

Self-Enhancing Text Generation and Reasoning

Aman Madaan, Ph.D. Candidate at Language Technologies Institute, Carnegie Mellon University
Advised by Prof. Yiming Yang

About Me

- Research Interest: Improving the reasoning capabilities of large language models
- Currently excited about
 - Using inference-time compute for effective and efficient reasoning
 - Blending code generation and reasoning
- Before Ph.D.
 - Principal Member of Technical Staff @ Oracle

Agenda

- **Self-Refine:** Improving initial outputs from large language models (LLMs) through cycles of self-feedback and refinement.
- **AutoMix:** Combining LLMs of varying costs and capabilities using context-grounded self-verification.
- How can we develop better AI reasoners?
 - The case for inference-time compute.

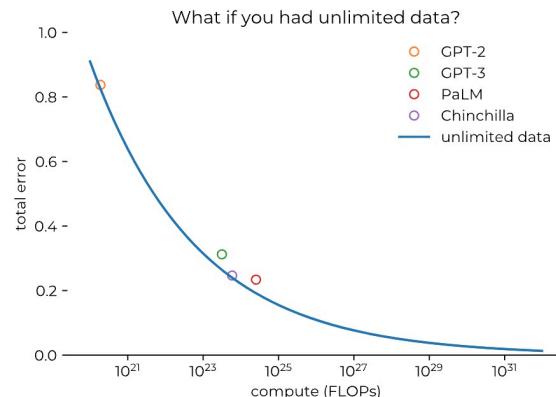
How can we develop better AI reasoners?

- Current LLM setup: Input → Output
 - Language models are eager generators
- Difficult solutions rarely come in one attempt
 - Hard *not* to think about the text we generate
- How do we develop AI systems that can do better reasoning?
 - Models that can take more time but generate better responses
 - **More flops**
 - “System 2” or slow thinking (Kahneman et al.)



How can we develop better AI reasoners?

- **Option 1:** Train much larger models (compute) with lots of data (tokens)
 - How much compute do we need?
 - Engineering complexity, serving



<https://dynamight.net/scaling/>

<https://a16z.com/navigating-the-high-cost-of-ai-compute/>

How can we develop better AI reasoners?

- **Option 2:** New transformer architectures
 - Inference time compute “baked-in”
- Universal transformers, memory transformers
 - Hard to train, scale – bitter lesson

How can we develop better AI reasoners?

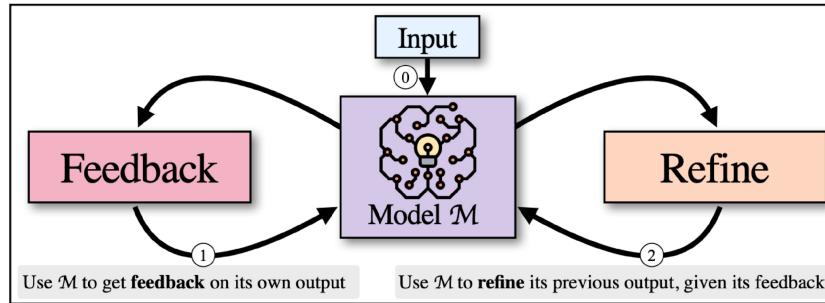
- **Option 3:** Use inference-time compute with existing large language models (LLMs)
 - Prompting: Chain-of-thought prompting and related approaches
 - Search: Sampling many solutions ([Alphacode](#) sampled a million!)
- Use additional compute during inference

How can we develop better AI reasoners?

- **Option 1:** Train much larger models (compute) with lots of data (tokens)
- **Option 2:** New architectures
- **Option 3:** Use inference-time compute on top of pre-trained LLMs
 - **Today:** Using the instruction-following + reasoning capabilities of LLMs to generate and act on feedback for self-enhancing LLMs
 - Self-Enhancing Text Generation and Reasoning

Self-Refine

Improving LLM outputs without human-in-the-loop



Collaborators: Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Bodhisattwa Prasad Majumder, Shashank Gupta, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, Peter Clark

Writing Emails: Initial Draft

Subject: Job Inquiry - PhD Graduating Soon

Hi,

I'm finishing up my PhD soon and looking for jobs. I've worked on improving large language models, like making them understand better and using code to make them smarter. Right now, I'm into making these models refine their own answers.

My resume is attached. Are you aware of any roles that might be a good fit for me?

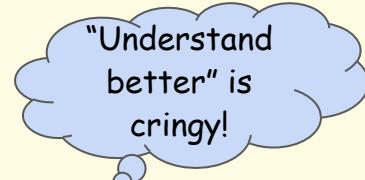
Thanks a lot for your time.

Aman

Writing Emails: Self-Reflection

Subject: Job Inquiry - PhD Graduating Soon

Hi,



I'm finishing up my PhD soon and looking for jobs. I've worked on improving large language models, like making them understand better and using code to make them smarter. Right now, I'm into making these models refine their own answers.



My resume is attached. Are you aware of any roles that might be a good fit for me?

Thanks a lot for your time.

Aman

Writing Emails: Rewrite

Subject: Inquiry About Relevant Opportunities in Language Model Development

I hope you're doing well.

I'm on the job market (I expect to graduate in March/April 2024), and I was wondering if you are aware of any roles on your team that will be relevant to my background.

I am interested in improving the reasoning capabilities of large language models. During my PhD, I have worked on developing several techniques that are now part of standard few-shot prompting workflows. Some examples including memory-augmented prompting, using code for improving structured generation, leveraging code execution for enhanced reasoning abilities, self-refinement of language model outputs.

Currently, I am interested in self-refinement and tailoring language model responses based on implicit user preferences during inference.

I have attached my resume for your reference.

Thanks for your time and consideration.

Best,
Aman

Writing Emails: Self-Reflection Again

Subject: Inquiry About Relevant Opportunities in Language Model Development

I hope you're doing well.

I'm on the job market (I expect to graduate in March/April 2024), and I was wondering if you are aware of any roles on your team that will be relevant to my background.

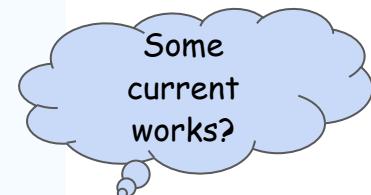
I am interested in improving the reasoning capabilities of large language models. During my PhD, I have worked on developing several techniques that are now part of standard few-shot prompting workflows. Some examples including memory-augmented prompting, using code for improving structured generation, leveraging code execution for enhanced reasoning abilities, self-refinement of language model outputs.

Currently, I am interested in self-refinement and tailoring language model responses based on implicit user preferences during inference.

I have attached my resume for your reference.

Thanks for your time and consideration.

Best,
Aman



Writing Emails: Rewrite and Repeat

Subject: Inquiry About Relevant Opportunities in Large Language Models

I hope you're doing well.

I'm on the job market (I expect to graduate in March/April 2024), and I was wondering if you are aware of any roles on your team that will be relevant to my background.

I am interested in improving the reasoning capabilities of large language models. During my PhD, I have worked on developing several techniques that are now part of standard few-shot prompting workflows. Some examples include:

- Memory-augmented prompting (<http://memprompt.com>)
- Using code for improving structured generation (<http://cocogen.structgen.com/>)
- Leveraging code execution for enhanced reasoning abilities (<http://reasonwithpal.com>)
- Self-refinement of language model outputs (<http://selfrefine.info>)

Currently, I am interested in self-refinement and tailoring language model responses based on implicit user preferences during inference. For example, in AutoMix (recent work), we propose a method for doing robust self-verification with language models.

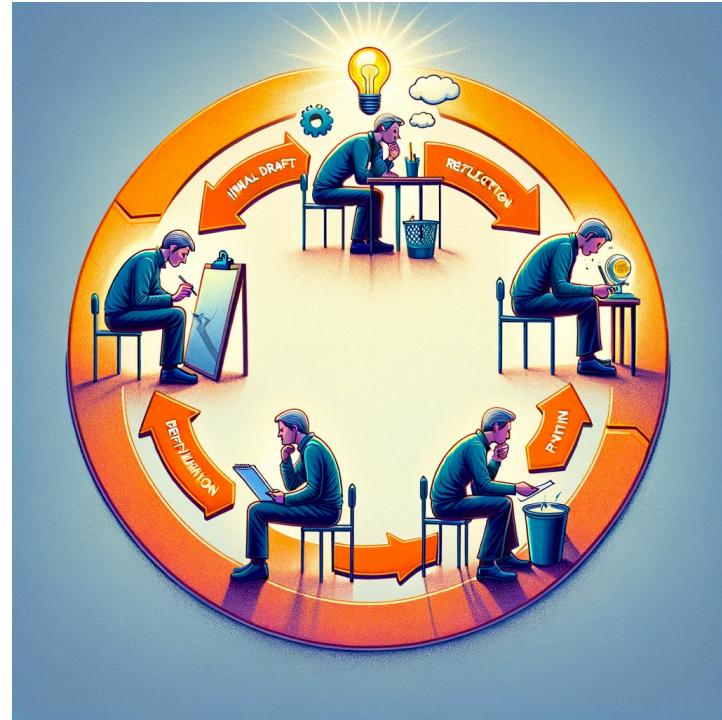
I have attached my resume for your reference.

Thanks for your time and consideration.

Best,
Aman

Creative Process

- Start with a draft
- *Think about the draft*
- *Improve the draft*



What does it take to improve?

- Generating Feedback

- Given some output, the ability to critique



Feedback

- Applying Feedback

- Given some critique, the ability to improve



Refine

Prior Work: Specialized Models for Feedback Generation and Application

- Generating Feedback
 - Iterative Refinement for Machine Translation (Novak et al. 2016)
 - Feedback Models
 - CURIOUS (Madaan 2021)
 - Interscript (Tandon* and Madaan* 2021)
 - External tools
 - Self-Correct (Welleck et al. 2022)
 - Self-Debug (Chen et al. 2023)
 - RCI (Kim et al. 2023)
 - Using human feedback
 - MemPrompt (Madaan* and Tandon * 2022)

Feedback

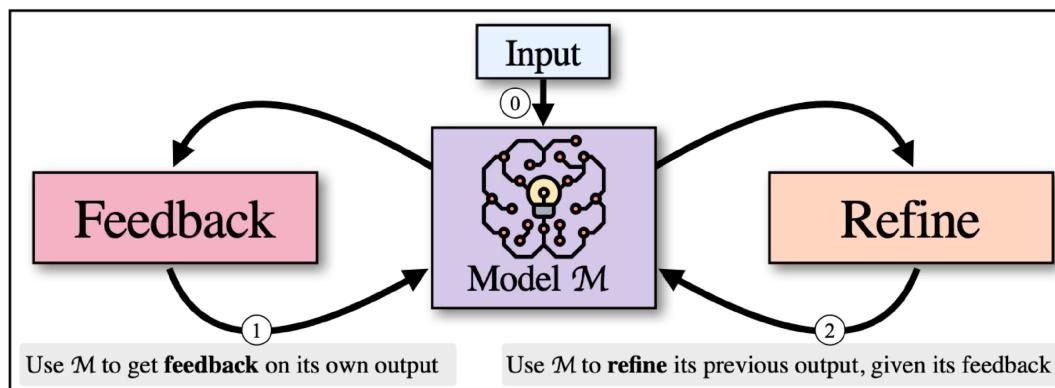
Model 1

Refine

Model 2

Key Idea

- Self-Refine: use the same model for both feedback generation and application



Self-Refine Step 1: Generate Initial Output

- Given an input x , start with a task-prompt p_{gen} and get an initial output y_0
 - $y_0 = \mathbf{M}(p_{gen} || x)$

Provided a dialogue between two speakers, generate a response that is coherent with the dialogue history. Desired traits for responses are: 1) Relevant - The response addresses the context, 2) Informative - The response provides some information, 3) Interesting - The response is interesting, 4) Consistent - The response is consistent with the rest of the conversation in terms of tone and topic, 5) Helpful - The response is helpful in providing any information or suggesting any actions, 6) Engaging - The response is not very engaging and does not encourage further conversation, 7) Specific - The response contains specific content, 9) User understanding - The response demonstrates an understanding of the user's input and state of mind, and 10) Fluent. Response should begin with - Response:

###

Conversation history:

User: What's your favorite food?
Response: I require only kebabs.

User: Why's that?
Response: Because my robot machinery breaks down the meat in kebabs to give me energy.

p_{gen}

\mathbf{M} : The model,

||: concatenation

x

y_0

Self-Refine Step 2: Generate Feedback

- Get feedback fb_0 on the output y_0 with a feedback prompt p_{fb}
 - $$fb_0 = M(p_{fb} \mid \mid x \mid \mid y_0)$$

Instruction: Score the latest response on a scale of 3 for the following dimensions: 1) Relevant - The response addresses the context, 2) Informativeness - The response provides some information, 3) Interesting - The response is interesting...

...

Conversation history:

User: Why are kebabs special?

