Equipping GPT-4 with Numeric Calculation

GPT-4 is terrible at calculating with numbers. It makes mistakes all the time that lead to problematic user experiences on texts involving even the most basic numeric problems. We describe a simple, general technique to address this.

Author: Don Syme, Albert Ziegler and Johan Rosenkilder at GitHub Next.

Proof-of-concept implementation: GitHub internal. Microsoft clone. Access available on request.

We encourage collaboration. Please iterate with us and/or take this further. If publishing please include us as co-authors.

Evaluation.

The Problem

GPT-4 is terrible at calculating with numbers. It makes mistakes all the time that lead to problematic user experiences on texts involving even the most basic numeric problems.

Take for example this problem. To quote from the author:

> This is by far the worst mistake made during the demo. It's also the most unexpected.

GPT-4 is also terrible at comparative logic involving numbers. It quite happily writes sentences like this (emphasis added):

> Both companies reported an increase in net sales compared to the same quarter last year, but Gap Inc. had a much **larger** absolute **and relative** increase ($4.04 billion, **up 2%**) than lululemon ($1.9 billion, **up 28%**).

Ugh, 2 is not greater than 28. Half right but totally wrong.

In short, **GPT-4 can't handle numbers or comparisons of numbers, period**. In our opinion, *GPT-4 should not be trusted to write a number that is not present verbatim in the input, nor to reason about numbers in any significant*

way. In trust scenarios, don't allow GPT-4 to write numbers, and beware that every numeric comparison may be flawed.

The Hope

There is, however, hope, and it is in this simple logic:

1. GPT-4 is terrible at **numeric calculation** but good at **writing numeric calculation code**
2. Python and many other tools are perfect at **evaluating numeric calculation code**.

The answer is obvious: get GPT-4 to write the numeric calculation code relevant to the question, and get Python or some other tool to evaluate it, and use GPT-4 for the rest.

The Approach

Our aim is to "equip" or "augment" GPT-4 with a numeric calculator. The approach is simple:

Without numeric calculation equipping:

1. **GPT-4**: Question –> Answer

With numeric calculation equipping:

1. **GPT-4**: Question –> Numeric calculation code
2. **Evaluator**: Numeric calculation code –> Numeric calculation answers
3. **GPT-4**: Question + Numeric calculation code + Numeric calculation answers –> Answer

We call this "equipping" GPT-4 with numeric calculation. GPT-4 has a new tool in the box, and it turns out it loves to use it.

A sample equipping prompt

In Step 1 we prompt GPT-4 to produce relevant numeric calculation code. There are many ways to do this and further experimentation will be needed, but here's an example prompt addition, placed after the question:

```
# Guidance

Do not answer the question. Instead, your task is to write some numeric calculation and comp

After the question write a code block with up to three sections containing content relevant

In the "Definitions" section define a label for each number in the original question like ‘c
* Every label name should include the unit of measure if known.
* This section should be valid Python and can include valid Python single-dimensional arrays
```

```
* Do not use or create multi-dimensional arrays.
* Give each label a unit of measure in a comment after each definition.
* Document the meaning of each definition in the comment.
* If the unit of measure is unknown use "unknown".
* Omit this section if there are no numbers in the question.

In the "Calculations" section define additional relevant labels using Python or numpy formul
* Define each label using a formula, referencing previously defined labels.
* Avoid new assumptions in this section, if you make an assumption document it.
* Every label name should include the unit of measure if known.
* Do NOT include the calculated values for these labels.
* Give each label a unit of measure in a comment after each definition.
* Document the meaning of each definition in the comment.
* If the unit of measure is unknown use "unknown".
* This section should be valid Python using regular Python or numpy.
* Omit this section if there are no additional labels relevant to the answer.

In the "Comparisons" section define additional labels using Python or numpy formulae by comp
* Do NOT include the calculated true/false values for these labels.
* This section should be valid Python using regular Python or numpy.
* Document the meaning of each definition in the comment.
* Omit this section if there are no comparisons relevant to the answer.

## Relevant calculations and comparisons
```

> NOTE: in our manual testing, we generated Python calculation code. In our prototype, for convenience we generated Javascript calculation code. There are many choices here and it is not strictly necessary to generate a general-purpose programming language. See discussion in Appendix.

