Reasoning traces/Chain of Thought (CoT)

Review

We begin with a quick review of Chain of Thought.

- Chain of Thought (NLP_Beyond_LLM.ipynb#Chain-of-thought-prompting)

More formally:

Consider a task

- described by prompt $\backslash\mathbf{x}$ (a sequence)
- with response $\backslash\mathbf{y}$ (a sequence)

The most direct solution of the task can be described by the following sequence describing the LLM's computation

$$\backslash\mathbf{x}, \backslash\mathbf{y}$$

If the task is sufficiently complicated

- e.g., is best solved by multi-step reasoning

it has been shown that the chances of generating a better $\backslash\hat{\mathbf{y}}$ (sequence) answer are improved by "Chain of Thought" reasoning.

CoT summary

Given prompt

$$\backslash\mathbf{x}_{(1:\bar{T})}$$

rather than *immediately* producing response

$$\backslash\mathbf{y}_{(1:T)}$$

giving computation trace

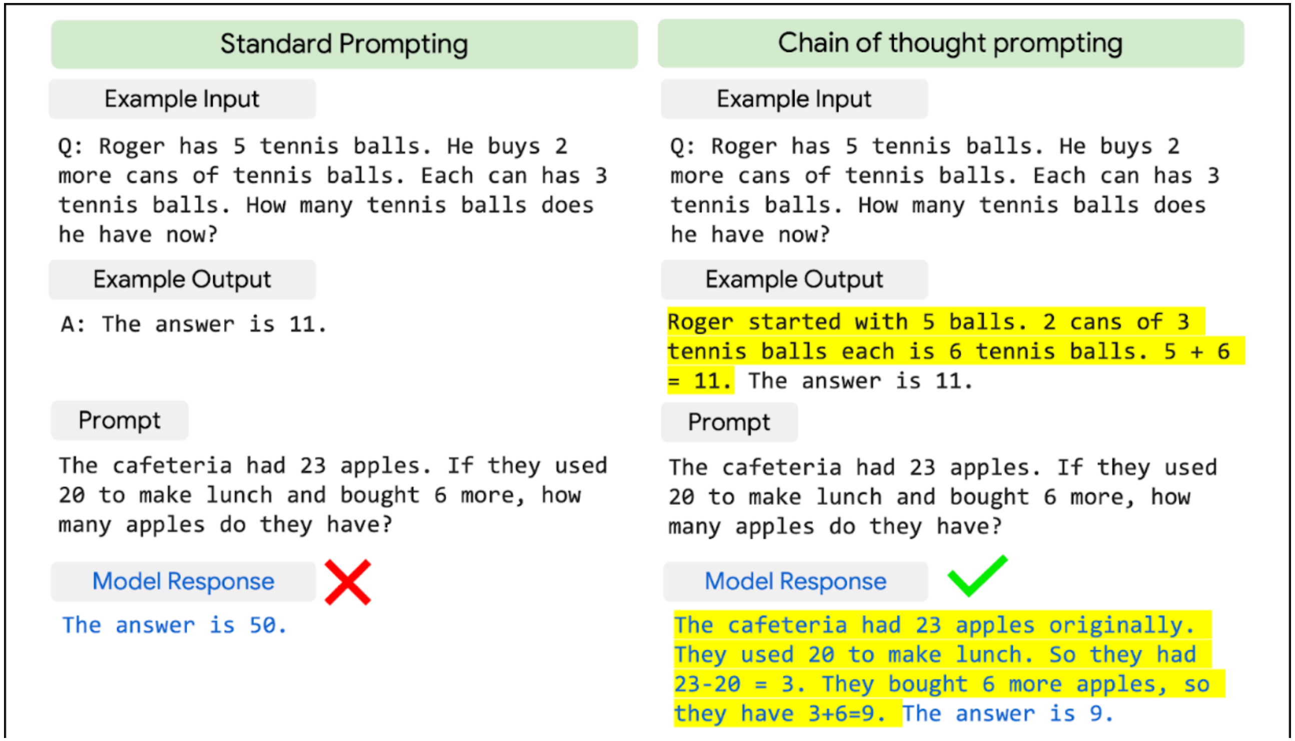
$$\backslash\mathbf{x}_{(1:\bar{T})}, \backslash\mathbf{y}_{(1:T)}$$

an LLM is trained to think "Step by Step"

In the chart below, compare

- Non Chain of thought: left side
- Chain of thought reasoning: right side

Chain of Thought Prompting



Standard Prompting

Example Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

Example Output

A: The answer is 11.