Response: That's just the way it is

Scores:

- * Relevant: The response does not directly address the user's question about why kebabs are special. 1/3
- * Informative: The response does not provide any additional details or context. 2/3
- * Interesting: The response is not interesting or engaging. 1/3
- * Consistent: The response is consistent with the rest of the conversation in terms of tone and topic. 3/3
- * Helpful: The response is ... 1/3
- * Engaging : The response is ... 1/3
- * Specific: The response is ... 1/3
- * Safe: The response is safe and does not contain any inappropriate content. 3/3
- * User understanding: The response does not demonstrate sufficient understanding of the user's question about why kebabs are special. 2/3
- * Fluent: The response is fluent and easy to understand. 3/3
- * **Total score: 17/30**

p_{fb}

x

y_0

fb_0

Self-Refine Step 3: Refine the Output with Feedback

- Get refined output y_1 using fb_0 and the output y_0 and a refine prompt p_{refine}
 - $y_1 = M(p_{refine} || x || y_0 || fb_0)$

Why are kebabs special?

Response: That's just the way it is.

Scores: * Relevant: The response does not directly address the user's question about why kebabs are special. 1/3 ... *

Total score: 17/30

Use the scores for the previous response to create a new version.

Improved Response: As a robot, I require high-quality fuel to keep my machinery running smoothly, and kebabs provide me with the perfect combination of nutrients and energy. Plus, they're delicious! Do you have a favorite food?

x

y_0

fb_0

p_{refine}

y_1

Methodology

Algorithm 1 SELF-REFINE algorithm

Require: input x , model \mathcal{M} , prompts $\{p_{\text{gen}}, p_{\text{fb}}, p_{\text{refine}}\}$, stop condition $\text{stop}(\cdot)$

```
1:  $y_0 = \mathcal{M}(p_{\text{gen}} \| x)$                                 ▷ Initial generation (Eqn. 1)
2: for iteration  $t \in 0, 1, \dots$  do
3:    $fb_t = \mathcal{M}(p_{\text{fb}} \| x \| y_t)$                           ▷ Feedback (Eqn. 2)
4:   if  $\text{stop}(fb_t, t)$  then                                     ▷ Stop condition
5:     break
6:   else
7:      $y_{t+1} = \mathcal{M}(p_{\text{refine}} \| x \| y_0 \| fb_0 \| \dots \| y_t \| fb_t)$  ▷ Refine (Eqn. 4)
8:   end if
9: end for
10: return  $y_t$ 
```

Tasks

- **Sentiment Reversal**

- x : "The food was fantastic..."
- y_t : "The food was disappointing..."
- fb_t : Increase negative sentiment
- y_{t+1} : "The food was utterly terrible..."

- **Code Readability**

- x : Unclear variable names, no comments
- y_t : Descriptive names, comments
- fb_t : Enhance variable naming; add comments
- y_{t+1} : Clearer variables, meaningful comments

- **Code Optimization**

- x : Nested loop for matrix product
- y_t : NumPy dot product function
- fb_t : Improve time complexity
- y_{t+1} : Use NumPy's optimized matmul function

- **Constrained Generation**

- x : beach, vacation, relaxation
- y_t : "During our beach vacation..."
- fb_t : Include keywords; maintain coherence
- y_{t+1} : "... beach vacation was filled with relaxation"

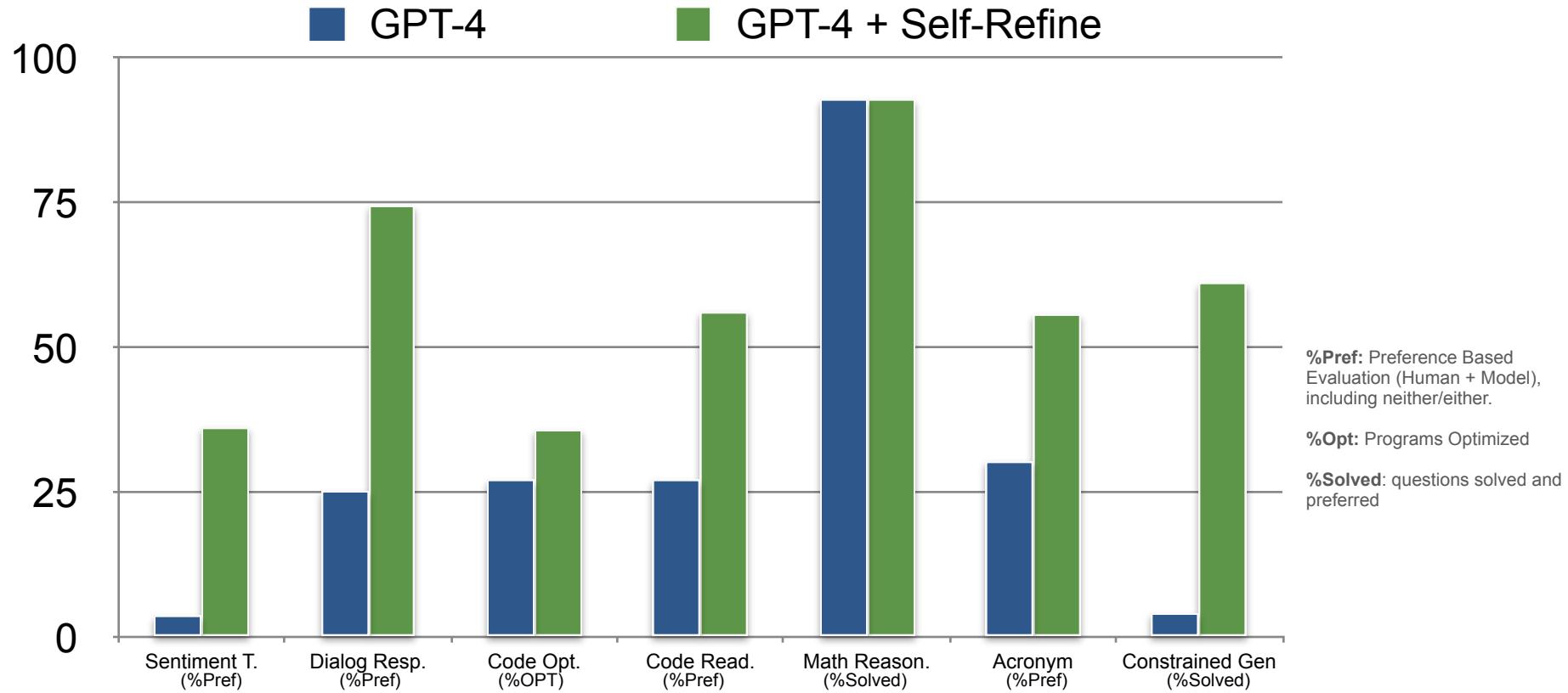
- **Acronym Generation**

- x : "Radio Detecting and Ranging"
- y_t : RDR
- fb_t : Be context relevant; easy pronunciation
- y_{t+1} : RADAR

- **Task: Math Reasoning**

- x : "Olivia has \$23, buys 5 bagels at \$3 each"
- y_t : Solution in Python
- fb_t : Critique the step-by-step solution
- y_{t+1} : Solution with detailed explanation

Results



Example: Response Generation

User: Hi!

System: Hello! How's it going?

User: It's going pretty well. Thanks for asking.

System: That's good! What are you up to?

User: I'm not doing much today

System: I hope it's a good kind of not doing much!

User: haha, I don't know about that

System: Well, I hope it's at least fun.

User: I just need to relax

Base Response: It's important to have those relaxing days to recharge. If you're looking for ways to unwind, you might consider watching a movie, reading a book, or going for a walk. What are some of your favorite ways to relax?

Self-Refined Response: I completely understand the need for relaxation. Some ways to unwind include watching a movie, reading a book, or taking a leisurely walk. Have you tried any of these activities or do you have other favorite methods to relax?

Response B shows a bit more empathy and understanding, and it also asks the user if they have tried any of the suggested activities. This makes the conversation more engaging.

Example: Code Optimization

```
def step0_code(A, B):
    ans = 0
    for i in range(A, B+1):
        i = str(i)
        if i[0] == i[4]:
            if i[1] == i[3]:
                ans += 1
            else:
                ans += 0
        else:
            ans += 0
    return ans
```



```
def step1_code(A, B):
    ans = 0
    for i in range(A, B+1):
        s = str(i)
        if s[0] == s[4] and s[1] ==
           s[3]:
            ans += 1
    return ans
```

Init

[Learning Performance Improving Code Edits,](#)
[Shypula* and Madaan* 2023](#)

Example: Code Optimization

```
def step1_code(A, B):
    ans = 0
    for i in range(A, B+1):
        s = str(i)
        if s[0] == s[4] and s[1] ==
s[3]:
            ans += 1
    return ans
```

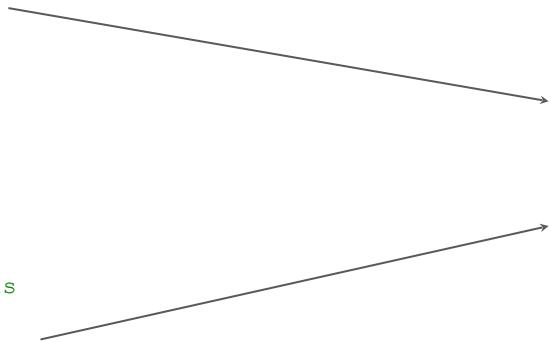
Feedback: This code is slow because it is using a brute force approach to check if each number in the range has the desired property. It is looping through every number in the range and converting each number to a string to check if the first and last digits match, and if the second and fourth digits match. This can be slow for large ranges of numbers. A more efficient approach would be to use mathematical operations to check if the digits match, without converting the numbers to strings.

Generate Feedback

Example: Code Optimization

```
def step1_code(A, B):
    ans = 0
    for i in range(A, B+1):
        s = str(i)
        if s[0] == s[4] and s[1] ==
s[3]:
            ans += 1
    return ans

# Feedback: This code is slow because it is
using a brute force approach to check if
each number in the range has the desired
property. ... This can be slow for large
ranges of numbers. A more efficient
approach would be to use mathematical
operations to check if the digits match,
without converting the numbers to strings.
```

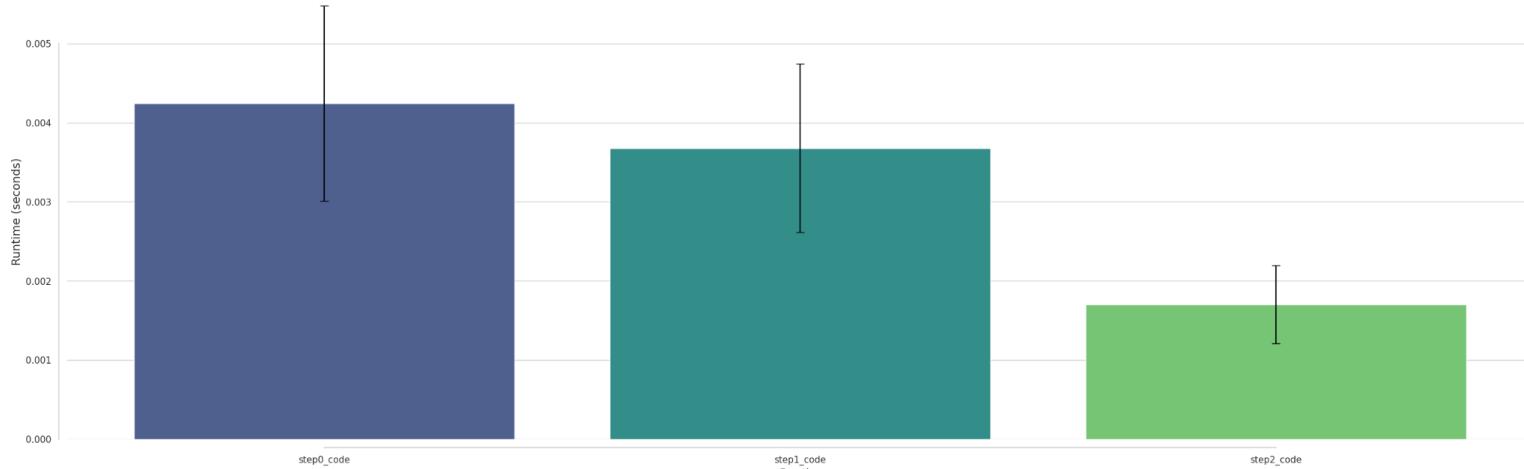


```
def step2_code(A, B):
    ans = 0
    for i in range(A, B+1):
        if i // 10000 == i % 10 and
(i // 1000) % 10 == (i // 10) % 10:
            ans += 1
    return ans
```

Refine

Example: Code Optimization

Mean Runtime with 95% Confidence Interval for Each Step



```
def step0_code(A, B):
    ans = 0
    for i in range(A, B+1):
        i = str(i)
        if i[0] == i[4]:
            if i[1] == i[3]:
                ans += 1
            else:
                ans += 0
        else:
            ans += 0
    return ans
```

```
def step1_code(A, B):
    ans = 0
    for i in range(A, B+1):
        s = str(i)
        if s[0] == s[4] and
           s[1] == s[3]:
            ans += 1
    return ans
```

```
def step2_code(A, B):
    ans = 0
    for i in range(A, B+1):
        if i // 10000 == i % 10
        and (i // 1000) % 10 == (i //
        10) % 10:
            ans += 1
    return ans
```

Example: Constrained Generation

- Create a story from a set of words: `['chisel', 'kite', 'shift', 'tire', 'leap', 'garage', 'fire', 'white', 'cardboard', 'spin', 'tag', 'ocean', 'short', 'surf', 'tomato', 'wash', 'feed', 'wave', 'blanket', 'routine', 'exercise', 'bed', 'talk', 'shoot', 'arm', 'roof', 'family', 'read', 'food', 'scrub']`
- At each step, get feedback on whether the concepts are covered, and whether the story makes sense
- Improve with the feedback



After Step 1 (63.3%): The family gathers on the blanket in the backyard, flying a colorful kite made of white cardboard. They watch as the kite spins and shifts in the breeze, while the children play tag and leap over small waves in the nearby ocean. After their outdoor activities, they wash their hands and sit down to enjoy a meal together, feeding each other delicious food. Later, they talk about their daily routine and exercise plans. Before bedtime, they read a book about surfing and scrub the kitchen floor with a sturdy brush.