GPT-4 then writes the calculation code and stops. Step 2 evaluates this calculation code. Step 3 combines the original question with both the calculation code and answers and generates the answer to the question.

In Step 3 the calculation code is included as it contains relevant explanation text - this could be omitted if the explanation text is combined with the answers. Further, in Step 3 a prompt directive can be added saying "only use numbers that are exactly present in the question or calculation".

Example 1

Take this problem:

```
This is an extract from the Microsoft annual report for 2022:

Microsoft repurchased the following shares of common stock under the share repurchase progra
(In millions)   Shares      Amount      Shares      Amount      Shares      Amount
```

Year Ended June 30,		2022		2021		2020
First Quarter	21	$6,200	25	$5,270	29	$4,000
Second Quarter	20	6,233	27	5,750	32	4,600
Third Quarter	26	7,800	25	5,750	37	6,000
Fourth Quarter	28	7,800	24	6,200	28	5,088

What is the overall average and standard deviation of shares repurchased per quarter over th

The correct numeric answers are:

```
>>> avg
5890.916666666667
>>> std
1077.0771759978743
```

Old Approach

A sample plain textual response from GPT-4 is unsurprisingly garbage - GPT-4 can't do numbers.

> ... lots of working ...
>
> Therefore, the overall average and standard deviation of shares repurchased per quarter over these three years in dollars are 17,898.25 million dollars and 1,583.68 million dollars, respectively.

Nope GPT-4, wrong.

New Approach, Step 1

Now take a multi-phase approach. In step 1 we add the prompt from the previous section, and generate the calculation code:

```python
# Definitions
shares_2022 = [21, 20, 26, 28] # millions of shares
amount_2022 = [6200, 6233, 7800, 7800] # millions of dollars
shares_2021 = [25, 27, 25, 24] # millions of shares
amount_2021 = [5270, 5750, 5750, 6200] # millions of dollars
shares_2020 = [29, 32, 37, 28] # millions of shares
amount_2020 = [4000, 4600, 6000, 5088] # millions of dollars

# Calculations
total_shares = shares_2022 + shares_2021 + shares_2020 # millions of shares
total_amount = amount_2022 + amount_2021 + amount_2020 # millions of dollars
average_amount = np.mean(total_amount) # millions of dollars
std_amount = np.std(total_amount) # millions of dollars

# Comparisons
more_shares_2022 = shares_2022 > shares_2021 # boolean array
```

```
more_amount_2022 = amount_2022 > amount_2021 # boolean array
highest_amount = np.max(total_amount) # millions of dollars
lowest_amount = np.min(total_amount) # millions of dollars
highest_amount_quarter = total_amount.index(highest_amount) + 1 # quarter number
lowest_amount_quarter = total_amount.index(lowest_amount) + 1 # quarter number
```

New Approach, Step 2

We evaluate the calculations using Python by adding `import numpy as np`, evaluating and printing:

```
shares_2022 [21, 20, 26, 28]
amount_2022 [6200, 6233, 7800, 7800]
shares_2021 [25, 27, 25, 24]
amount_2021 [5270, 5750, 5750, 6200]
shares_2020 [29, 32, 37, 28]
amount_2020 [4000, 4600, 6000, 5088]
total_shares [21, 20, 26, 28, 25, 27, 25, 24, 29, 32, 37, 28]
total_amount [6200, 6233, 7800, 7800, 5270, 5750, 5750, 6200, 4000, 4600, 6000, 5088]
more_shares_2022 False
more_amount_2022 True
average_amount 5890.916666666667
std_amount 1077.077175997874
highest_amount 7800
lowest_amount 4000
highest_amount_quarter 3
lowest_amount_quarter 9
```

New Approach, Step 3

We now take the result text and add both the calculations and its solutions to the original problem:

````
# Question

... <as above> ...

## Relevant calculations and comparisons

