning framework uses formal feedback to teach models to check steps, showing that reasoning improves when it is tied to external constraints (Verma et al., 2025). This failure mode applies to arithmetic too. A scratchpad that says a carry is 1 should be checked against the column sum. Fluency is not enough.

These four failure modes give me a research checklist. When a model succeeds, I should ask: did it learn the algorithm, or a heuristic? Did it generalize across distribution shifts, or stay near familiar examples? Did it explore alternatives when needed, or follow one path? Did its reasoning steps satisfy verifiable constraints, or only sound plausible? This checklist is one of the most important outcomes of my reading. It is not tied to one paper. It is a way to design experiments and interpret results.

Why Small Models Are the Right Place to Start

One reason I am excited about small LLM experiments is that they make these questions testable. Frontier models are powerful, but they are difficult to analyze. Their training data is huge, their

internal representations are complex, and their benchmark performance can be affected by unknown exposure. Small models trained from scratch on synthetic data are less impressive in raw ability, but they offer control.We can decide exactly what examples the model sees, what formats appear, what lengths are included, and what types of reasoning traces are used.

This matches the spirit of the controlled distribution approach. If the research question is whether scratch space improves generalization, then the training and test distributions must be designed intentionally. A small model can be trained on one digit no carry addition, then tested on one digit carry. It can be trained on two digit no borrow subtraction, then tested on cascading borrow. It can be trained on one prompt format, then tested on another. Because the task is synthetic, the evaluation can be exhaustive rather than sampled. Every possible input in a range can be tested.

Small models also make failure analysis more manageable. If a model with fewer parameters fails on a certain case, we can inspect examples, outputs, and intermediate scratchpads. we can compare direct answer and scratchpad versions under the same architecture. We can vary dataset proportions, such as mostly easy cases with a small percentage of hard carry cases. I can evaluate whether the model memorizes frequent patterns or learns a procedure. These experiments may not prove what frontier models do, but they can reveal principles about training, scratch space, and generalization.

The readings suggest that this kind of project is not too small to matter. Arithmetic looks simple, but the lookahead limitation paper shows that addition can expose structural issues in autoregressive generation. The distribution paper shows that controlled synthetic environments can reveal whether reasoning transfers. The planning paper shows that formal verification can improve reasoning. The long CoT survey shows that reasoning research needs clarity about depth, exploration, and reflection. A small model project can combine these ideas in a clean setting.