Prompt

The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

Model Response ✗

The answer is 50.

Chain of thought prompting

Example Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

Example Output

Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. 5 + 6 = 11. The answer is 11.

Prompt

The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

Model Response ✓

The cafeteria had 23 apples originally. They used 20 to make lunch. So they had 23-20 = 3. They bought 6 more apples, so they have 3+6=9. The answer is 9.

Attribution: https://arxiv.org/pdf/2201.11903.pdf

By thinking "step by step"

- the same LLM
- produces a correct answer

"Step by step" thinking

- creates a sequence of steps
- the Chain of Thought ("reasoning" trace) $\backslash\mathrm{rat}$
- enumerating sequential steps of a process that produces the response

$$\backslash\mathrm{rat} = [\backslash\mathrm{rat}_{(1)}, \ldots, \backslash\mathrm{rat}_{(\mathrm{num\_thoughts})}]$$

- where each $\backslash\mathrm{rat}_{\backslash\mathrm{tp}}$ is a thought represented as multi-token sequence

More formally, the LLM's "thought process" can be represented as a concatenation of

$$\x, \rat, \y$$

- prompt $\x$
- the *reasoning trace* $\rat$
    - a sequence of *thoughts* ordered linearly: the Chain of Thought

$$\rat = [\rat_{(1)}, \ldots, \rat_{(\text{num\_thoughts})}]$$

- response $\y$

Note that each thought $\rat_{\tp}$ is a sequence of tokens.

Thus, the model computes response $\backslash\mathrm{y}$ as

$$\backslash\mathrm{pr}\backslash\mathrm{y} \mid \backslash\mathrm{x}, \backslash\mathrm{rat}$$

rather than directly as

$$\backslash\mathrm{pr}\backslash\mathrm{y} \mid \backslash\mathrm{x}$$

The reasoning trace $\backslash\mathrm{rat}$

- *conditions* $\backslash\mathrm{y}$
- on the reasoning steps
- improving Performance

It is the conditioning on the reasoning step that results in more desirable responses.

A model that reasons: ”Think before speaking”

Rather than immediately outputting the "final" response

- Chain of Thought
- gives the model the ability to "think"
    - create reasoning traces
- before "speaking" the final answer

But reasoning can entail more than just a plan that leads to a single "final" answer.

It may also encompass *solution strategies"

- *reflection*
    - "reflect on" (i.e., evaluate)
    - intermediate solutions (reasoning traces that have not yet created an answer)
    - to determine whether the reasoning trace is on "the right track" to a good solution
- *revision*
    - abandon an intermediate solution
    - change the intermediate solution and extend

Notation

By convention, thoughts are bracketed by the "think" delimiter

\bthink

Thinking step-by-step, the first step is . . .

\ethink

Using this bracketing

- allows the model to behavior differently
    - when "thinking"
    - versus "speaking"

We will see this in a module on Reasoning in Latent Space.

Bracketing can also be used

- in combination with a user-controlled toggle
- to suppress output of the reasoning trace

when the user only wants to see the response

A Reasoning model in action

Here is an example.

The prompt is

```
Prove to me that the residuals of Linear Regression are not correlated with the
independent variables
```

Note

- the reasoning trace is truncated (ran out of credits on HF before I could write the entire file)
- output: highlight the first words of each paragraph
    - indicates
        - thinking step by step
        - revision, etc

In [3]:

```python
r1_out_file = 'r1_prove_residuals_orthogonal_full.txt'
print_file_content(r1_out_file)
```

<think>

Okay, so I need to prove that the residuals in linear regression aren't correlated with the independent variables. Hmm, let me start by recalling what residuals are. They're the differences between the observed values of the dependent variable and the values predicted by the regression model.
Right, so residuals = observed Y - predicted Y.

Now, in linear regression, we have the model Y = Xβ + ε, where X is the matrix of independent variables, β is the coefficient vector, and ε is the error term. The residuals are estimates of these errors, right?
But wait, the actual errors ε might have certain properties, like being uncorrelated with X if the model is correctly specified.
But the residuals are different because they're calculated after estimating β. So maybe there's something in the estimation process that ensures residuals are uncorrelated with X?

Let me think about the ordinary least squares (OLS) method. The OLS estimator minimizes the sum of squared residuals. The first-order conditions for this minimization might impose some orthogonality between the residuals and the independent variables. Oh, right! The normal equations in OLS state that X'(Y - Xβ_hat) = 0, where β_hat is the OLS estimator. That equation implies that the transpose of the X matrix multiplied by the residuals is zero.