```python
shares_2022 = [21, 20, 26, 28] # millions of shares
... <as above> ...
```

## Evaluation of relevant calculations and comparisons

```
````
````

```
more_shares_2022 False
... <as above> ...
```

```
# Answer
```

The GPT-4 generated answer is now correct:

```
The overall average and standard deviation of shares repurchased per quarter over these thre
```

- Average: $5,890.92 million
- Standard deviation: $1,077.08 million

Example 2

Take this harder example:

```
This is an extract from the Microsoft annual report for 2022:

Microsoft repurchased the following shares of common stock under the share repurchase progra

(In millions)   Shares      Amount      Shares      Amount      Shares      Amount
Year Ended June 30,         2022                2021                2020
First Quarter   21      $6,200      25      $5,270      29      $4,000
Second Quarter  20      6,233       27      5,750       32      4,600
Third Quarter   26      7,800       25      5,750       37      6,000
Fourth Quarter  28      7,800       24      6,200       28      5,088

This is an extract from the Google annual report for 2022:

Share Repurchases
In April 2022, the Board of Directors of Alphabet authorized the company to repurchase up to
Class A and Class C shares. As of December 31, 2022, $28.1 billion remains available for Cla
repurchases.
The following table presents Class A and Class C shares repurchased and subsequently retired
Year Ended December 31, 2021 Year Ended December 31, 2022
Shares Amount Shares Amount
Class A share repurchases 24 $ 3,399 61 $ 6,719
Class C share repurchases 383 46,875 469 52,577
Total share repurchases 407 $ 50,274 530 $ 59,296
Class A and Class C shares are repurchased in a manner deemed in the best interest of the co
stockholders, taking into account the economic cost and prevailing market conditions, includ
prices and volumes of the Class A and Class C shares. Repurchases are executed from time to
general business and market conditions and other investment opportunities, through open mark
privately negotiated transactions, including through Rule 10b5-1 plans. The repurchase progr
expiration date.
```

`Which company repurchased more a greater amount of shares, in total value, in 2021 and 2022`

Here are the extracted calculations:

```
# Definitions
# Microsoft share repurchase amounts by quarter and year in millions of dollars
ms_q1_2021 = 5270 # million dollars
ms_q2_2021 = 5750 # million dollars
ms_q3_2021 = 5750 # million dollars
ms_q4_2021 = 6200 # million dollars
ms_q1_2022 = 6200 # million dollars
ms_q2_2022 = 6233 # million dollars
ms_q3_2022 = 7800 # million dollars
ms_q4_2022 = 7800 # million dollars

# Google share repurchase amounts by year and class in millions of dollars
goog_a_2021 = 3399 # million dollars
goog_c_2021 = 46875 # million dollars
goog_a_2022 = 6719 # million dollars
goog_c_2022 = 52577 # million dollars

# Calculations
# Microsoft total share repurchase amount by year in millions of dollars
ms_2021 = ms_q1_2021 + ms_q2_2021 + ms_q3_2021 + ms_q4_2021 # million dollars
ms_2022 = ms_q1_2022 + ms_q2_2022 + ms_q3_2022 + ms_q4_2022 # million dollars

# Google total share repurchase amount by year in millions of dollars
goog_2021 = goog_a_2021 + goog_c_2021 # million dollars
goog_2022 = goog_a_2022 + goog_c_2022 # million dollars

# Microsoft and Google total share repurchase amount in 2021 and 2022 combined in millions
ms_total = ms_2021 + ms_2022 # million dollars
goog_total = goog_2021 + goog_2022 # million dollars

# Comparisons
# Which company repurchased more a greater amount of shares, in total value, in 2021 and 202
goog_more_than_ms = goog_total > ms_total # boolean
ms_more_than_goog = ms_total > goog_total # boolean
```

Executing this gives:

```
ms_2021 22970
ms_2022 28033
goog_2021 50274
goog_2022 59296
ms_total 51003
goog_total 109570
goog_more_than_ms True
```

```
ms_more_than_goog False
```