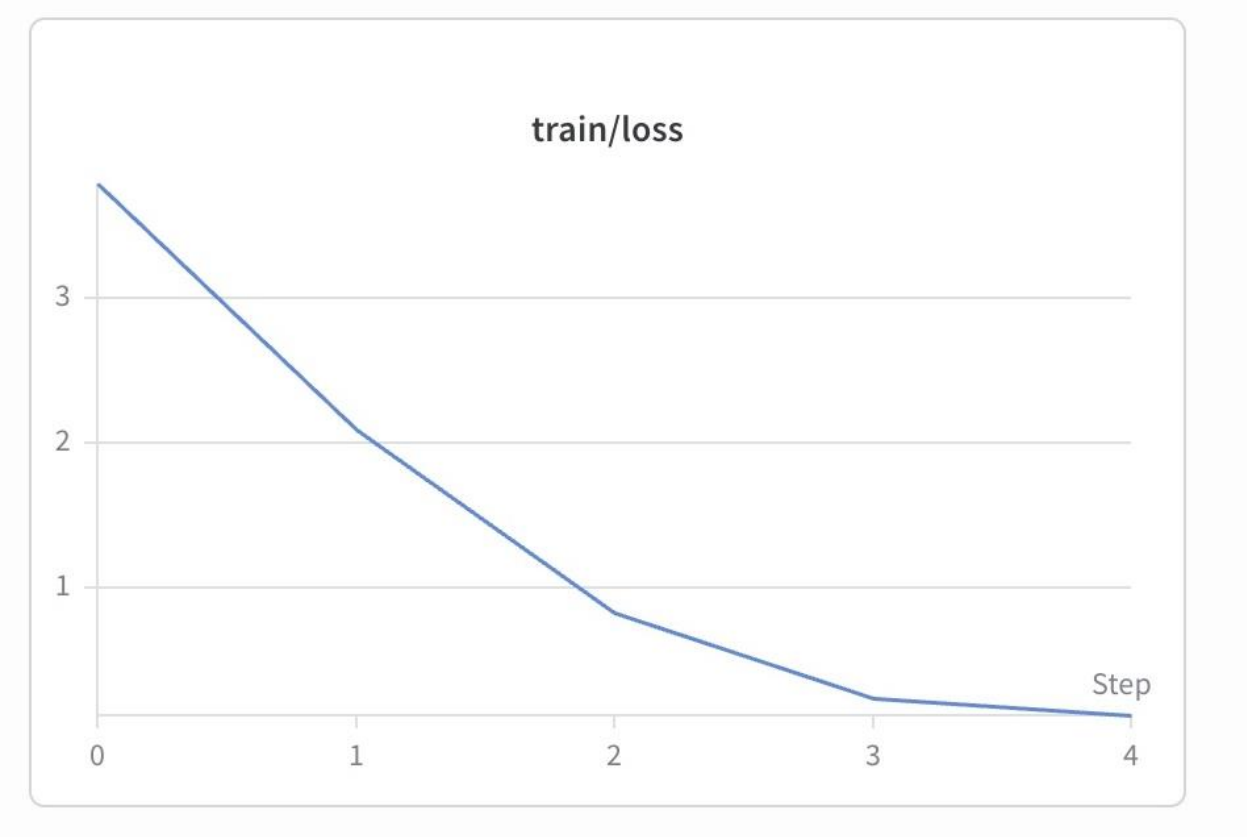
The goal would not be to build the strongest arithmetic model. The goal would be to understand what kind of training makes a model behave more algorithmically. Does explicit carry notation help? Does right to left scratchpad generation help? Does mixed difficulty training help or confuse the model? Does a verifier guided loss improve intermediate steps? Does the model generalize to longer numbers? These questions are small enough to test but connected to large debates about LLM reasoning.

Some of my LLM Experiments

1. **Emoticons Dataset**

```
alphabet = ['smile :)\n','sad :(\n','heart <3\n','wink ;)\n','tongue out :p\n','shocked :o\n','broken heart </3\n','hurray \o/\n','laugh :D\n','skeptical :/\n','kiss :x\n','happytears :"D\n','angry >:(\n','dead x_x\n','cry T_T\n','highfive o/\o\n','crab V=(o o)=V\n','cheerleader *\o/*\n','salute o7\n','surprise O_O\n']
```