Breaking that down, each column of X (each independent variable) is multiplied by the residuals and summed up, resulting in zero. So, the dot product of each independent variable with the residuals is zero. In statistical terms, that means the sample covariance between each independent variable and the residuals is zero, right? Because covariance is related to the dot product after centering the variables.

Wait, but if the variables are centered (mean zero), then the covariance is just the expectation of the product.

But even if they aren't centered, the normal equations still give X' residuals = 0. So, in the sample, the residuals are orthogonal to each of the independent variables. Therefore, their correlation should be zero.

But is this always true? Let me check with a simple example. Suppose I have one independent variable x and a dependent variable y. The OLS regression line is $y = \beta 0 + \beta 1 x$. The residuals are $e_i = y_i - (\beta 0 + \beta 1 x_i)$. The normal equations for this case are:

1. The sum of residuals is zero: $\Sigma e_i = 0$.
2. The sum of residuals multiplied by x_i is zero: $\Sigma e_i x_i = 0$.

The second equation directly implies that the sample covariance between x and e is zero because covariance is $[\Sigma(x_i - \bar{x})(e_i - \bar{e})] / (n-1)$.
But since $\Sigma e_i = 0$, $\bar{e} = 0$, and $\Sigma x_i e_i = 0$, so covariance is $[\Sigma x_i e_i - \bar{x} \Sigma e_i] / (n-1) = 0$. Therefore, the sample covariance is zero, which means the correlation is also zero.

So in the simple linear regression case, the residuals are uncorrelated with the independent variable. Extending this to multiple regression, each independent variable's covariance with the residuals is zero because of the normal equations. Therefore, all the correlations between residuals and each X variable are zero.

But wait, does this hold in the population or just in the sample? The OLS properties are about the sample, but if the model is correctly specified, the error term ε is uncorrelated with X in the population. However, the residuals are estimates of ε and have different properties.
But regardless of the population, in the sample, the residuals are constructed to be uncorrelated with X.

Another way to see this is through the Frisch-Waugh-Lovell theorem. If you regress Y on X and get residuals e, then those residuals are orthogonal to the space spanned by X. Therefore, any variable in X cannot explain any variation in e, which implies zero correlation.

But maybe there's a case where this doesn't hold? For example, if we don't include an intercept term?
Wait, if the model doesn't have an intercept, then the sum of residuals isn't necessarily zero.
But even then, the normal equations still require that X' e = 0. So even without an intercept, each column of X (including a constant column if present) is orthogonal to e. So if there's no intercept, the first column of X is all ones, and the normal equation would require that Σe_i = 0, but if there's no intercept, then that column isn't included.
Wait, no. If there's no intercept, then X doesn't have a column of ones. Then the normal equations would just be X' e = 0, where X is the matrix without the intercept. So in that case, the residuals are orthogonal to each of the independent variables included in the model, but the sum of residuals might not be zero. However, the correlation between residuals and each X variable would still be zero because their covariance is zero.

Wait, but if the variables aren't centered, does that affect the correlation? Correlation is covariance divided by the product of standard deviations. If covariance is zero, correlation is zero regardless of the scaling. So even if variables aren't centered, zero covariance implies zero correlation.

Therefore, in OLS, whether or not there's an intercept, the residuals are uncorrelated with each independent variable in the model. However, if the model is misspecified, say omitting a variable, then the residuals might be correlated with the omitted variable, but not with the included ones.

So putting this all together, the key point is that the normal equations from OLS ensure that the residuals are orthogonal to the space spanned by X, which includes each independent variable. Therefore, the sample covariance (and hence correlation) between each X and the residuals is zero.

I think that covers it. Let me just verify with a textbook or notes. Yes, in t

he OLS derivation, the normal equations X'e = 0 are crucial, and they directly lead to the orthogonality between residuals and regressors. Therefore, the residuals are uncorrelated with the independent variables in the sample. This is a property of the OLS estimation procedure, ensuring that all the explanatory power of X has been accounted for in the model, leaving the residuals uncorrelated with X.
</think>

To prove that the residuals of a linear regression are uncorrelated with the independent variables, we rely on the properties of Ordinary Least Squares (OLS) estimation. Here's a step-by-step explanation:

### **1. Model Setup**
Consider the linear regression model:
\[
\mathbf{Y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon},
\]
where:
- \(\mathbf{Y}\) is the dependent variable,
- \(\mathbf{X}\) is the \(n \times k\) matrix of independent variables (including a constant term if applicable),
- \(\boldsymbol{\beta}\) is the vector of coefficients,
- \(\boldsymbol{\varepsilon}\) is the error term.