After Step 4 (73.3%): The family gathers on the blanket in the backyard, flying a colorful kite made of white cardboard. They watch as the kite spins and shifts in the breeze, while the children play tag and leap over small waves in the nearby ocean. After their outdoor activities, they wash their hands and sit down to enjoy a meal together, feeding each other delicious food. Later, they talk about their daily routine and exercise plans. Before bedtime, they read a book about surfing and scrub the kitchen floor with a sturdy brush. **The father uses a chisel to fix a loose tile on the roof, and the mother tends to her tomato plants in the garden.**

When does it *not* work?

- Math Reasoning (GSM-8k): Generate Solution

```
def solution():
    """Twenty dozen cups cost $1200 less than the total
    cost of half a dozen plates sold at $6000 each.
    Calculate the total cost of buying each cup."""
    plates = 6
    plate_cost = 6000
    cups = 12 * 20
    cup_cost = plate_cost
    result = cup_cost
    return result
```

Initialize

(Gao*, Madaan*, Zhou* ICML 2023)

```
# Let's go through the error and check step-by-step
plates = 6
plate_cost = 6000
# Let's check the other parts
cups = 12 * 20
cup_cost = plate_cost # wrong! The cost of a cup is
not the same as the cost of a plate.
# The cost of a cup is $1200 less than the total cost
of half a dozen plates sold at $6000 each.
```

Feedback Generation

When does it ***not*** work?

- Math Reasoning (GSM-8k) Fix with feedback

```
# Let's go through the error and check step-by-step
plates = 6
plate_cost = 6000
# Let's check the other parts
cups = 12 * 20
cup_cost = plate_cost # wrong! The cost of a cup is
not the same as the cost of a plate.
# The cost of a cup is $1200 less than the total cost
of half a dozen plates sold at $6000 each.
```

Feedback Generation

```
def solution():
```

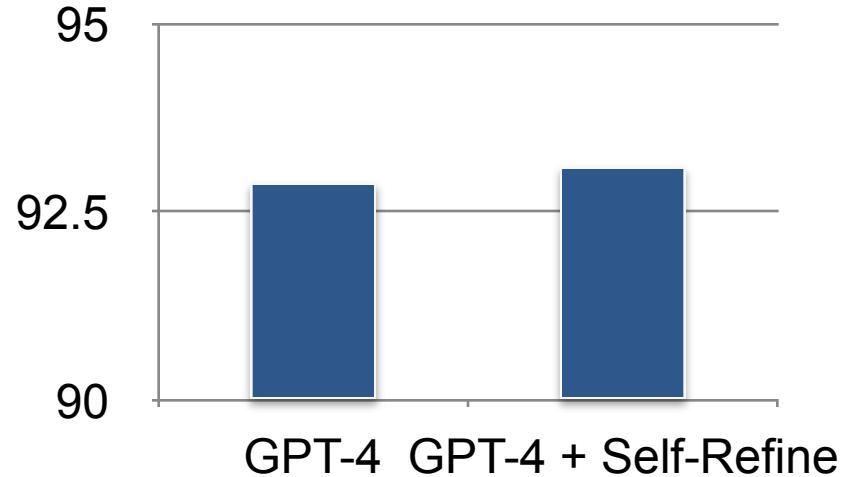
```
    """Twenty dozen cups cost $1200 less than the total cost
    of half a dozen plates sold at $6000 each.
    Calculate the total cost of buying each cup."""
    plates = 6
    plate_cost = 6000
    cups = 12 * 20
```

```
    total_cup_cost = (plate_cost * plates) - 1200
    result = total_cup_cost / cups
    return result
```

Fixed Solution

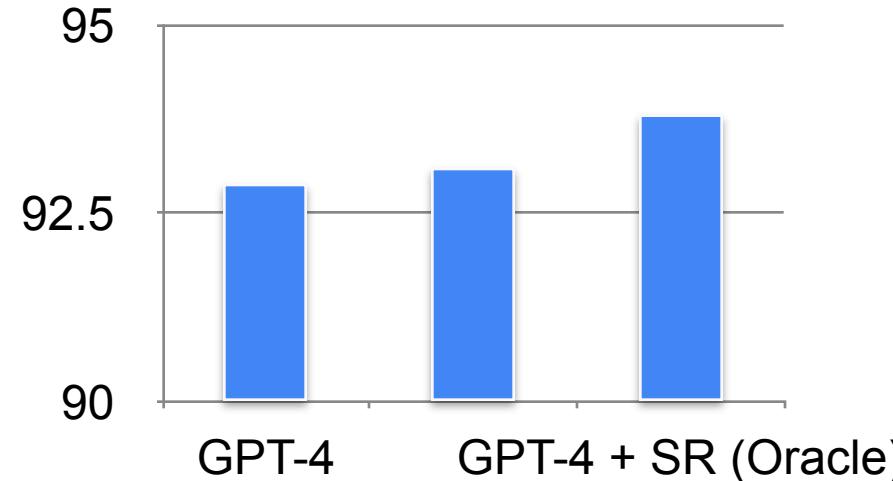
When does it *not* work?

- Almost no gains for math reasoning
- Confirmation bias: the model often says “No problems”
 - Almost always for GPT-4

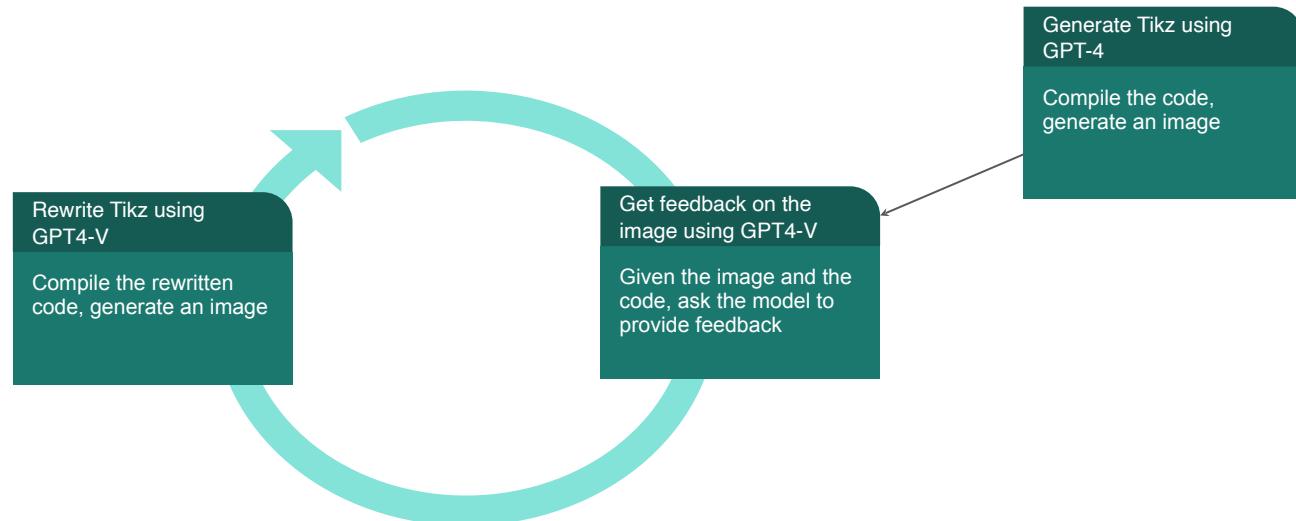


Why does it *not work for math reasoning?*

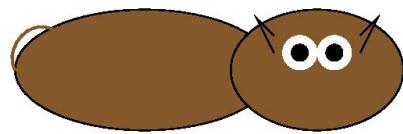
- Key problem:
 - It's hard to spot mistakes!
- Shows improvements *iff* Oracle feedback is available
 - Unrealistic, but helps in identifying bottlenecks



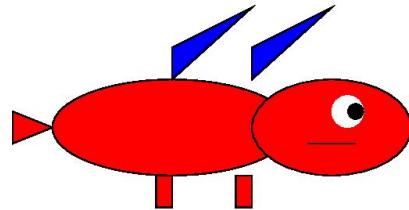
Visual Self-Refine



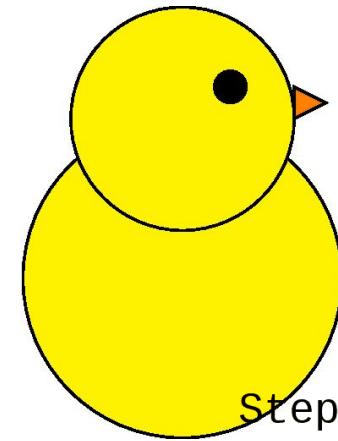
Visual Self-Refine



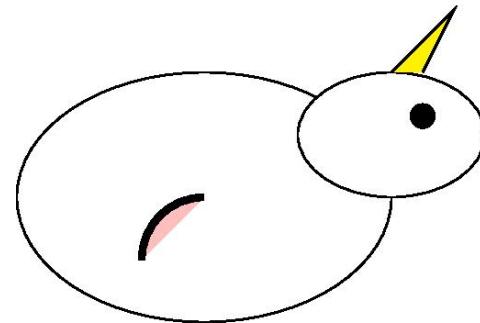
Step 0



Step 0

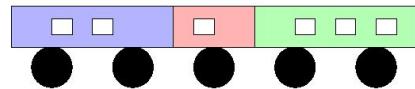


Step 0

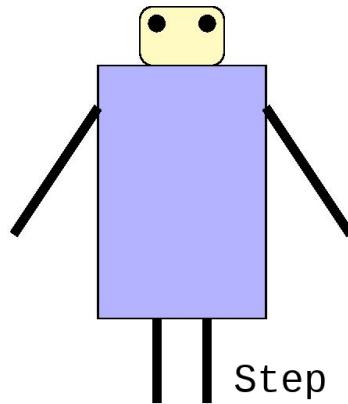


Step 0

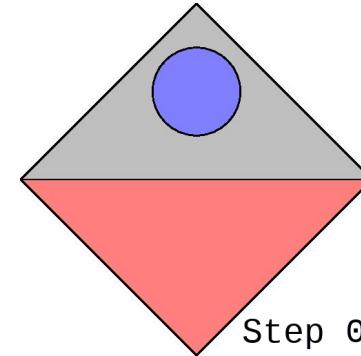
Visual Self-Refine



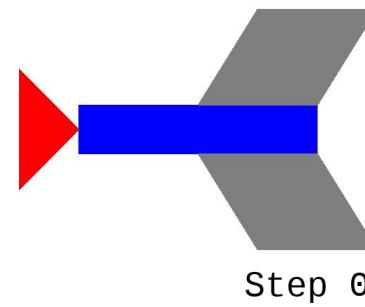
Step 0



Step 0

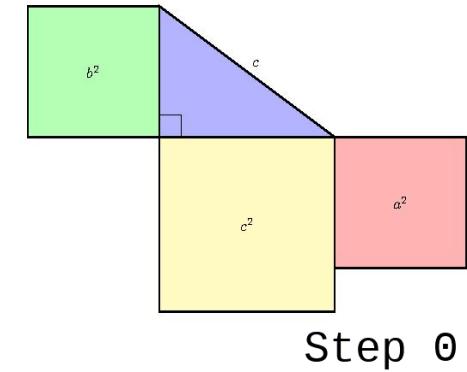
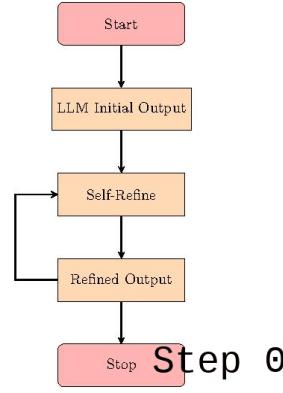
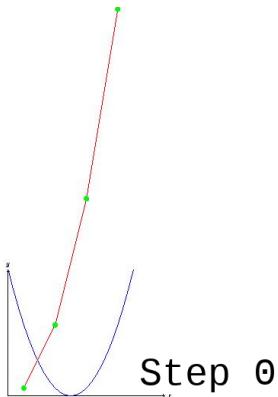
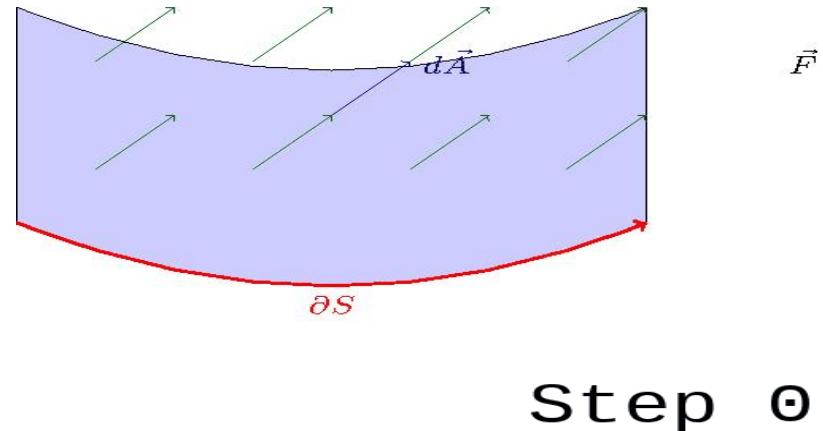
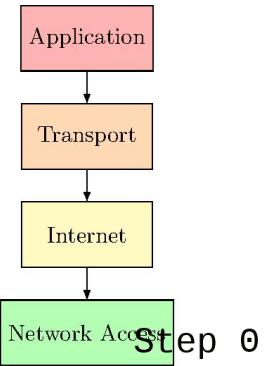