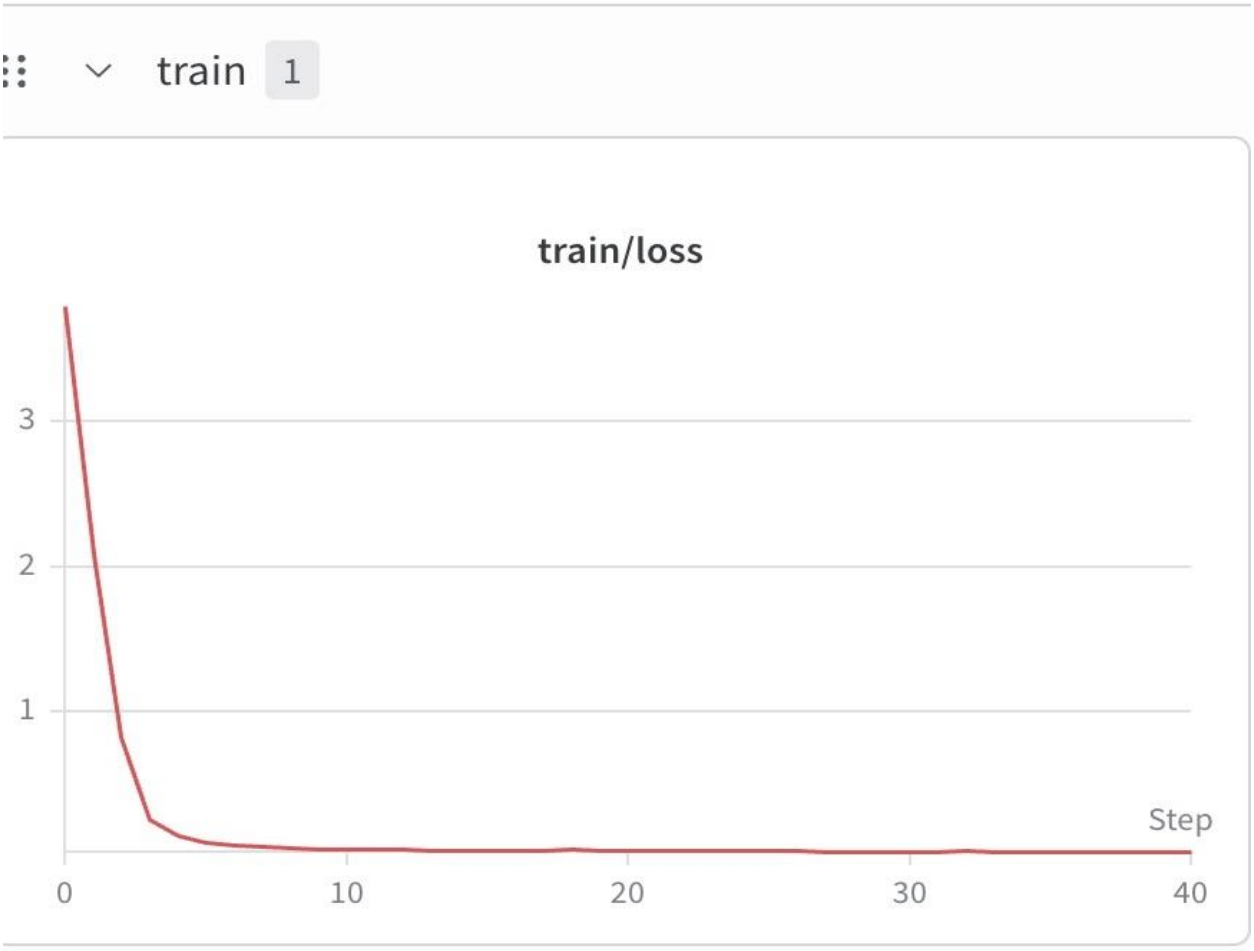
I manually made an emoticons dataset. At first, I ran the model for only 200 iterations so I could quickly check whether the setup was working correctly. The early output showed that the model was starting to learn the basic format, but it was still not consistent enough.



Start- train loss 3.7948, val loss 3.7875

End- loss 0.1113, time 267.28ms, mfu 0.06

After that, I increased the training to 2000 iterations, and the results became much better.