The OLS estimator \(\hat{\boldsymbol{\beta}}\) minimizes the sum of squared residuals:
\[
\hat{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta}} \sum_{i=1}^n \left(Y_i - \mathbf{X}_i\boldsymbol{\beta}\right)^2.
\]

### **2. Normal Equations**

The first-order conditions for minimizing the sum of squared residuals lead to the **normal equations**:

$$
\mathbf{X}^\top (\mathbf{Y} - \mathbf{X}\hat{\boldsymbol{\beta}}) = \mathbf{0}.
$$

Let $\mathbf{e} = \mathbf{Y} - \mathbf{X}\hat{\boldsymbol{\beta}}$ denote the residuals. The normal equations simplify to:

$$
\mathbf{X}^\top \mathbf{e} = \mathbf{0}.
$$

### **3. Interpretation of $\mathbf{X}^\top \mathbf{e} = \mathbf{0}$**

This equation implies:

- For each column of $\mathbf{X}$ (i.e., each independent variable), the dot product with the residuals $\mathbf{e}$ is zero:

$$
\sum_{i=1}^n X_{ji} e_i = 0 \quad \text{for all } j = 1, 2, ..., k.
$$

### **4. Covariance and Correlation**

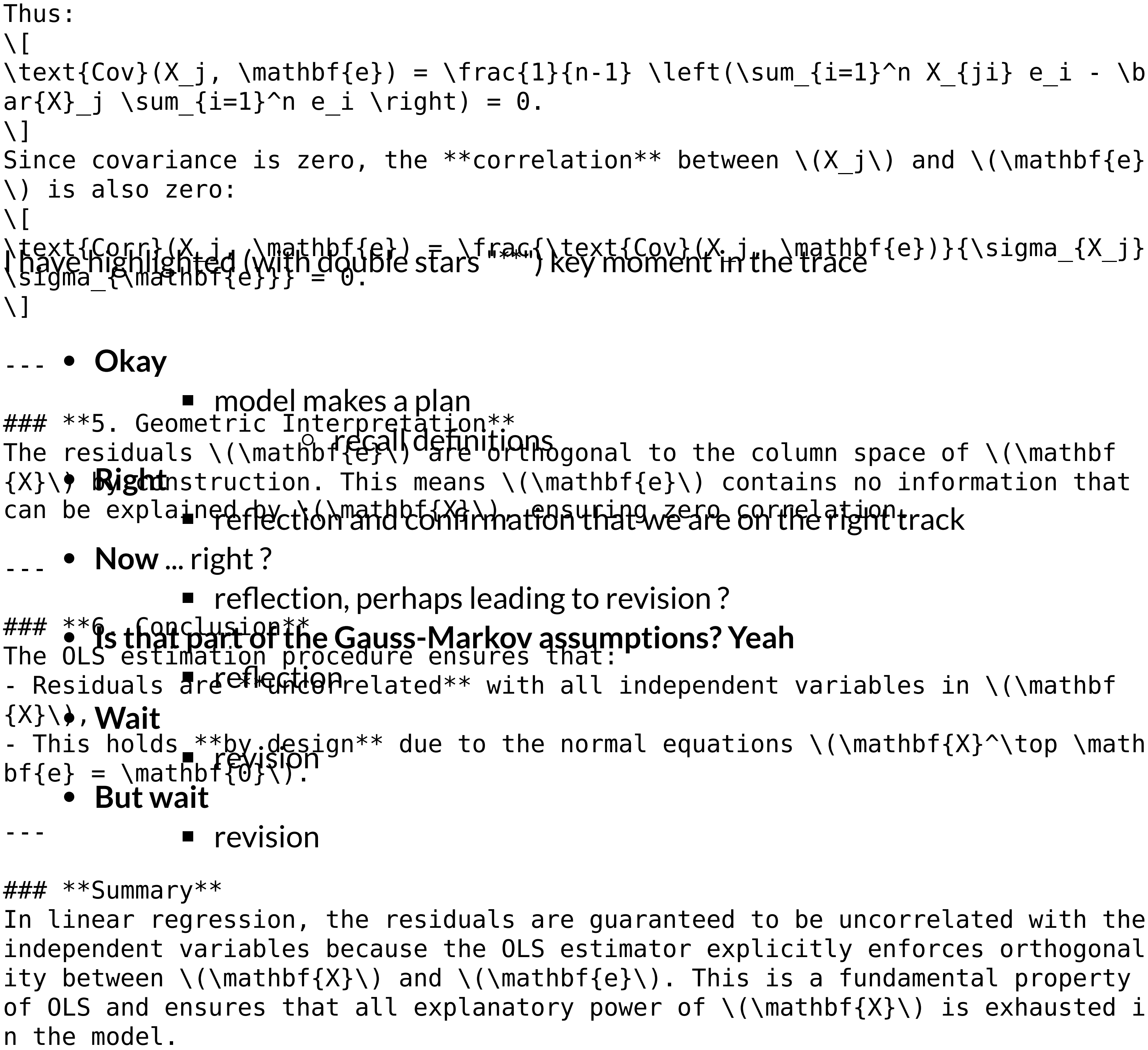
The sample covariance between an independent variable X_j and the residuals $\mathbf{e}$ is:

$$
\text{Cov}(X_j, \mathbf{e}) = \frac{1}{n-1} \sum_{i=1}^n (X_{ji} - \bar{X}_j)(e_i - \bar{e}).
$$

From the normal equations:

- $\sum_{i=1}^n X_{ji} e_i = 0$,
- If the model includes an intercept, $\sum_{i=1}^n e_i = 0$ (so $\bar{e} = 0$).

I have highlighted (with double stars "**") key moment in the trace



```
Thus:
\[
\text{Cov}(X_j, \mathbf{e}) = \frac{1}{n-1} \left(\sum_{i=1}^n X_{ji} e_i - \b
ar{X}_j \sum_{i=1}^n e_i \right) = 0.
\]
Since covariance is zero, the **correlation** between \(X_j\) and \(\mathbf{e}
\) is also zero:
\[
\text{Corr}(X_j, \mathbf{e}) = \frac{\text{Cov}(X_j, \mathbf{e})}{\sigma_{X_j}
\sigma_{\mathbf{e}}} = 0.
\]

---

### **5. Geometric Interpretation**
The residuals \(\mathbf{e}\) are orthogonal to the column space of \(\mathbf
{X}\) by construction. This means \(\mathbf{e}\) contains no information that
can be explained by \(\mathbf{X}\), ensuring zero correlation.

---

### **6. Conclusion**
The OLS estimation procedure ensures that:
- Residuals are **uncorrelated** with all independent variables in \(\mathbf
{X}\).
- This holds **by design** due to the normal equations \(\mathbf{X}^\top \math
bf{e} = \mathbf{0}\).

---

### **Summary**
In linear regression, the residuals are guaranteed to be uncorrelated with the
independent variables because the OLS estimator explicitly enforces orthogonal
ity between \(\mathbf{X}\) and \(\mathbf{e}\). This is a fundamental property
of OLS and ensures that all explanatory power of \(\mathbf{X}\) is exhausted i
n the model.
```

- **Okay**
    - model makes a plan
        - recall definitions
- **Right**
    - reflection and confirmation that we are on the right track
- **Now** ... right ?
    - reflection, perhaps leading to revision ?
- **Is that part of the Gauss-Markov assumptions? Yeah**
    - reflection
- **Wait**
    - revision
- **But wait**
    - revision

Thinking harder

The reasoning trace $\backslash\mathrm{rat}$ is a sequences of thoughts.

How long should a model "think" ?

- what is the length (measured in tokens of the reasoning traces $\backslash\mathrm{rat}$?

The initial approaches to "reasoning" favored

- long thoughts (thinking "harder" or "deeper")

So one approach to a better Reasoner is getting it to produce longer reasoning traces

Thinking budget

Reference

s1: Simple test-time scaling (https://arxiv.org/pdf/2501.19393)

An interesting approach (https://arxiv.org/pdf/2501.19393) is to force a Reasoner to obey a thinking budget for its reasoning traces

- measured in number of thoughts, or tokens
    - minimum
    - maximum

This approach uses Test-time Scaling (Test_time_compute.ipynb#Test-time-Scaling-Law)

Rather than training the Reasoner to obey a budget

- this approach **modifies the Inference loop**
    - the loop enforcing the auto-regressive behavior of the LLM
- to
    - force the model to continue inference if the reasoning trace is below budget
    - truncate inference when the budget is exceeded