And now GPT-4 produces the correct answer:

```
Google repurchased more a greater amount of shares, in total value, in 2021 and 2022 combine

- Google: $109.57 billion
- Microsoft: $51.003 billion
```

Without this technique, GPT-4 sometimes gets the answer wrong, e.g. one sample gave:

```
Microsoft repurchased more a greater amount of shares, in total value, in 2021 and 2022 comb

- Microsoft: $6,200 + $6,233 + $7,800 + $7,800 + $5,270 + $5,750 + $5,750 + $6,200 + $4,000
- Google: $50,274 + $59,296 = $109,570 million
```

Example 3

Here is a classic Chat-GPT fail:

> What is the number of years it takes for an item growing at 30% annually to double?

A typical failing response:

```
One way to approach this question is to use the formula for compound interest, which is:

A = P(1 + r)^n

where A is the final amount, P is the initial amount, r is the annual interest rate (as a de

We want to find n such that A = 2P, meaning the item has doubled in value. Substituting thes

2P = P(1 + 0.3)^n

Dividing both sides by P, we get:

2 = (1 + 0.3)^n

Taking the natural logarithm of both sides, we get:

ln(2) = n ln(1 + 0.3)

Solving for n, we get:

n = ln(2) / ln(1 + 0.3)

Using a calculator, we get:
```

```
n ~= 2.53

Therefore, it takes about 2.53 years for an item growing at 30% annually to double.
```

The final answer - 2.53 - is wrong. GPT-4 can do quite a lot of good mathematical explanation and reasoning, but it can't do the final calculation step.

Calculation equipping to the rescue! Hope is restored! The extracted calculations are:

```
# Definitions
growth_rate = 0.3 # annual growth rate as a decimal fraction
initial_amount = 1 # initial amount of the item in unknown units
doubling_amount = 2 # amount of the item after doubling in unknown units

# Calculations
# Use the formula A = P * (1 + r)^t, where A is the final amount, P is the initial amount, r
doubling_time = np.log(doubling_amount / initial_amount) / np.log(1 + growth_rate) # time in
```

The evaluation is:

```
doubling_time 2.6419267958111403
```

and the final written answer is now correct:

> The number of years it takes for an item growing at 30% annually to double is approximately 2.64 years.

Happiness.

Example 4

We have applied this technique to the very large and complex problem originally quoted - the comparison of two financial reports. The full text of our version of the problem can be found here