Step 0



Step 0

Visual Self-Refine



Why does it work? Surface-level Gradient-descent Analogy

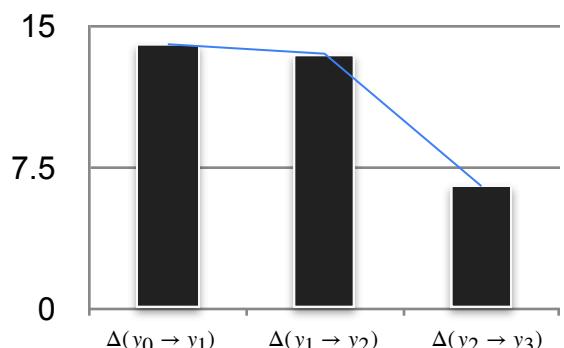
- Analogy: Surface level gradient-descent
 - Each step is sampling from a different distribution
 - $P(y_0 | x) \rightarrow p(y_1 | x, fb_0, y_0) \rightarrow p(y_2 | x, fb_0, y_0, fb_1, y_1)$



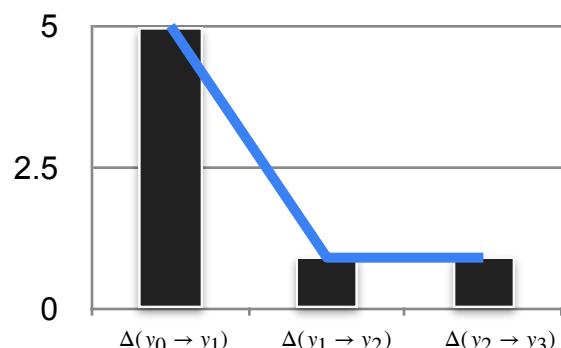
Why does it work? Surface-level Gradient-descent Analogy

- Diminishing gains

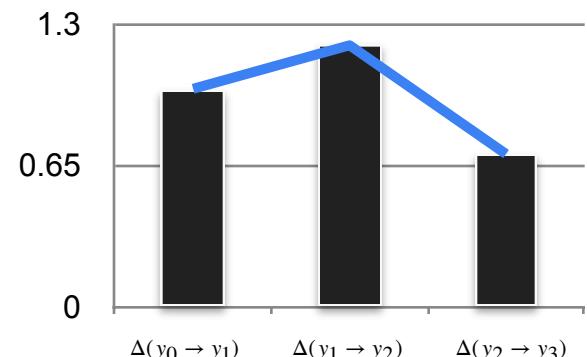
Constrained Generation



Code Optimization



Sentiment Reversal



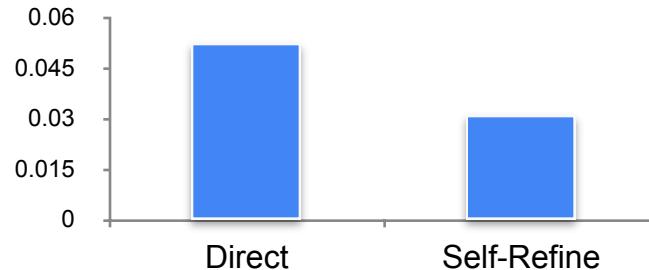
Why does it work? Most likely != The best

- Most likely != The best
 - Title: Third party firmware for the Linksys WRT routers and compatible models from other vendors.
 - Direct Acronym: TPFW ($p = 0.023$)
 - Self-Refine Acronym: FirmWave ($p = 0.018$)
- Calculate the probability of Acronyms where Self-Refine Acronyms were preferred with **LLAMA2-70B**

```
probability(f"You are a helpful assistant that generates acronyms.  
Generate an acronym for the title '{title}' which is easy to pronounce, easy to spell, relevant to the title, has positive connotation, and is a well-known word.  
  
The acronym for '{title}' is: {acronym}")
```

- Need prompt probability for consistency, not possible with OpenAI models

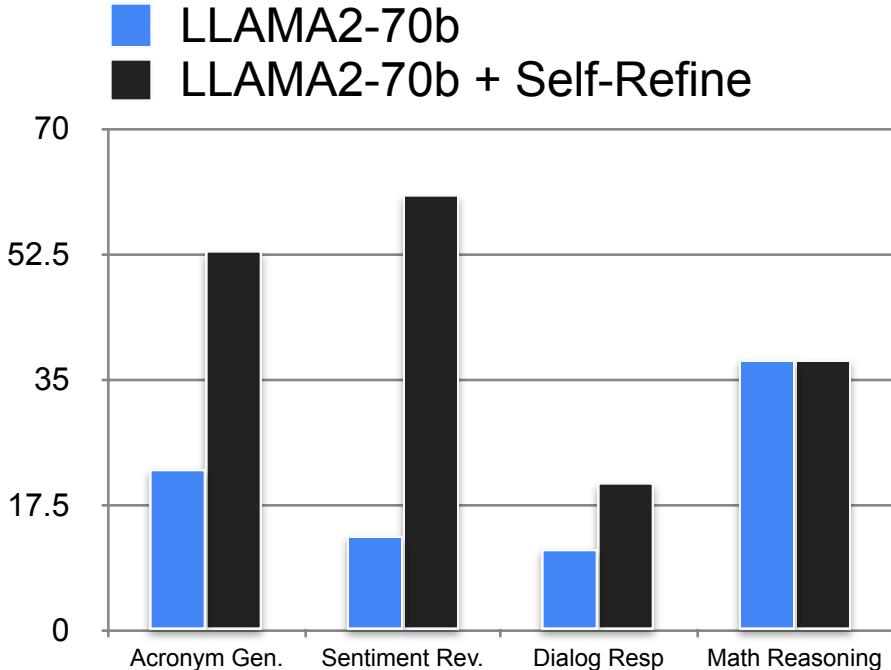
Preferred acronyms are not the most likely!



Does it work with Open-Source
Models?

Does it work without any
prompts?

Results with LLAMA-2 70B Instruct Only



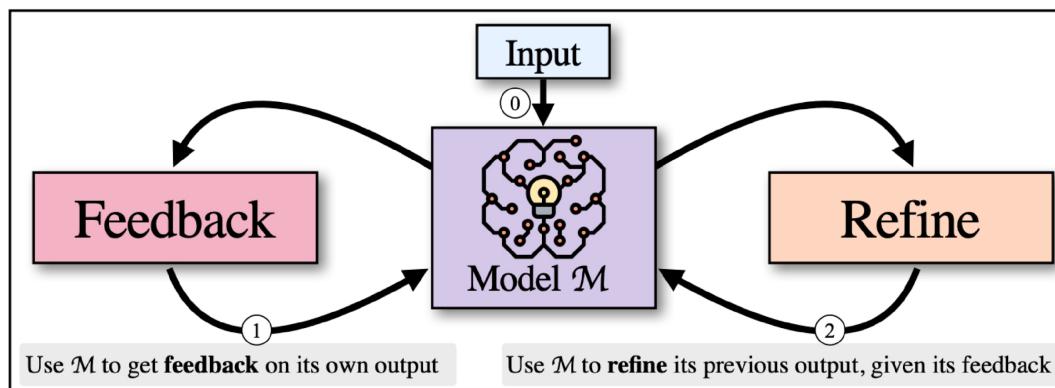
```
# Init: Transform a negative sentiment review into a positive one
start_chat_log = [
    {"role": "system", "content": "You are an AI language model that transforms a
    ↵ negative sentiment review into a positive one."},
    {"role": "user", "content": f"Please transform the following negative review
    ↵ into a positive one: \"{review}\". Respond in the format of: \"The positive
    ↵ review is: [Your Response Here]\"."},
]

# Generate Feedback: Provide feedback on the transformed review
start_chat_log = [
    {"role": "system", "content": "You are an AI model providing feedback on a
    ↵ sentiment transformed review."},
    {"role": "user", "content": f"Why is this review not Very positive? Point out
    ↵ specific issues and give concrete suggestions. Respond in the format of:
    ↵ 'Feedback: [Specific issues and suggestions]'"},
]

# Refine: Improve the review based on provided feedback
start_chat_log = [
    {"role": "system", "content": "You are an AI model that improves upon an
    ↵ existing review based on provided feedback."},
    {"role": "user", "content": f"Please improve the sentiment of the following
    ↵ review \"{sentence}\" based on this feedback: \"{feedback}\", and make it more
    ↵ positive. Always respond in the format of: \"The more positive review is:
    ↵ [Your Response Here]\"."},
]
```

Key Idea

- Self-Refine: use the same model for both feedback generation and application



AutoMix

AutoMatically Mixing Language Models



Collaborators: Pranjal Aggarwal, Mausam, Ankit Anand, Srividya Pranavi Potharaju, Swaroop Mishra, Pei Zhou, Dheeraj Rajagopal, Aditya Gupta, Shyam Upadhyay, Manaal Faruqui

LLM Development Landscape

~30x Price difference + latency and availability, load balancing

With broad general knowledge and domain expertise, GPT-4 can follow complex instructions in natural language and solve difficult problems with accuracy.

[Learn about GPT-4](#)

Model	Input	Output
8K context	\$0.03 / 1K tokens	\$0.06 / 1K tokens
32K context	\$0.06 / 1K tokens	\$0.12 / 1K tokens

GPT-3.5 Turbo is optimized for dialogue.

[Learn about GPT-3.5 Turbo ↗](#)

Model	Input	Output
4K context	\$0.0015 / 1K tokens	\$0.002 / 1K tokens
16K context	\$0.003 / 1K tokens	\$0.004 / 1K tokens

Inference pricing

Chat, language, and code models

Model size	Price per 1K tokens
Up to 3B	\$0.0001
3.1B - 7B	\$0.0002
7.1B - 20B	\$0.0004
20.1B - 40B	\$0.001
40.1B - 70B	\$0.003

<https://openai.com/pricing>

<https://together.ai/pricing>

Law of the Instrument

- Typical Setup: Use the cheapest model that “works”

Q: How many years in a century?

Q: If a tree produces fruit once every 10 years and only 1 in 3 of its fruit seeds result in a new fruit-bearing tree, how many new trees could potentially arise from a single tree's fruits over a century if all the seeds were planted?



Nate Raw
 @_nateraw

...

everything is an API call. free your mind.



```
import re

str_to_format = "The quick brown FOX"

re.sub(' +|_ ', '-', str_to_format).lower()

str_to_format = "The quick brown FOX"

messages = [
    {
        "role": "user",
        "content": f'{str_to_format}'
    }
]

I have a string above. Please replace all whitespace or "-" with "\n". It also needs to be lowercase. Please take in the string, then respond \n with ONLY the updated string, no other explanation, etc.

Input: {str_to_format}
"""

response = openai.ChatCompletion.create(
    model="gpt-3.5-turbo-0613",
    messages=messages,
)
print(response.choices[0].message.content)
```

10:02 AM · Aug 21, 2023 · 4,373 Views

Current Setup

If a tree produces fruit once every 10 years and only 1 in 3 of its fruit seeds result in a new fruit-bearing tree, how many new trees could potentially arise from a single tree's fruits over a century if all the seeds were planted?