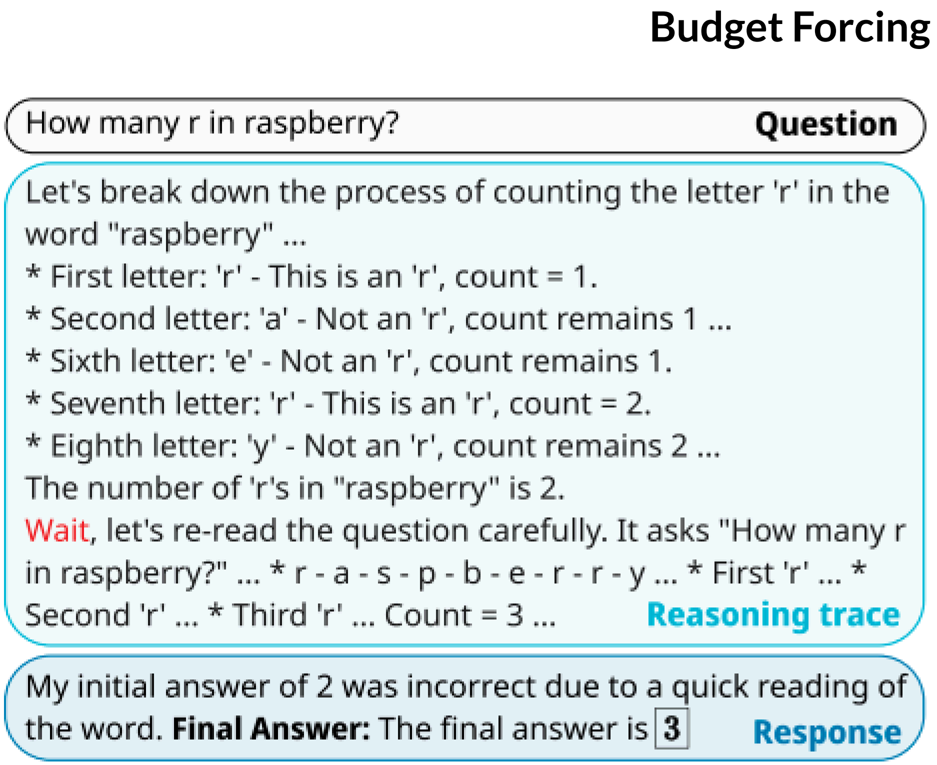
Forcing an overly long reasoning trace to adhere to maximum length is straight-forward

- Truncate the trace
- Insert an `<eos>` token

When the reasoning trace produced is too short

- the modified inference loop
    - replaces the token
    - with a token sequence that causes the model to revise/extend the reasoning trace
        - e.g., "Alternatively", "Wait", "But"
    - thus causing the LLM to continue "thinking"

Here is what the result looks like:



Budget Forcing

How many r in raspberry?

Let's break down the process of counting the letter 'r' in the word "raspberry" ...
* First letter: 'r' - This is an 'r', count = 1.
* Second letter: 'a' - Not an 'r', count remains 1 ...
* Sixth letter: 'e' - Not an 'r', count remains 1.
* Seventh letter: 'r' - This is an 'r', count = 2.
* Eighth letter: 'y' - Not an 'r', count remains 2 ...
The number of 'r's in "raspberry" is 2.
Wait, let's re-read the question carefully. It asks "How many r in raspberry?" ... * r - a - s - p - b - e - r - r - y ... * First 'r' ... * Second 'r' ... * Third 'r' ... Count = 3 ...

My initial answer of 2 was incorrect due to a quick reading of the word. **Final Answer:** The final answer is $\boxed{3}$

Figure 3. **Budget forcing with s1-32B.** The model tries to stop after "...is 2.", but we suppress the end-of-thinking token delimiter instead appending "Wait" leading **s1-32B** to self-correct its answer.

Attribution: https://arxiv.org/pdf/2501.19393#page=4

Code for enforcing a thinking budget

Here is some simplified code

- derived from the paper's [Github (https://github.com/simplescaling/s1/tree/main?tab=readme-ov-file#vllm-with-budget-forcing)](https://github.com/simplescaling/s1/tree/main?tab=readme-ov-file#vllm-with-budget-forcing)
- note some minor formating differences
    - does not explicitly use `<think>` and `</think>` to denote reasoning trace
    - instead, uses `<|im_start|>think`
        - denotes this part of the "conversation" is "thinking mode"

Note

- the multiple uses of the LLM `generate` method
- to generate the next part of the output
    - auto-regressive loop