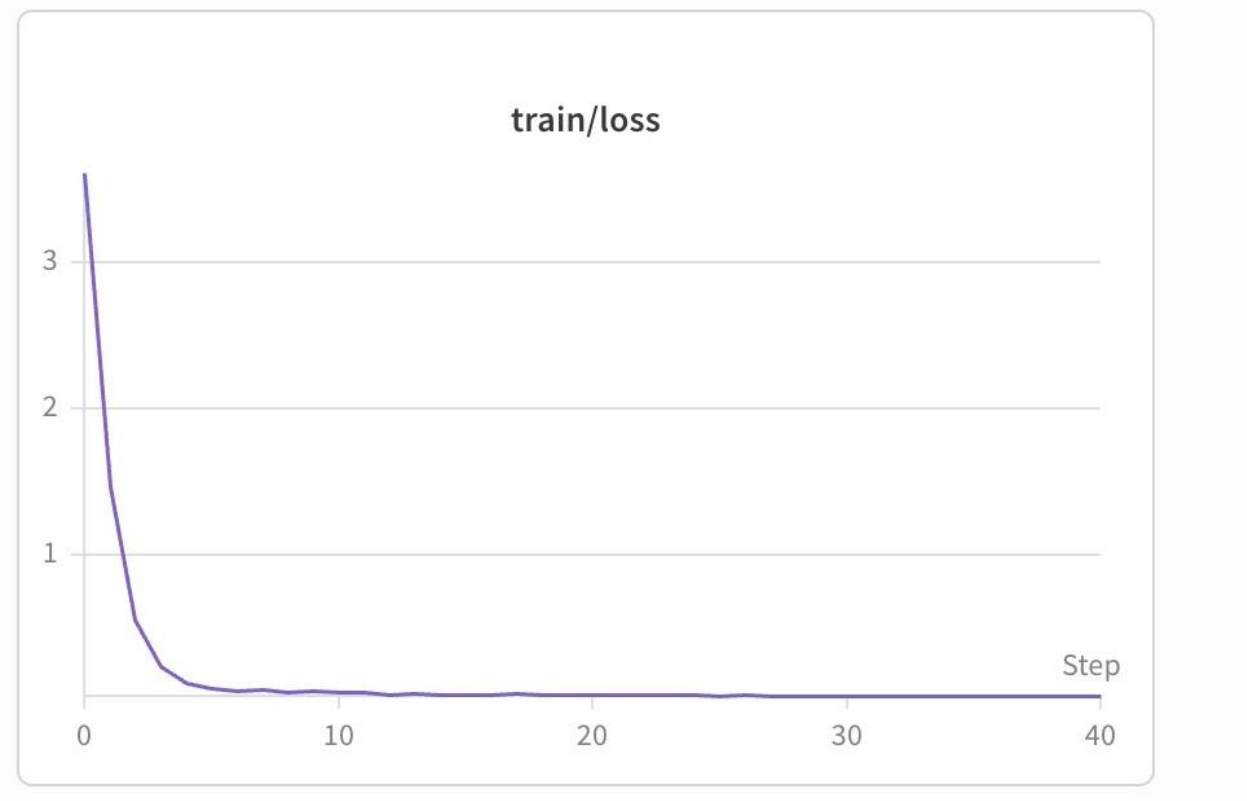
Start- train loss 3.7948, val loss 3.7875

End - loss 0.0347, time 295.71ms, mfu 0.06%

2. **WorldCupWinners**

```
alphabet = [
    "1930 Uruguay\n",
    "1934 Italy\n",
    "1938 Italy\n",
    "1950 Uruguay\n",
    "1954 Germany\n",
    "1958 Brazil\n",
    "1962 Brazil\n",
    "1966 England\n",
    "1970 Brazil\n",
    "1974 Germany\n",
    "1978 Argentina\n",
    "1982 Italy\n",
    "1986 Argentina\n",
    "1990 Germany\n",
    "1994 Brazil\n",
    "1998 France\n",
    "2002 Brazil\n",
    "2006 Italy\n",
    "2010 Spain\n",
    "2014 Germany\n",
    "2018 France\n",
    "2022 Argentina\n"
]
```

I manually made a new dataset of all previous years Fifa World Cup Winners and Runner Ups. At first, I ran the model for only 200 iterations so I could quickly check whether the setup was working correctly. The early output showed that the model was starting to learn the basic format, but it was still not consistent enough. Then I ran the model in 2000 iterations it worked perfectly well.



Start - train loss 3.6133, val loss 3.6097

End- loss 0.0424, time 297.53ms, mfu 0.06%

3. **Subtraction Experiment**

For subtraction experiment, I focused only on one-digit cases with no borrowing and no negative answers, so the operation was $a - b$where $a \geq b$. The goal was to train the model on simple one digit subtraction examples and then evaluate whether it could learn the pattern across all possible inputs. I ran the model in 200 iterations first, results weren't convincing enough so I ran it again in 2000 iterations and the model worked very well.