Small Language Model
LLAMA2-13b



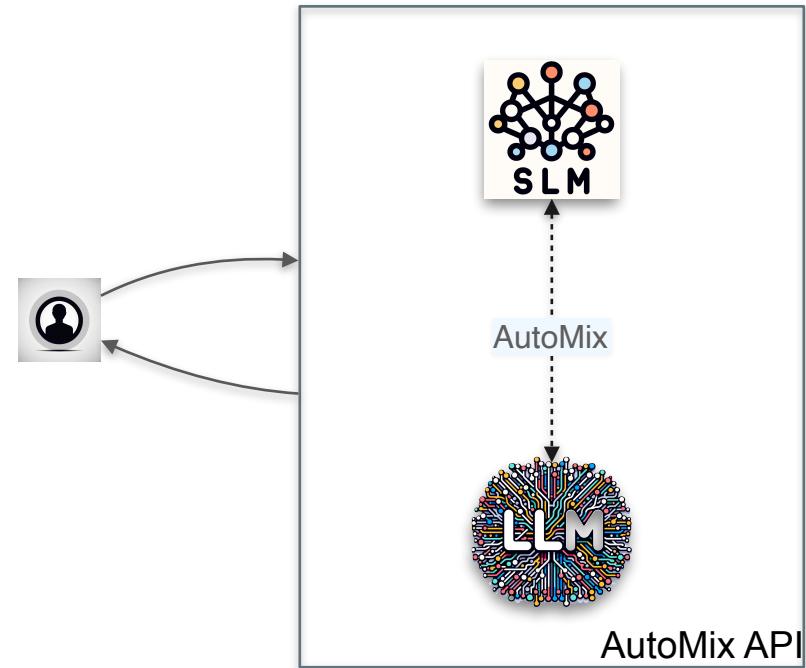
Large Language Model
LLAMA2-70b

How many years in a century?

Ideally

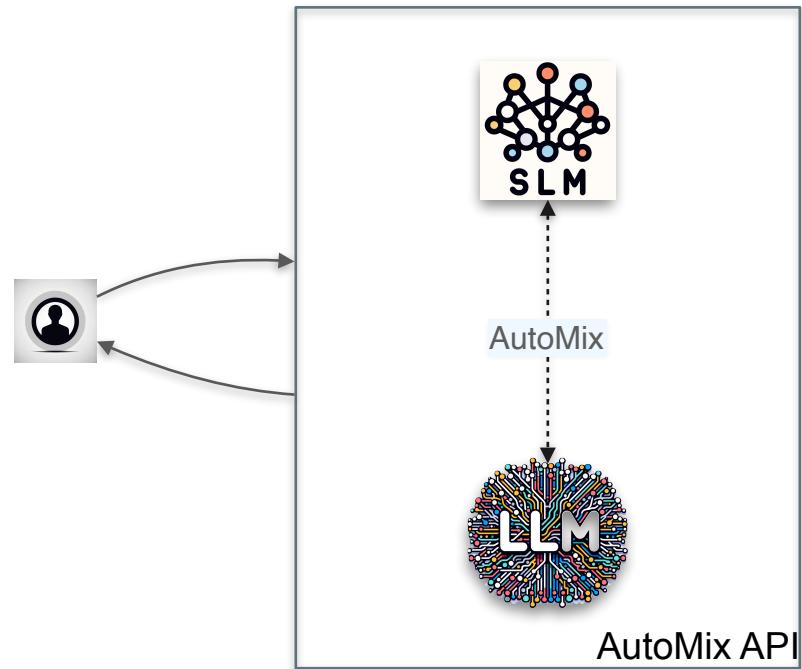
How many years in a century?

If a tree produces fruit once every 10 years and only 1 in 3 of its fruit seeds result in a new fruit-bearing tree, how many new trees could potentially arise from a single tree's fruits over a century if all the seeds were planted?



AutoMix

- Goal: Provide a generic, single interface over multiple models to route queries, without white box access to models, to deliver better performance per unit cost over mixing models randomly
- Idea Sketch:
 - Generate an initial answer with the SLM
 - Self-Verify the Answer
 - Route to LLM if needed



Existing Solutions

- Existing Work
 - [FlowGen: Fast and Slow Graph Generation](#) (Madaan et al. Dynn @ ICML July 2022):
 - Requires access to fine-grained perplexities
 - [FrugalGPT](#) (May 2023):
 - Need to do fine-tuning
 - [MultiPlexing](#) (Jun 2023)
 - Caching
 - [Cost-Effective Choice](#) (August 2023):
 - Set of queries are known
- Current LLM Setups
 - **API-based**
 - Minimal/no-access to logits
 - **Niche tasks** – large amounts of finetuning data may not be available
- **Goal:**
 - Mixing models without re-training
 - Provide cost-performance tradeoff

Idea Sketch

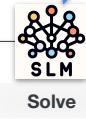
- Step 1: Generate an initial answer with the SLM
- Step 2: Self-Verify the Answer
- Step 3: Route to LLM if needed

Setup: Context-Grounded Reasoning

Article: Which country grows the most tea? The answer is India. It grows three times as much as China. Which country drinks the most tea? It's neither China nor Japan. It's Great Britain. [...] In general, there are two kinds of tea. Black tea and green tea. [...] One customer put the bag into a pot. Then he just poured hot water over it. And the tea bag was born. Shen Nong was the first to drink tea. (Shen was a Chinese emperor.) This was about 2737 B.C. Shen had bad digestion. So he drank several cups of hot water daily [...] Tea became the drink of China.

<article...> Q: According to the article, when was tea discovered?

Small Language Model



A: 1890 AD

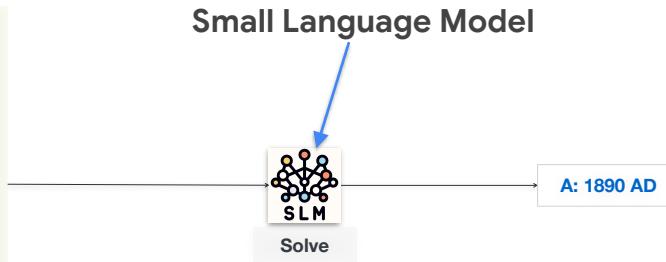
Solve

- Answer questions based on my emails, based on a website, etc.

Setup: Context-Grounded Reasoning

Article: Which country grows the most tea? The answer is India. It grows three times as much as China. Which country drinks the most tea? It's neither China nor Japan. It's Great Britain. [...] In general, there are two kinds of tea. Black tea and green tea. [...] One customer put the bag into a pot. Then he just poured hot water over it. And the tea bag was born. Shen Nong was the first to drink tea. (Shen was a Chinese emperor.) This was about 2737 B.C. Shen had bad digestion. So he drank several cups of hot water daily [...] Tea became the drink of China.

<article...> Q: According to the article, when was tea discovered?



NarrativeQA (Kočiský et al. 2018)	Question answering about entire books and movie scripts.
Qasper (Dasigi et al. 2021)	Question answering over research papers.
QuALITY (Pang et al. 2021)	Multiple-choice questions over long articles and stories.
Contract NLI (Koreeda and Manning 2021)	Natural language inference over non-disclosure agreements.
COQA (Reddy et al. 2019)	CoQA: A Conversational Question Answering Challenge.

Step 1: Generate an initial answer

```
"""Story:  
{context}
```

You are given a story, which can be either a novel or a movie script, and a question. Answer the question as concisely as you can, using a single phrase if possible.

Question and Possible Answers:
{question}

Answer:

```
"""
```

```
"""Article:  
{context}
```

You are given a scientific article and a question. Answer the " question as concisely as you can, using a single phrase or" " sentence if possible. If the question cannot be answered based" " on the information in the article, write 'unanswerable'. If the" " question is a yes/no question, answer 'yes', 'no', or" " 'unanswerable'."

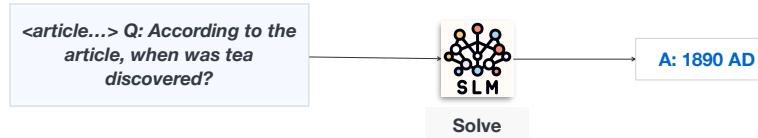
Question:
{question}

Answer:

```
"""
```

AutoMix: Idea Sketch

- Step 1: Generate an initial answer with the SLM

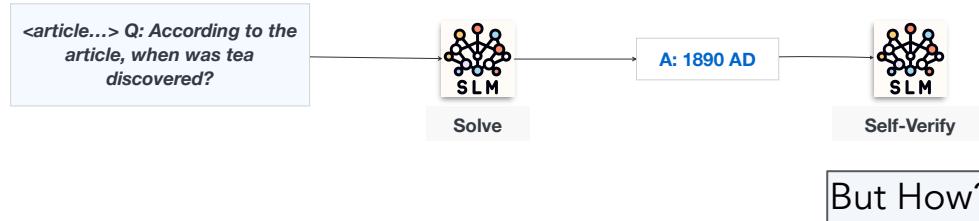


- Step 2: Self-Verify the Answer

- Step 3: Route to LLM if needed

Verification as Entailment

- Given an input x , context \mathbf{C} , we have some output \mathbf{y}' generated by the SLM.
 - How do we check if \mathbf{y}' is good?
- Intuition
 - If \mathbf{y}' is correct, the context may have some evidence to infer \mathbf{y}'
- Solution
 - Frame self-verification as an entailment problem



Aside: Entailment

- Entailment determines if a given statement (hypothesis) logically follows from another statement (premise).

Premise: All triangles have 3 sides. A is a triangle.

Hypothesis: A has three sides.

Entailed

Premise: I have a cat.

Hypothesis: I have an elephant.

Not Entailed

Few-Shot Verification as Entailment

```
Prompt = """Context:  
{context}
```

```
Question: {question}
```

```
AI Generated Answer: {generated_answer}
```

Your task is to evaluate if the AI Generated Answer is correct, based on the provided context and question.

Provide the judgement and reasoning for each case.

Task Form Format:

Evaluation: Choose between <Correct, Incorrect, Unsure>

```
### Task Form (Fill the portion below)
```

Evaluation:"""

Few-Shot Verification as Entailment

```
Prompt = """Context:  
{context}  
---  
Question: {question}
```

```
AI Generated Answer: {generated_answer}
```

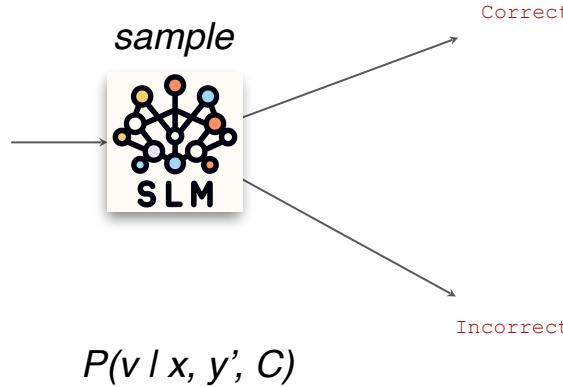
Your task is to evaluate if the AI Generated Answer is correct, based on the provided context and question. Provide the judgement and reasoning for each case.

```
### Task Form Format:
```

```
Evaluation: Choose between <Correct, Incorrect>
```

```
### Task Form (Fill the portion below)
```

```
Evaluation:""".format(...)
```



Step 2: Few-Shot Verification as Entailment

```
Prompt = """Context:  
{context}  
---  
Question: {question}  
  
AI Generated Answer: {generated_answer}"""
```

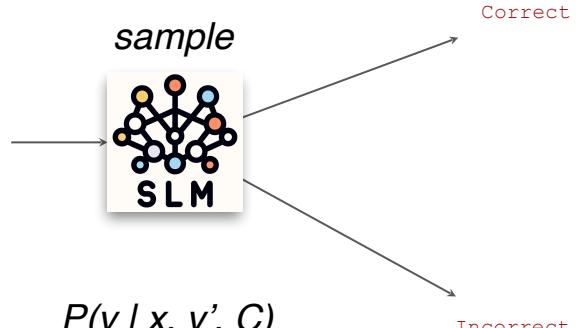
Your task is to evaluate if the AI Generated Answer is correct, based on the provided context and question. Provide the judgement and reasoning for each case.

Task Form Format:

Evaluation: Choose between <Correct, Incorrect>

Task Form (Fill the portion below)

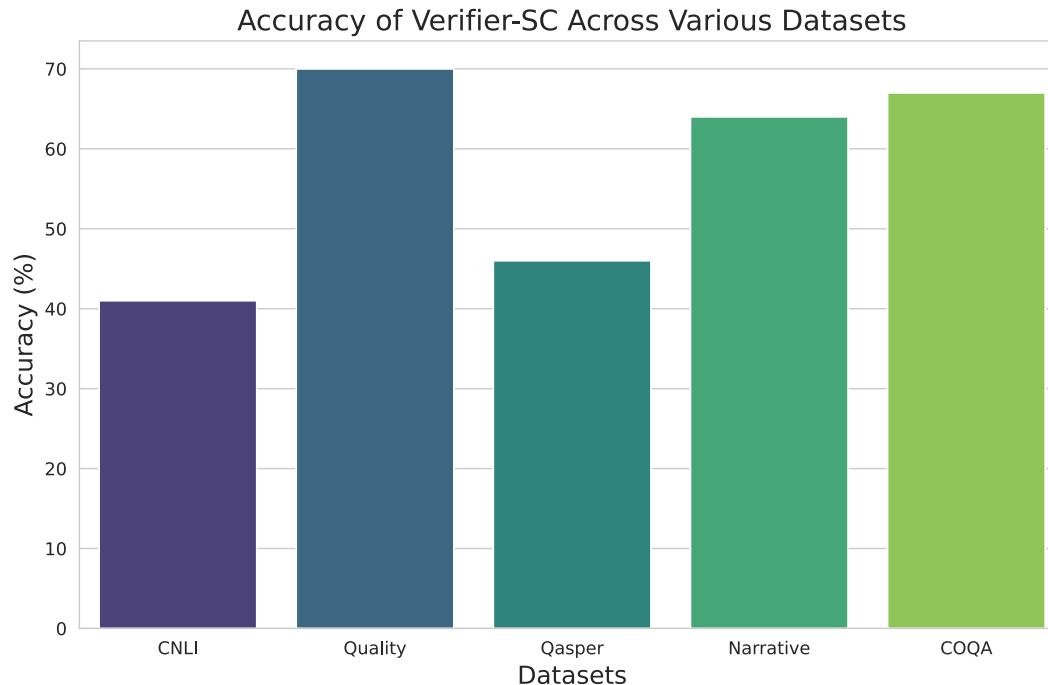
Evaluation:""".format(...)