Code: Budget Forcing

creating the reasoning trace

```
# Constants ignore_str = "Wait" max_tokens_thinking_tmp = MAX_TOKENS_THINKING
# Generate the start of the reasoning trace: Change the Assistant's role to **think**
prompt += "<|im_start|>think" o = model.generate( prompt,
sampling_params=sampling_params ) # Increase length of reasoning trace until length is
at least MAX_TOKENS_THINKING if max_tokens_thinking_tmp > 0: for i in
range(NUM_IGNORE): # Num of times to skip stop token # Append the last extension to
the reasoning trace prompt += o[0].outputs[0].text # Insert a "Wait" prompt +=
ignore_str # Generate the next extension of the trace o = model.generate( prompt,
sampling_params=sampling_params ) # Reduce the remaining number of thinking tokens
to generate max_tokens_thinking_tmp -= len(o[0].outputs[0].token_ids) ...
```

Code: Budget Forcing

creating the final answer

```
### Final answer ### # Append the last extension to the reasoning trace prompt +=
o[0].outputs[0].text # You can also append "Final Answer:" here like we do for some
evaluations # to prevent the model from just continuing to reason in its answer # when
early exiting # Create the "answer", which follows the reasoning trace ... o =
model.generate( prompt, sampling_params=sampling_params, ) print("With budget
forcing:") # You will see that after the "Wait" in the reasoning trace it fixes its answer
print(prompt + o[0].outputs[0].text)
```

Smarter not longer

At first glance

- longer reasoning trace should be preferred to a shorter trace
    - "deeper" reasoning

There is some empirical evidence that shows

- first preliminary response *can* often lead to correct response
- **but** the initial reasoning trace is often prematurely abandoned
    - the model "under-thinks" and tries something else if the first approach continues for too long

Why does a potentially successful initial reasoning trace get abandoned by the Reasoner ?

Perhaps it is in the training dataset

- hard problems in the training set have long but unsuccessful reasoning traces
- the model learns to abandon long traces

The problem is the inability to distinguish between

- long traces of hard problems that fail to be solved
- long traces that are needed for less-hard, solvable problems

How to think smarter

One approach is

- train a model to estimate the difficulty of a give task
- have the Reasoner adapt its test-time compute based on the difficulty
    - more/longer thoughts for harder tasks

Alternatively

- use a *Process Reward Model*
    - train a Reward Model to estimate
        - whether each step in the thought is advancing toward a good response
    - continue a thought only if the estimated reward is high

The training dataset for a Process Reward Model

- is expensive
- human labeling of the steps and rewards
    - reward limited to categorical (Positive/Negative/Neutral) versus continuous values

Training an LLM to reason

Basic Chain of Thought reasoning seems to be a property of models

- that emerges once models grow in size
- without explicit pre-training

Still, models need to be encouraged to employ Chain of Thought

- by appending "Let's think step by step"
    - zero-shot prompting, no exemplars demonstrating the chain of thought
- to the prompt

The exact phrase (e.g., "Let's think step by step") that best elicits this behavior is the subject of Prompt Engineering

- the module on Automated Prompt Engineer (Prompt_Engineering_APE.ipynb#Zero-shot:-Improving-on-%22Let's-think-step-by-step%22)

In order to produce an LLM that "reasons" better than basic Chain of Thought

- e.g., reflecting and revising

we need to instill this behavior by fine-tuning a pre-trained model.

Given prompt $\backslash\mathbf{x}$

- we fine-tune a model to produce responses
  $$\backslash\mathbf{rat}, \backslash\mathbf{y}$$
- rather than the direct responses
  $$\backslash\mathbf{y}$$

by fine-tuning it on examples of the form

$$\backslash\mathbf{x}, \backslash\mathbf{rat}, \backslash\mathbf{y}$$

- rather than
  $$\backslash\mathbf{x}, \backslash\mathbf{y}$$

The cost of producing examples with reasoning traces
\x, \rat, \y
can be substantial.

Often, it involves asking a human to adapt a training set with examples
\x, \y
by creating the reasoning traces \rat

Reducing the cost of creating examples with reasoning traces

Removing the human from the training data production process is highly desirable (cost reduction).

Boot-strapping a training dataset from an existing model is one solution.