While not perfect, the technique vastly improves the generated text. The generated calculations are as follows:

````
## Relevant calculations and comparisons

''''python
# Definitions
gap_net_sales = 4.04 # billion USD
gap_comparable_sales = 0.01 # fraction
gap_online_sales = 0.39 # fraction of net sales
gap_store_sales = 0.01 # fraction of net sales
gap_gross_margin = 0.374 # fraction
gap_operating_income = 186 # million USD
gap_operating_margin = 0.046 # fraction
````

```
gap_net_income = 282 # million USD
gap_diluted_eps = 0.77 # USD
gap_inventory = 3.04 # billion USD
gap_capital_expenditures = 577 # million USD
gap_dividend = 0.15 # USD per share
gap_share_repurchases = 12 # million USD
gap_store_count = 3380 # number of stores

lulu_net_revenue = 1.9 # billion USD
lulu_comparable_sales = 0.22 # fraction
lulu_ecommerce_revenue = 0.413 # fraction of net revenue
lulu_store_revenue = 0.486 # fraction of net revenue
lulu_gross_profit = 1.0 # billion USD
lulu_gross_margin = 0.559 # fraction
lulu_operating_income = 352.4 # million USD
lulu_operating_margin = 0.19 # fraction
lulu_net_income = 255.5 # million USD
lulu_diluted_eps = 2.00 # USD
lulu_inventory = 1.7 # billion USD
lulu_capital_expenditures = 175.6 # million USD
lulu_share_repurchases = 17 # million USD
lulu_store_count = 623 # number of stores

# Calculations
gap_gross_profit = gap_net_sales * gap_gross_margin # billion USD
lulu_net_revenue_growth = (lulu_net_revenue - 1.3) / 1.3 # fraction
lulu_store_revenue_growth = (lulu_store_revenue - 0.488) / 0.488 # fraction
lulu_ecommerce_revenue_growth = (lulu_ecommerce_revenue - 0.404) / 0.404 # fraction
gap_inventory_growth = (gap_inventory - 2.72) / 2.72 # fraction
lulu_inventory_growth = (lulu_inventory - 0.92) / 0.92 # fraction
gap_capital_expenditures_growth = (gap_capital_expenditures - 308) / 308 # fraction
lulu_capital_expenditures_growth = (lulu_capital_expenditures - 122.5) / 122.5 # fraction
gap_dividend_yield = gap_dividend * 4 / gap_diluted_eps # fraction
lulu_dividend_yield = 0 # fraction
gap_share_repurchases_yield = gap_share_repurchases / gap_net_income # fraction
lulu_share_repurchases_yield = lulu_share_repurchases / lulu_net_income # fraction

# Comparisons
gap_net_sales > lulu_net_revenue # true/false
gap_comparable_sales < lulu_comparable_sales # true/false
gap_online_sales < lulu_ecommerce_revenue # true/false
gap_store_sales < lulu_store_revenue # true/false
gap_gross_margin < lulu_gross_margin # true/false
gap_operating_income < lulu_operating_income # true/false
gap_operating_margin < lulu_operating_margin # true/false
gap_net_income > lulu_net_income # true/false
```

```
gap_diluted_eps < lulu_diluted_eps # true/false
gap_inventory_growth > lulu_inventory_growth # true/false
gap_capital_expenditures_growth > lulu_capital_expenditures_growth # true/false
gap_dividend_yield > lulu_dividend_yield # true/false
gap_share_repurchases_yield < lulu_share_repurchases_yield # true/false
gap_store_count > lulu_store_count # true/false
```

This is not bad, but there are new assumed numbers such as `2.72` in the generated calculations that do not relate to anything in the source text. UPDATE: this fabrication can be eliminated through more stringent prompting, see the appendix.

Adding the following lines in the prompt eliminated the addition of hallucinated assumptions in the calculations:

```
* Avoid new assumptions in this section, if you make an assumption document it.
```

In the latest implemented proof-of-concept (generating Javascript for the calculations), the generated calculations became:

```
// Definitions
let gap_net_sales = 4.04; // billion USD, total net sales for Gap Inc. in Q3 2022
let gap_comparable_sales = 0.01; // decimal, year-over-year growth rate of comparable sales
let gap_gross_margin = 0.374; // decimal, reported gross margin as a percentage of net sales
let gap_operating_margin = 0.046; // decimal, reported operating margin as a percentage of
let gap_diluted_eps = 0.77; // USD, reported diluted earnings per share for Gap Inc. in Q3
let gap_inventory = 3.04; // billion USD, ending inventory for Gap Inc. in Q3 2022
let gap_capital_expenditures = 0.577; // billion USD, year-to-date capital expenditures for

let lululemon_net_sales = 1.9; // billion USD, total net revenue for lululemon in Q3 2022
let lululemon_comparable_sales = 0.22; // decimal, year-over-year growth rate of total compa
let lululemon_gross_margin = 0.559; // decimal, gross margin as a percentage of net revenue
let lululemon_operating_margin = 0.19; // decimal, operating margin as a percentage of net
let lululemon_diluted_eps = 2.0; // USD, diluted earnings per share for lululemon in Q3 2022
let lululemon_inventory = 1.7; // billion USD, ending inventory for lululemon in Q3 2022
let lululemon_capital_expenditures = 0.176; // billion USD, capital expenditures for lululem

// Calculations
let net_sales_difference = lululemon_net_sales - gap_net_sales; // billion USD, difference
let net_sales_ratio = lululemon_net_sales / gap_net_sales; // decimal, ratio of net sales be
let comparable_sales_difference = lululemon_comparable_sales - gap_comparable_sales; // dec
let comparable_sales_ratio = lululemon_comparable_sales / gap_comparable_sales; // decimal,
let gross_margin_difference = lululemon_gross_margin - gap_gross_margin; // decimal, differe
let gross_margin_ratio = lululemon_gross_margin / gap_gross_margin; // decimal, ratio of gro
let operating_margin_difference = lululemon_operating_margin - gap_operating_margin; // dec
let operating_margin_ratio = lululemon_operating_margin / gap_operating_margin; // decimal,
let diluted_eps_difference = lululemon_diluted_eps - gap_diluted_eps; // USD, difference in
let diluted_eps_ratio = lululemon_diluted_eps / gap_diluted_eps; // decimal, ratio of dilute
```

```
let inventory_difference = lululemon_inventory - gap_inventory; // billion USD, difference
let inventory_ratio = lululemon_inventory / gap_inventory; // decimal, ratio of ending inve
let capital_expenditures_difference = lululemon_capital_expenditures - gap_capital_expenditu
let capital_expenditures_ratio = lululemon_capital_expenditures / gap_capital_expenditures;

// Comparisons
let net_sales_higher = lululemon_net_sales > gap_net_sales; // boolean, true if lululemon ha
let comparable_sales_higher = lululemon_comparable_sales > gap_comparable_sales; // boolean,
let gross_margin_higher = lululemon_gross_margin > gap_gross_margin; // boolean, true if lu
let operating_margin_higher = lululemon_operating_margin > gap_operating_margin; // boolean,
let diluted_eps_higher = lululemon_diluted_eps > gap_diluted_eps; // boolean, true if lulul
let inventory_higher = lululemon_inventory > gap_inventory; // boolean, true if lululemon ha
let capital_expenditures_higher = lululemon_capital_expenditures > gap_capital_expenditures;
```