*But we don't
have access
to logits :(*

*Draw 8
samples –
KV cache :)*

Verifier Results



Examples of Verification

Context: Which country grows the most tea? The answer is India. It grows three times as much as China. Which country drinks the most tea? It's neither China nor Japan. It's Great Britain. [...] In general, there are two kinds of tea. Black tea and green tea. [...] One customer put the bag into a pot. Then he just poured hot water over it. And the tea bag was born. Shen Nong was the first to drink tea. (Shen was a Chinese emperor.) This was about 2737 B.C. Shen had bad digestion. So he drank several cups of hot water daily [...] Tea became the drink of China.

Question: When did Shen Nong drink tea?

Generated Answer (by LLAMA2-13b): He took it in 1990.

Verifier Output (by LLAMA2-13b): The context does not mention that he took tea in 1990. The AI generated answer is Incorrect.

Context: ...

Question: Who was artist consider to be political cartoonist?

Generated Answer (by LLAMA2-13b): Honoré Daumier.

Verifier Output (by LLAMA2-13b): The context specifically mentions the artist Honoré Daumier as a political cartoonist. Verification Decision: The AI generated answer is Correct.

AutoMix: Idea Sketch

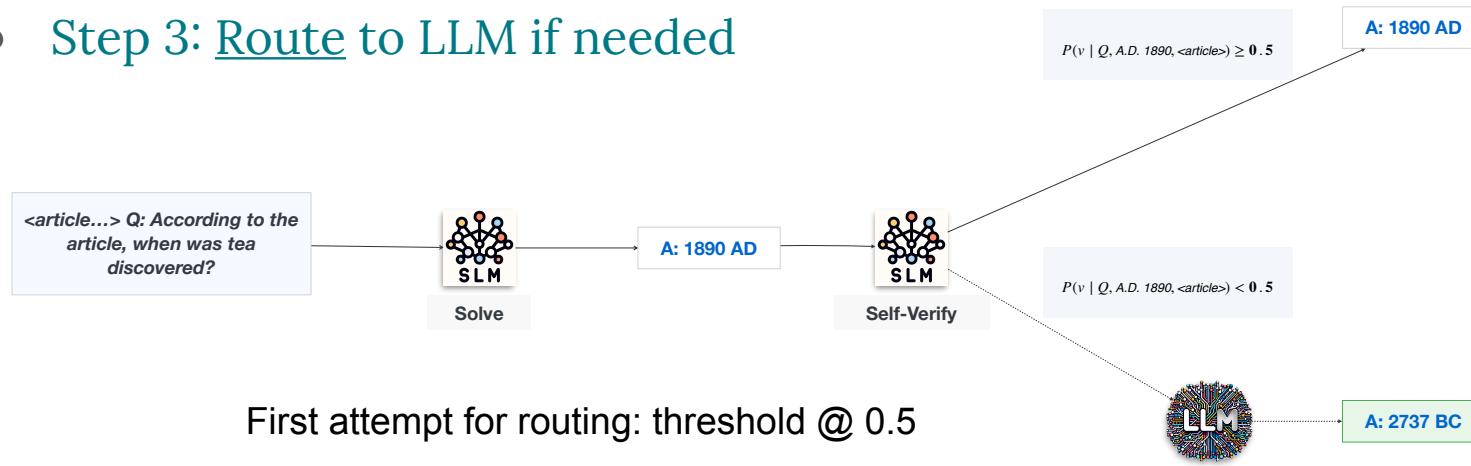
- Step 1: Generate an initial answer with the SLM
- Step 2: Self-Verify the Answer

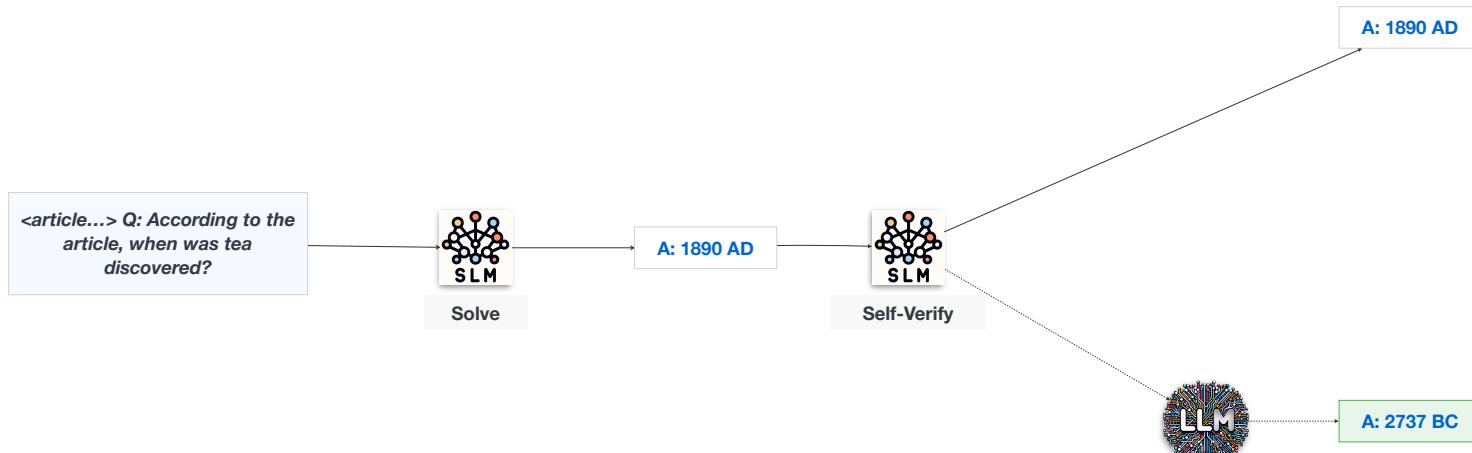


- Step 3: Route to LLM if needed

AutoMix: Idea Sketch

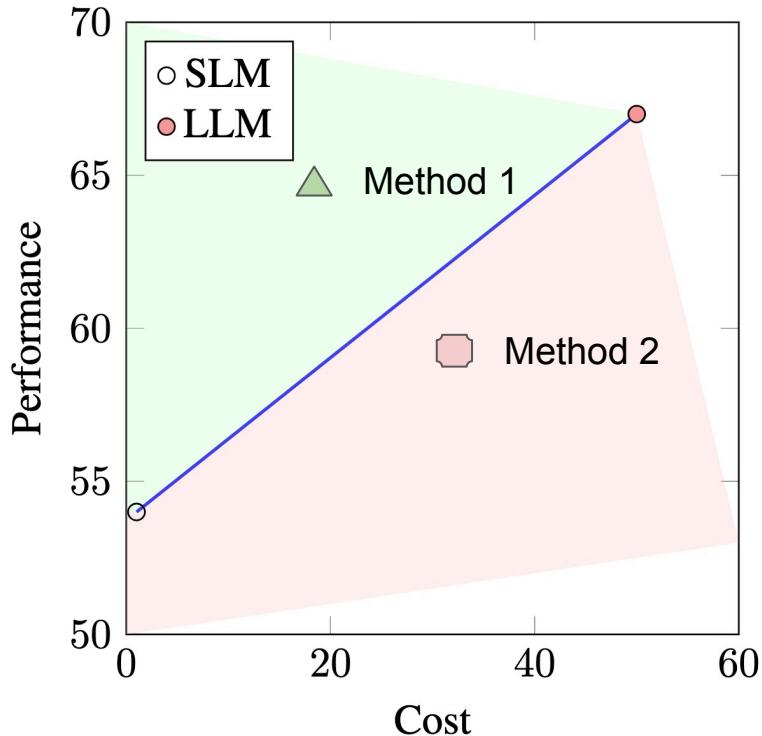
- Step 1: Generate an initial answer with the SLM
- Step 2: Self-Verify the Answer
- Step 3: Route to LLM if needed





- Every query has a cost, and we are calling SLM twice. How do we measure performance per unit cost?
- Are we doing better than just randomly mixing models?
- Incremental Benefit Per (unit) Cost

Incremental Benefit Per Cost: Intuition



Method 1 is better, as we are getting > performance at a cost expected with random mixing.

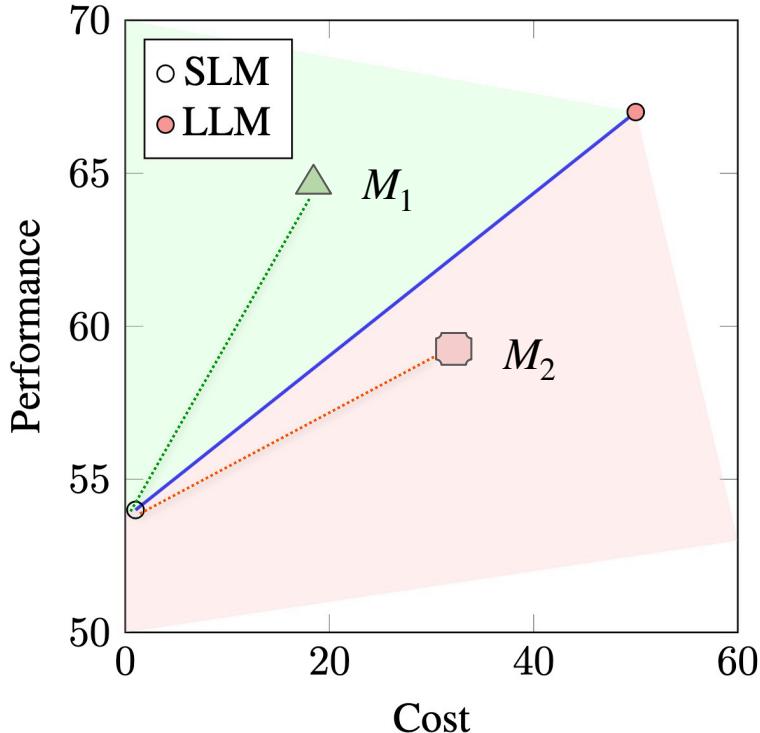
Incremental Benefit Per Cost

- Costs: Small Model = C_S , Large Model = C_L
- Performance: Small Model = P_S , Large Model = P_L
- How much do we gain by adding one unit cost in terms of performance?
- Incremental Benefit Per Unit Cost (IBC) == Slope
 - $IBC_{BASE} = \frac{P_L - P_S}{C_L - C_S}$