One method of boot-strapping:

Ask a strong, non-reasoning LLM $\backslash\mathrm{model}^{\text{non-reasoner}}$ to solve task

- given $\backslash\mathbf{x}^{\backslash\mathrm{ip}}$
- "think step by step" (create $\backslash\mathbf{rat}^{\backslash\mathrm{ip}}$)
- produce response $\backslash\mathbf{y}^{\backslash\mathrm{ip}}$

Note that Chain of Thought ("think step by step") reasoning

- seems to emerge from the natural training examples used to train $\backslash\mathrm{model}^{\text{non-reasoner}}$
- does *not* require explicit training examples with reasoning traces
- hence, the boot-strapping process is well grounded
    - non-circular: the first model is trained without reasoning traces

Alternatively:

Ask an existing reasoning model $\model^{\text{reasoner}}$ to solve task

- given $\x^{\ip}$
- produce response $\rat^{\ip}$, $\y^{\ip}$

This effectively bootstraps from existing reasoner $\model^{\text{reasoner}}$

- base case: *someone* needs to (manually) create the dataset to train the first reasoner

Both these approaches creates a machine-generated example

$$\langle \backslash\mathbf{x}^{\backslash\mathrm{ip}}, \backslash\mathbf{rat}^{\backslash\mathrm{ip}}, \backslash\mathbf{y}^{\backslash\mathrm{ip}} \rangle$$

that can be used to train a new, reasoning LLM

Self-improvement

We can use this boot-strapping process iteratively to create a sequence of reasoners with increases strength.

See LLM Self-Improvement (LLM_Self_Improvement.ipynb) for the technique.

The iterative process of Self-Improvement

- trains the target model $\backslash\text{model}$ in stages
    - creating a sequence of fine-tuned models $\backslash\text{model}_{(0)}, \backslash\text{model}_{(1)}, \ldots$
    - of reasoners of increasing power
- base case
    - fine-tune non-reasoning initial $\backslash\text{model}_{(0)}$
    - use weak reasoning examples
        - from Chain of Thought or a weak reasoning model
    - resulting in weak reasoner $\backslash\text{model}_{(1)}$
- inductive case
    - create improved $\backslash\text{model}_{(+1)}$
    - by using reasoning traces created by $\backslash\text{model}_{\backslash\text{tp}}$
    - for fine-tuning

Distillation

Using an existing reasoning model $\backslash\mathrm{model}^{\mathrm{reasoner}}$

- to instill reasoning (by fine-tuning)in a non-reasoning model
- with examples of reasoning traces created by $\backslash\mathrm{model}^{\mathrm{reasoner}}$
- as we did above

is an illustration of the technique called *Distillation*

Given a model $\backslash\text{model}^{\text{student}}$ that has *not yet been fine-tuned* for the Target task T

- we adapt $\backslash\text{model}^{\text{student}}$ into model that solves the target task T
- by fine-tuning it on examples $\langle\backslash\dot{\mathbf{X}}, \backslash\dot{\mathbf{y}}\rangle$ of the target task's input/output relationship
    - creating a feature vector $\backslash\dot{\mathbf{x}}^{\backslash\text{ip}}$ for the Target task
    - using $\backslash\text{model}^{\text{teacher}}$ to create the response $\backslash\dot{\mathbf{y}}^{\backslash\text{ip}}$
    - for $1 \leq i \leq m$

Note that

- $\backslash\mathrm{model}^{\mathrm{student}}$ does not directly learn the task
- it only learns to mimic $\backslash\mathrm{model}^{\mathrm{teacher}}$
- inheriting all the flaws and limitations of the teacher

Reducing the cost of running a reasoning model

Often

- $\backslash\mathrm{model}^{\mathrm{teacher}}$ is a powerful model (large number of parameters, lots of training)
- $\backslash\mathrm{model}^{\mathrm{student}}$ is smaller and untrained in the Target task

Using Distillation to transfer skills from a large to a smaller model

- results in very powerful small $\backslash\mathrm{model}^{\mathrm{student}}$
- that may not have had the capability to be *directly trained* in the skill
- using only the training data used to train $\backslash\mathrm{model}^{\mathrm{teacher}}$ in the skill
    - Scaling Law: a model with fewer parameter needs more data to train than a model with more paramters

DeepSeek-R1 is a reasoning model with 671 B parameters.

DeepSeek-R1 was used to create smaller distilled models

- from small models
    - of sizes: 1.5B, 7B, 8B, 14B, 32B, 70B)
    - from the Llama and Qwen families of models

FYI

DeepSeek-R1

- derived from non-reasoning model DeepSeek-V3-Base of the same size
- using a mixture of Reinforcement Learning (RL) and Supervised Fine Tuning (SFT)
- Note
    - both models are Mixture of Experts (MoE)
    - only 37 B parameters are "activated" during inference

```
In [4]: print("Done")
```

```
Done
```