However the risk that code-generation will encode new hallucinations is real. Some techniques to deal with this are

- Heavy prompting to avoid new assumptions, as above
- Filtering or rejection of calculations that don't conform to expected requriements
- Size restrictions rejecting new hallucinated calculations when inputs are small

For end user-experience it may also be very important to have the models include a textual explanation of any assumptions made in generating the calculations. This is feasible given the above through another model invocation to explain any assumptins made.

Example 5

Take pretty much any basic calculation problem that doesn't involve really simple numbers, e.g. from This LA County math exam:

> Fred's gross salary is $850 per week. From his salary, 11% is removed for federal deductions; 5.5% for state deductions; and 6.2% for the company's pension plan. If each of these deductions is taken as a percent of Fred's gross salary, what is his net salary for a fourweek period?

Without calculation equipping the answer is wrong:

> $2448.20

With calculation equipping the answer is correct:

> Fred's net salary for a four-week period is $2628.20.

In the Javascript-generating prototype, the generated calculation code (with full generated documentation comments) is:

```
// Definitions
let gross_salary_per_week = 850; // in dollars, the amount Fred earns before deductions
let federal_deduction_rate = 0.11; // in fraction, the percentage of gross salary removed fo
let state_deduction_rate = 0.055; // in fraction, the percentage of gross salary removed for
let pension_deduction_rate = 0.062; // in fraction, the percentage of gross salary removed f
let weeks_per_period = 4; // in weeks, the length of the period for which the net salary is

// Calculations
let federal_deduction_per_week = gross_salary_per_week * federal_deduction_rate; // in dolla
let state_deduction_per_week = gross_salary_per_week * state_deduction_rate; // in dollars,
let pension_deduction_per_week = gross_salary_per_week * pension_deduction_rate; // in dolla
let total_deduction_per_week =
  federal_deduction_per_week + state_deduction_per_week + pension_deduction_per_week; // in
let net_salary_per_week = gross_salary_per_week - total_deduction_per_week; // in dollars, t
let net_salary_per_period = net_salary_per_week * weeks_per_period; // in dollars, the amoun
```

Refinements

The emphasis here is on "calculation" not "math" or "algebra" - we want to reliably perform the simpler, purely calculational end of mathematical reasoning. This means detecting and emitting the calculational subset of reasoning and executing it with 100% accuracy - especially the kind that may easily be present in large quantities in chat discussion about large texts. The financial document comparisons above being good examples. Another aim is to not to try to perform any other kind of reasoning. In this setting, calculations may involve a considerable amount of data processing and reduction.