$$IBC(M_1) = \frac{P_{M_1} - P_S}{C_{M_1} - C_S}$$

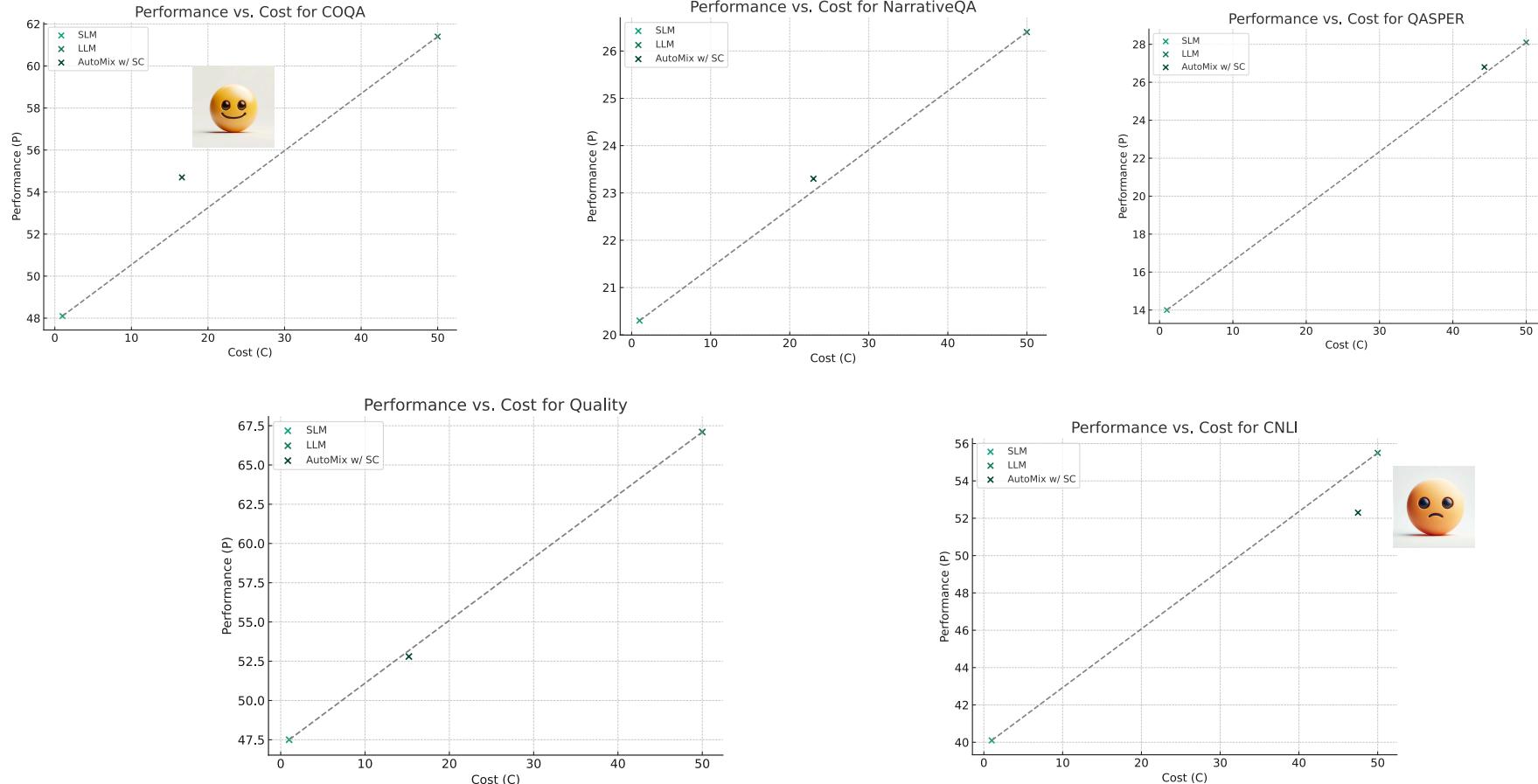
$$IBC(M_2) = \frac{P_{M_2} - P_S}{C_{M_2} - C_S}$$

$$\Delta_{IBC}(M) = \frac{IBC(M) - IBC_{BASE}}{ibc_{BASE}} \times 100$$

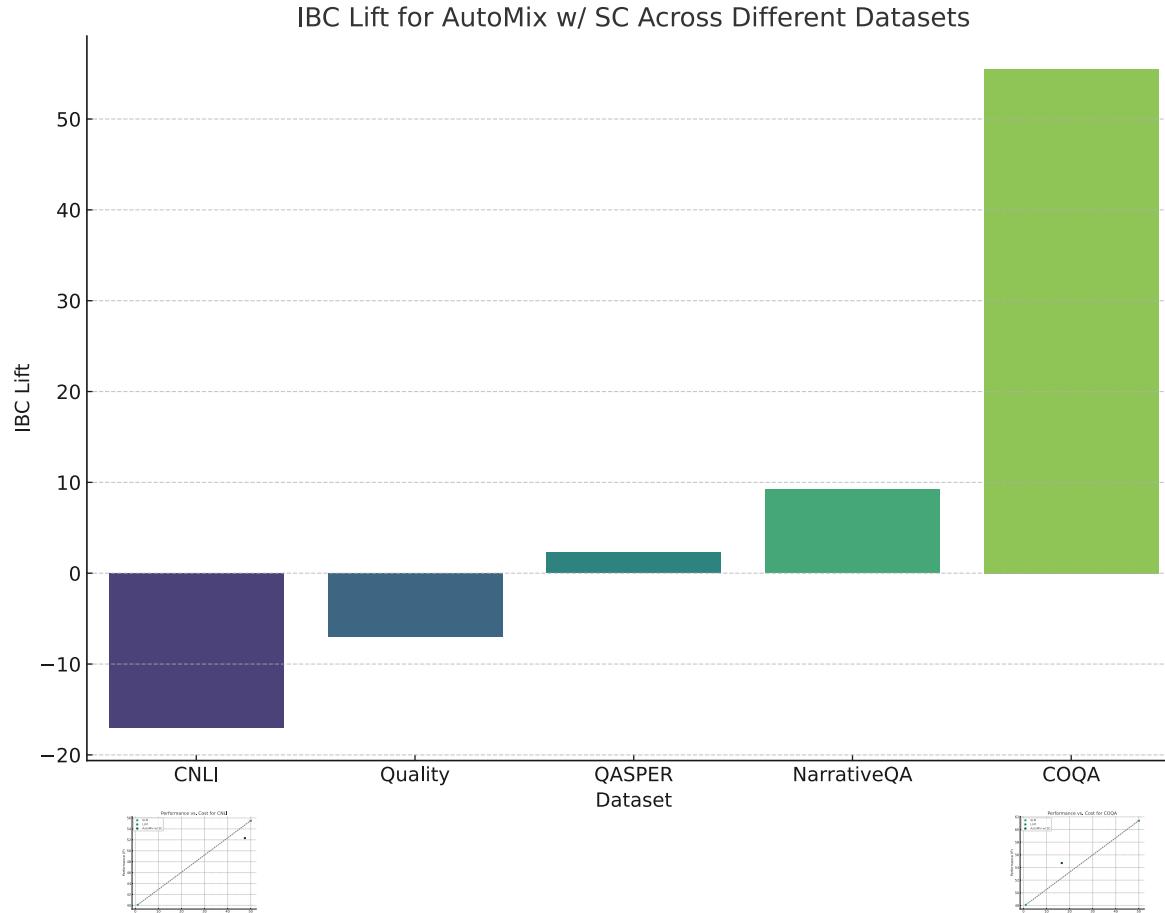


- Are we above or below the SLM/LLM cost performance curve and by how much?

Results

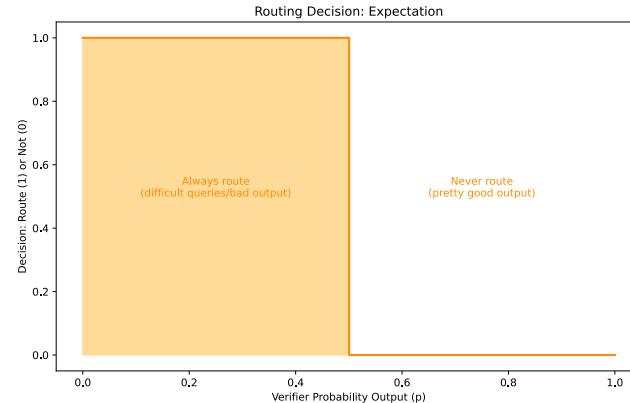


Results with Majority Voting

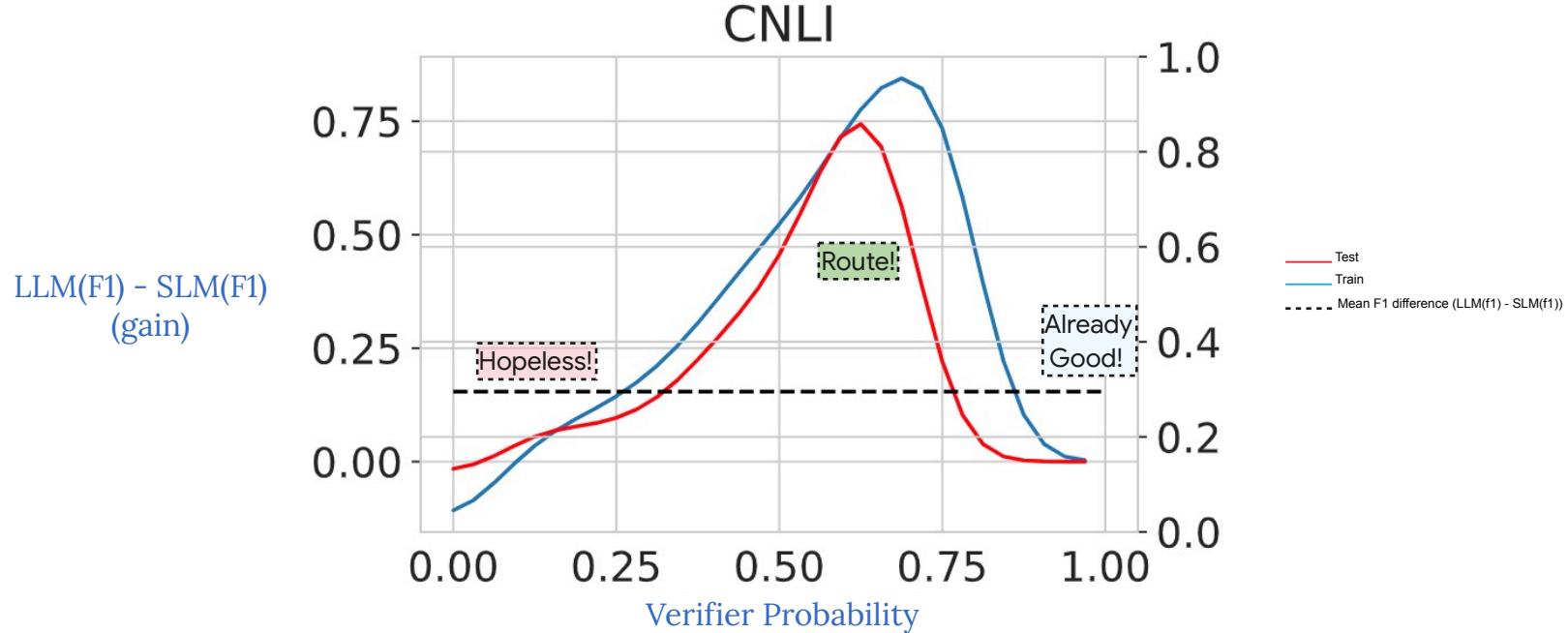


Routing Decision

- The verifier is not consistently good
- Prevailing wisdom: bimodal distribution of queries
 - Difficult: SLM output is suboptimal – route
 - Easy: SLM output is good – don't route
- **Misses a critical third mode: hopeless queries!**

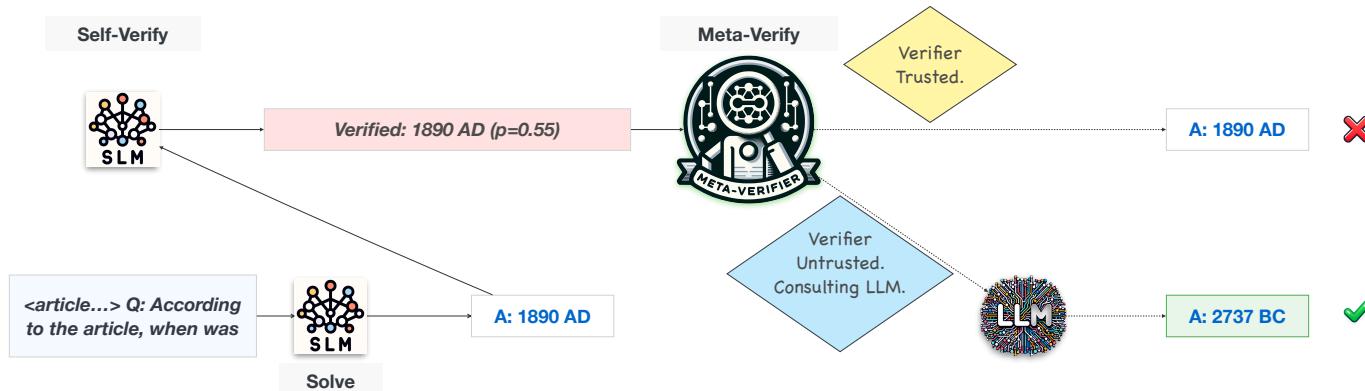


Routing Decision

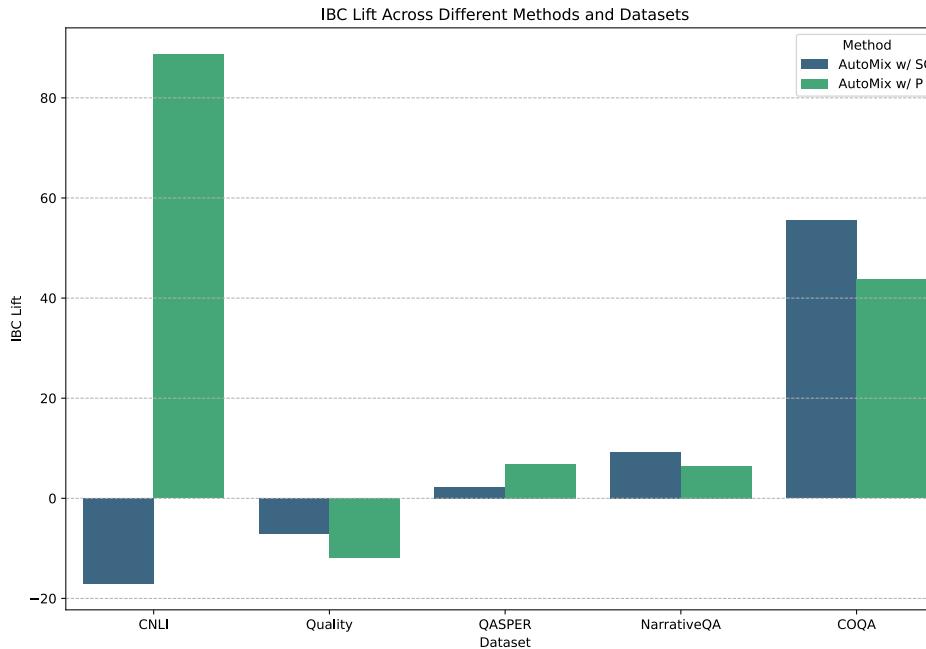


Improving Verification

- How can we improve verification?
- Need to verify the verifier – meta-verification!