Start - train loss 2.6982, val loss 2.6999

End - loss 0.6778, time 246.00ms, mfu 0.05%

0-0=0
9-1=8
9-9=0
2-1=1
7-1=6
3-2=1
9-7=2
0-0=0
6-4=2
9-8=1
6-3=0
1-0=0
0-1=0
2-3=1
4-0=1
2-3=0

6-1=

(base) radiathpatwary@Radiaths-MacBook-Air-2 Subtraction_Experiment % python evaluation_sub.py --dataset 1digit --train_digits 1 --eval_digits 2
number of parameters: 0.79M
20-11=0
6 Correct Answer: 9
20-12=0
6 Correct Answer: 8
20-13=0
6 Correct Answer: 7
20-14=2- Correct Answer: 6
20-15=6- Correct Answer: 5
20-16=0
6 Correct Answer: 4
20-17=1
6 Correct Answer: 3
20-18=1

I will further proceed with my findings on two digit subtractions in my future research.

Conclusion

The main conclusion of this blog is that LLM reasoning should be judged by mechanisms rather than appearances. A long explanation is not automatically deep reasoning. A correct answer is not automatically algorithmic understanding. A chain of thought is not automatically faithful. A model that works on familiar examples has not necessarily generalized. These lessons appeared again and again across the readings, even though the papers studied different methods and tasks.

At the same time, the report does not argue that chain of thought is useless. The stronger conclusion is that chain of thought is conditionally useful. It can help when demonstrations are diverse, when reasoning is structured, when search is needed, when verification is available, and when the test setting remains close enough to learned patterns. It can fail when the task requires deeper state propagation, when distribution shifts remove shortcuts, when the model overthinks, or when fluent text hides logical inconsistency.

The research direction that grows from this conclusion is to test scratch space carefully in small controlled models. The most important question is not whether scratchpads improve average accuracy. The real question is whether scratchpads change the model's computational strategy and improve robustness on the hard cases that reveal reasoning. That means evaluating by case type, distribution shift, intermediate trace correctness, and response to verification.

My proudest achievement this semester is building that question. Through all these readings and small experiments, I went from absolutely no knowledge in LLMs, to a general interest in LLMs, and thus, to a potential research direction that I might pursue in the future about scratch space, generalization, and reasoning reliability. This can lead to a repository, a research report, or a future paper. More importantly, it gives me a way to keep learning about LLMs by doing research rather than only reading about it.

References

Baeumel, T., van Genabith, J., & Ostermann, S. (2025). The lookahead limitation: Why multi-operand addition is hard for LLMs. arXiv:2502.19981.

Chen, Q., Qin, L., Liu, J., Peng, D., Guan, J., Wang, P., Hu, M., Zhou, Y., Gao, T., & Che, W. (2025). Towards reasoning era: A survey of long chain-of-thought for reasoning large language models. arXiv:2503.09567.

Chen Yueh-Han, C., McCarthy, R., Lee, B. W., He, H., Kivlichan, I., Baker, B., Carroll, M., & Korbak, T. (2026). Reasoning models struggle to control their chains of thought. arXiv:2603.05706.

Verma, P., La, N., Favier, A., Mishra, S., & Shah, J. A. (2025). Teaching LLMs to plan: Logical chain-of-thought instruction tuning for symbolic planning. arXiv:2509.13351.

Yao, S., Yu, D., Zhao, J., Shafran, I., Griffiths, T. L., Cao, Y., & Narasimhan, K. (2023). Tree of thoughts: Deliberate problem solving with large language models. Advances in Neural Information Processing Systems, 36, 11809-11822.

Zhang, Z., Zhang, A., Li, M., & Smola, A. (2022). Automatic chain of thought prompting in large language models. arXiv:2210.03493.

Zhao, C., Tan, Z., Ma, P., Li, D., Jiang, B., Wang, Y., Yang, Y., & Liu, H. (2026). Is chain-of-thought reasoning of LLMs a mirage? A data distribution lens. arXiv:2508.01191.