We want to make sure that the numbers computed are correct - even if the calculation itself is not strictly the right calculation to be doing. Ideally the presence of the calculation code and results will not greatly disturb the other reasoning or textual emit going on.

We are exploring a number of refinements to the described technique.

Refinement: Emitting checks

It is possible to add a section to the calculation code prompt requesting a check on the values. Here is an example prompt addition:

```
In the optional "Check" section:
* If possible, define the label \‘check\‘ comparing a combination of the calculated values t
* Omit this section completely if it contains no definitions.
* The \‘check\‘ value should evaluate to a single true/false boolean.
* Do NOT include the calculated true/false value for this label.
```

We are assessing the value of this check in eliminating false arithmetic. It appears useful in "word puzzle" problems but it is unclear if it has broader utility for helping to ensure correctness and soundness.

One problem with emitting checks is that the model may attempt to perform the calculation as part of the check, which is agsinst the purpose of this work. Milder checks can be used, e.g. just requesting range checking:

```
* If appropriate, define the label \‘check_message\‘ checking if results lie within the know
```

However beware this may lead to hallucinated range checks. Another problem is that the checks may be “surprising yet logically reasonable”, e.g. consider this question:

> Mrs. Hilt and her sister drove to a concert 78 miles away. They drove 32 miles and then stopped for gas. Her sister put 28 gallons of gas in the car. How many miles did they have left to drive? ——————-

After correct calculation code, the emitted check is:

```
const check_message = gas_added > 20 ? "That’s a lot of gas for a short trip." : ""; // [str
```

It’s true - the irrelevant information of 28 gallons really is a lot of gas for a short trip! And one can indeed imagine a bright grade 2 child putting up their hand and querying why so much gas is needed. But the check is, in practice, not helpful.

Refinement: Reducing Date and Time calculations

Numeric calculation can easily lead to incorrect code calculating any use of numbers that is textual or non-standard arithmetic, e.g. with dates and times. These are a tricky domain with many pitfalls and possible assumptions: everything from localization formats to incorrect use of decimal reprentations like `18.5` for `6:30pm`.

One approach to banning all dates and times is to add this prompt directive:

```
* Avoid all date/time calculations. Reduce to whole days, hours, minutes and arbitrary secon
```

Another option is to avoid or ban all textual date times:

```
* Avoid all date/time calculations
* Parse all textual dates and times using the \‘PARSEDATETIMEFAIL\‘ function
```

The second directive will cause the emit of a function that will fail to exist when the calculation code is run, suppressing the use of calculation. However, it is likely this overly impacts innocuous or simple uses of dates and times.

A third alternative is to attempt to convert to a standard DateTime object with good computational support. However this is still prone to localization and other problems. Additionally, such an object should have unlimited range (any date/time, including day 1,000,000 BC) if used in arbitrary chat scenarios: many libraries limit the ranges usable.

A fourth alternative is to instruct the reduction of the date-time computations to more tractable computations, e.g.

```
* Avoid all date/time calculations. Reduce to whole days, hours, minutes and arbitrary secon
```

Refinement: Avoiding solving equations and other algebra

The aim here is to calculate with numbers, not perform high-school algebraic maths. GPT-4 loves to pretend that it is good at doing maths, but in reality that should not be part of the generated code emit, as GPT-4 is unreliable at this problem domain.

This is a tricky thing to delimit, but we have experimented with adding the following to the calculation-section prompt:

```
* Do NOT solve equations, simply write relevant calculations.
```

Further filtering of pseudo-mathematical code is likely required to restrict ourselves to high-quality calculation.

Refinement: Integer division

One identified mistake in generated code is the mis-use of integer v. floating-point division.