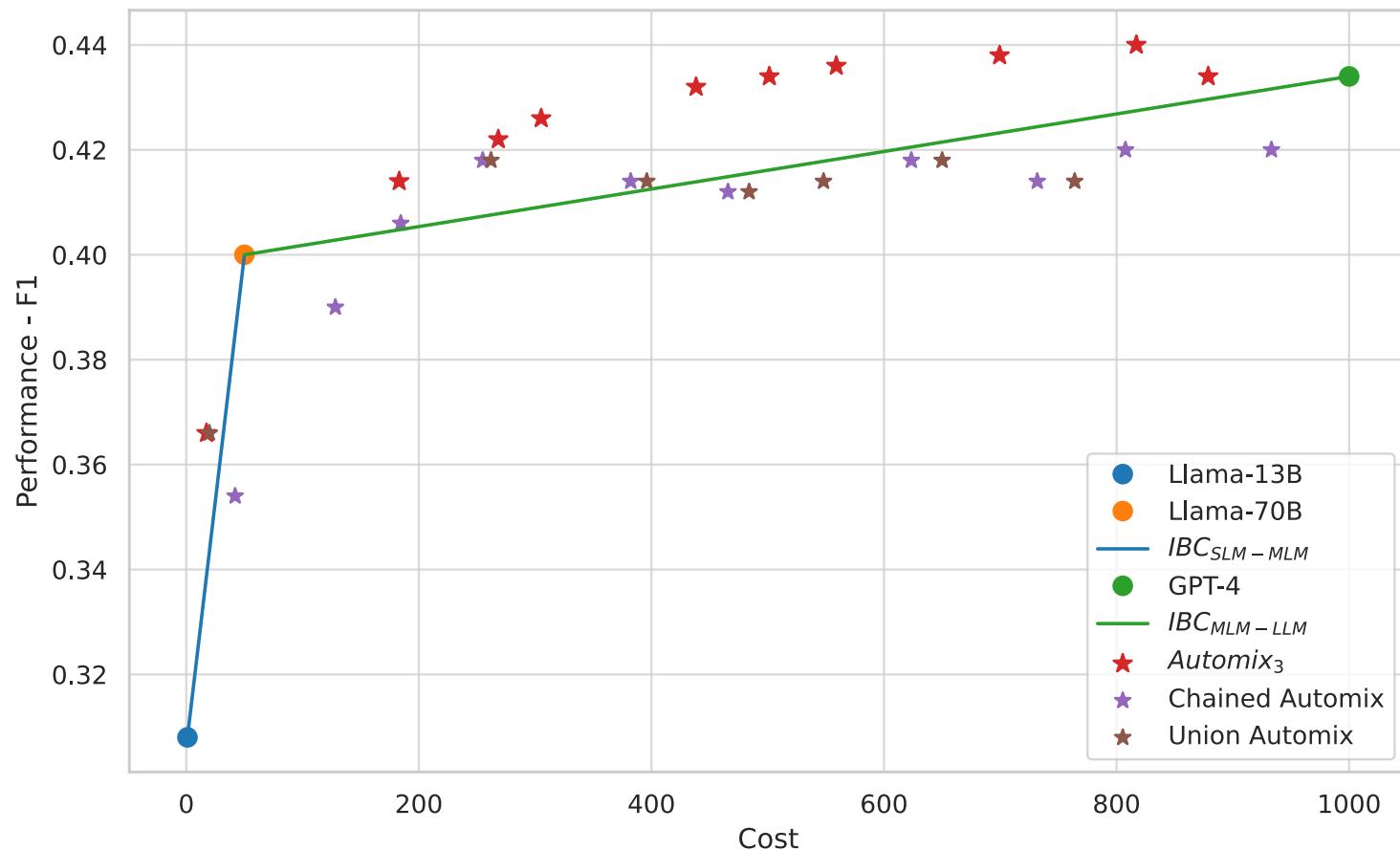


Results with POMDP



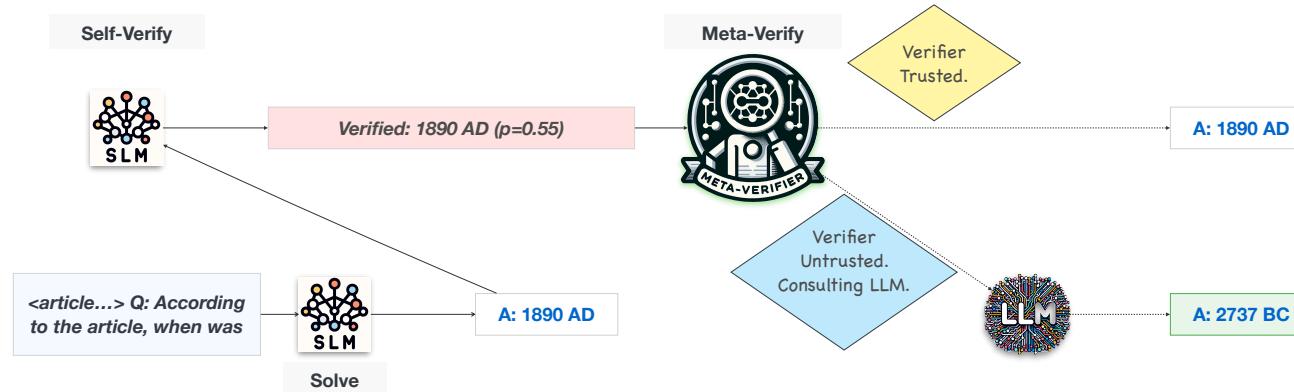
Helps with cases
where a simple
threshold won't cut it!

Extend to Three Models



AutoMix

- Context-Grounded Self-Verification, then route to a larger model
- Meta-verification to fix noisy verification
- Measure using incremental benefit per cost

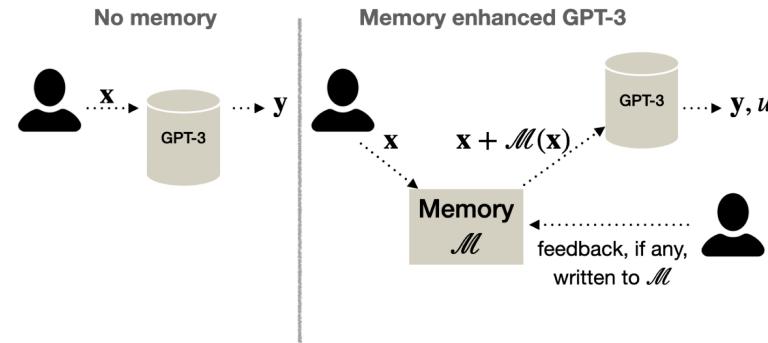


Self-Enhancing Text Generation and Reasoning

- Research Goal
 - Stateful Inference
 - Giving Machines a Deep Thinking Mode – trading inference time compute for effective reasoning
- Ingredients
 - Reacting to Feedback + Generating Feedback (Neurips 2023)
 - Self-Improvement
 - Memory to prevent repeated mistakes (EMNLP 2022, NAACL 2022)
 - Tool Augmentation (ICML 2023, EMNLP 2022)
 - Leveraging external input
 - Specialized computation



MemPrompt



Program-aided Language Models

- Olivia has \$23. She bought five bagels for \$3 each. How much money does she have left?

Olivia had 23 dollars. 5 bagels for 3 dollars each will be dollars. So she has dollars left.

```
def solution():
    money_initial = 23
    bagels = 5
    bagel_cost = 3
    money_spent = bagels * bagel_cost
    money_left = money_initial - money_spent
    result = money_left
    return result
```

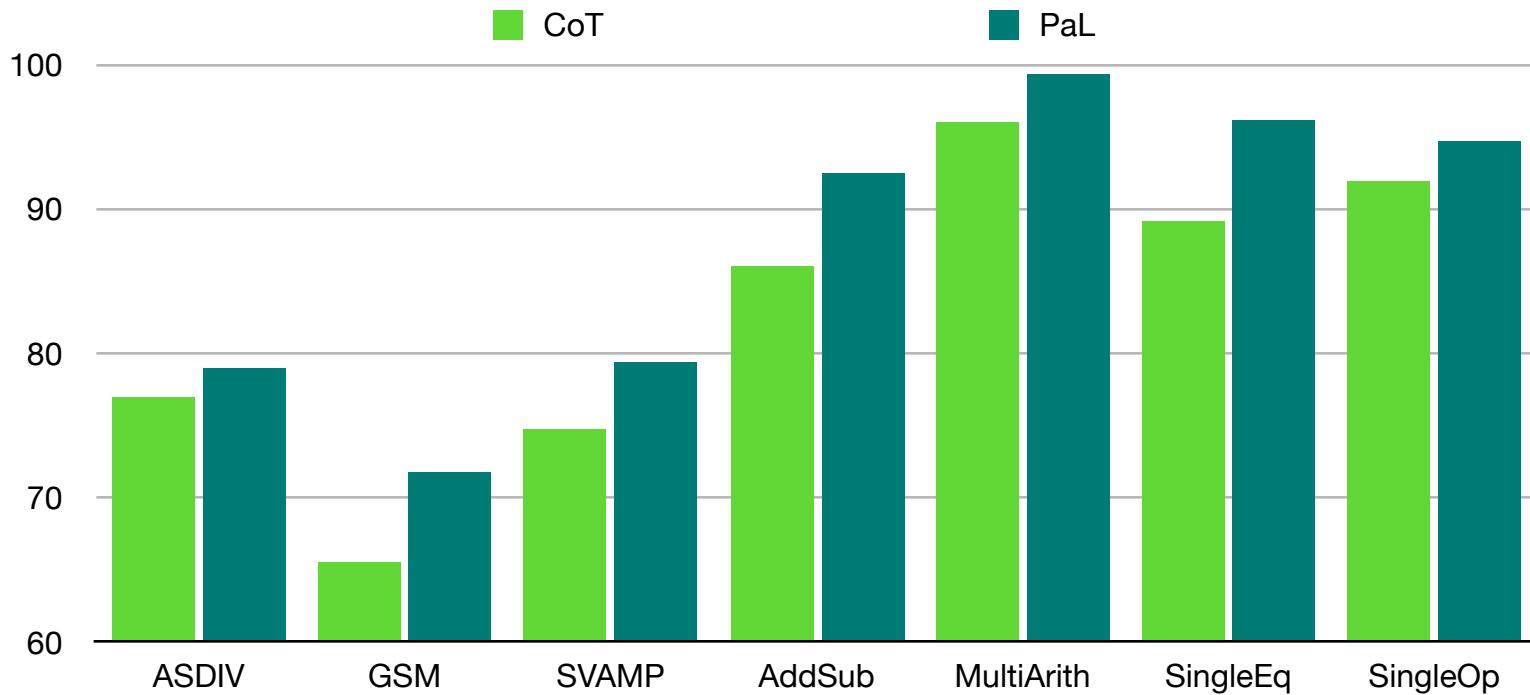
CoT

PaL

Comparison with CoT:

- The language model is responsible for generating a high-level plan that is executed to derive the answer
- The results are obtained after running the program

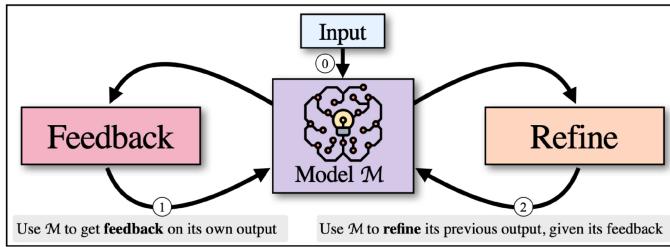
Program-aided Language Models



What's next? (Opinion)

- Resurgence of techniques for getting more out of a fixed language model during inference
 - Search, Planning, Self-Verification, Self-Refinement
 - GOFAI + LMs
- Current research focus:
 - LMs that can solve challenging code reasoning tasks in the real world
 - Hyper-personalized LMs
 - Learning from a few mistakes

Self-Enhancing Text Generation and Reasoning



Improving LLM outputs without human-in-the-loop



Automatically Mixing Language Models