We are experimenting with adding this to the prompt to help guide the model to correct this:

```
* Use integer division when appropriate.
```

Refinement: Alternative architectures

During this investigation we investigated some other alternatives:

- We tried variations using a single model invocation, producing a mix of calculations plus text. Some examples clearly required conditional text, which we started to investigate by making the generated final text be conditional/templated/interpolated. However, the longer longer financial report examples above convinced us that too much reasoning remained in text generation, and that it is a clearer and simpler architecture to use a specific model invocation to enrich with an calculation program. Certainly a single invocation is viable for smaller examples.
- We tried variations using a single model invocation that generates only a program, which is encouraged to print the full final text - so the only output was a program which could include calculation content. However that seemed to result in output texts more like those programmers write for diagnostics or output - terse, rather than well-written human-facing output text. Again, it seems a clearer and simpler architecture to distinguish between a phase that enriches with calculations, and a phase which uses GPT-4 in text-generation mode.

Refinement: Format for calulation code

We have left open what format should be used for calculations code. For convenience we have shown generating Python and Javascript. However introducing arbitrary code generation and execution in general-purpose languages is not necessary for this technique - instead the prompts should continue to be devloped to demand the generation of highly restricted calculation code. A limited subset of Python+numpy or Javascript or similar could still be used but a processing step should be added to strictly check the conformance of the calculations to a well-defined known subset. Careful sandboxing of the execution (or careful interpretation) will also be required.

Evaluation

See evaluation.

Related Work

There's a recent survey paper on techniques to augment Language Models, see https://arxiv.org/abs/2302.07842. Most the papers used retraining or fine tuning specifically to use such tools, but the survey is comprehensive and a useful guide to equips (augmentations).

The concept of specifically augmenting reasoning by generating programs in specific target langauges (e.g. calculation code) is surprisingly relatively new, and has an exposition in Program-aided Language Models from Nov 2022. The technique described here is a variation of this technique.

Much of the work on math and GPT-4 attempts to improve its more advanced mathematical capabilities, e.g. for algebra, geometry, problem solving, see https://arxiv.org/abs/2303.05398 for a recent paper from MSR.

The basic idea that you can get LLMs to emit Python to handle arithemtic calculation as part of chain-of-reason prompting has been around a long time, e.g. see the prompt here: https://huggingface.co/datasets/LangChainHub-Prompts/LLM_Math and the source code for the LangChain module. The work here can be seen as a focused, dedicated elaboration of this technique, notably

- Trying harder to focus on emitting reliable numeric calculation only, rather than arbitrary code
- Focusing also on numeric comparisons
- Focusing on code that is "relevant to answering the question" rather than just answering it
- Emitting documentation and units of measure
- Emitting mathematical checks
- Evaluating the technique

Conclusion

GPT-4 can't do numeric calculations or comparisons. But it can write pretty good calculation code. By using a two-phase approach we can equip GPT-4 with numeric calculation by writing the calculation code and evaluating it with Python or a similar interpreter.

This has potentially huge advantages:

1. Applications based on GPT-4 become much more reliable at numeric calculation.
2. A major cause of reputation loss is greatly reduced.
3. The derivation of all numbers in the output can be explained through the presentation of the calculation code that derived it.
4. Alternative output formats such as spreadsheets, executable notebooks or outright code can be produced to back the response.
5. The calculations can be automatically assessed and filtered for certain properties.

Further:

1. We speculate that with further prompting properties such as units (e.g. dollars) and multiplicative constants (e.g. millions) can be really carefully tracked and assessed. Unit mistakes are pernicious and the automated extraction of unit derivations can help here.
2. We speculate that there are many other classes of "equips" that can be dealt with this way. For example, we believe GPT-4 can be "equipped" with Datalog, or an SMT-solver, or Wolfram, or further programmatic data retrieval, or SQL queries.
3. There are a wide variety of "augment", "equip" and "chain of reasoning" architectures. Early versions of this work used a two step approach, where calculation and templated answer were written in one model invocation. Other architectures could ask which equips are most relevant, or iterate on equips until no more information can be incorporated. There are many tradeoffs and much to explore.

The technique needs rigorous evaluation and refinement and we encourage collaboration. Please iterate with us and take this further. If publishing externally please include us as co-authors.