Evaluating "GPT4 Equipped with Numeric Calculation"

See the report for an overview of the technique.

> Caveat!!! this work is about calculation, not math. Much of the work on math and GPT-4 attempts to improve its more advanced mathematical capabilities, e.g. for algebra, geometry, problem solving, is mostly based on word problems, and the typical evaluation problem sets are oriented in this way. This is understandable for LLM-based research - why would be use an LLM to try to add "16.84812 + 19.29039" let alone to evaluate trigonometric functions? However in real-life chat people will try to do exactly these kinds of problems: there is a strong user expectation that they can throw in any calculation and the chat will get it right, or else refuse to answer.
>
> NOTE: Evaluation is currently being done using Javascript codegen. We may look into Python codegen and other options.

TLDR

We developed some indicative problem sets that exhibit the following characteristics. These sets were developed partly in order to explore the boundary of what this technique can handle.

Raw calculation (decimals, integers, math functions):

- without equip: **50% mistake rate** (77/153), many extreme mistakes
- with equip: **0% mistake rate** (153/153)

Financial calculations (mortgage, rate of return, WACC):

- without equip: **57% mistake rate**, 4/14
- with equip: **14% mistake rate** 12/14

Tabular data calculations:

- without equip: **50% mistake rate** 4/8
- with equip: **0% mistake rate** 8/8

Unit conversion calculations problems

- without equip: **36% mistake rate** 7/11
- with equip: **0% mistake rate** 11/11

Date calculations:

- without equip: **30% mistake rate** 7/10
- with equip: **20% mistake rate** 8/10

Word puzzles:

- Without equip: **11% mistake rate** (253 failures 2050/2303)

- With equip: **6.2% mistake rate** (145 failures 2158/2303)

The impact of some variations on codegen were also tested:

- With equip (emitChecks): 140 failures 2163/2303
- With equip (noEliminateDateTime): 153 failures 2150/2303
- With equip (noEmitComparisons): 164 failures 2139/2303
- With equip (noEmitUnits): 166 failures 2137/2303

Details: Raw Calculation

We created a set of decimal and integer calculation problems like this:

> What is the result of adding -942.12 and 1441.23? Give answer rounded to two decimal places.
>
> What is the result of multiplying -942.12 by 1441.23? Give answer rounded to two decimal places. ... What is the natural logarithm of 1441.23? Give answer rounded to two decimal places.
>
> Is 0.45 plus 0.67 less than or equal to 1 minus (1/3)? Answer "true" or "false" without quotes.
>
> Is 100 divided by 5 less than or equal to 20? Answer "true" or "false" without quotes.

With categories:

- decimal calculation (1, 2, 3, 4 decimal places)
- decimal comparison
- integer comparison

The results are:

- without equip: 76/153 (50% failure rate, many extreme mistakes)
- with equip: 150/153 (2% mistake rate, only minor mistakes, see below)

Without equip, GPT-4 is sometimes surprisingly good at knowing tables of existing functions like sin, cos, exp:

> expected: "2.1", actual: "2.1", question: What is the natural logarithm of 8.131? Give answer rounded to one decimal places.

However the scale of mistakes is also extraordinary e.g.

> expected: "90.79", actual: "40.11", question: Calculate $(2.12^{3)}2$. Give answer rounded to two decimal places.
>
> expected: "2211733.5", actual: "1,948,789,000.7", question: What is 6.21 raised to power 8. Give answer rounded to one decimal places.

With equip, 98% of answers were exactly correct. The remaining minor mistakes were errors in the last decimal place:

```
expected: "24.533", actual: "24.532", question: What is e raised to power 3.2.  Give answer
expected: "7.3891", actual: "7.3890", question: What is the result of calculating e raised t
expected: "7.2733", actual: "7.2732", question: What is the natural logarithm of 1441.23?  (
```

These stemmed from

- Two cases of incorrect textual rounding of a correctly calculated result in the final GPT-4 question-answering phase
- One case where the calculated code used a hardwired approximation to `e`:

  ```
  const e = 2.71828; // base of natural logarithm [unknown]
  ```

These cases can clearly be handled by prompt refinements (e.g. move rounding computations into the calculated code, and always using precise values of constants like `e` and `pi`).

Details: Mathematical word puzzles

We used a problem set of 2300 childrens maths puzzles, up to grade 6 US curriculum. The puzzles are primarily word problem solving, not calculation.

> NOTE: essentially every problem in this data set can be made to fail with raw GPT-4 simply by making the numbers involved sufficeintly large or adding decimal places. The existing GPT-4 pass rates for this problem set are somewhat deceptive as they assume child-like numbers are involved in real-world problems.

We ran the strategy described here on a modified version of this data set where:

- some additional instructions were added to the questions specifying exact intended output formats
- some answers were corrected (the data set contained mistakes)
- some questions were clarified (they were highly ambiguous and open to interpretation, or assuming prior questions in the data set had been asked).
- the text of some questions triggered Responsible AI filters and was modified in otherwise harmless ways.

These adjustments applied to both GPT-4 and GPT-4e.

When run with the technique here, the error rate reduces from 11% to 6.5%:

```
Without numeric calculation equip: 253/2296 failures
With numeric calculation equip: 151/2296 failures
```

The mistake rates in the different grades of problems are affected as follows:

```
grade 1: 2 --> 1       // mistakes out of 194
grade 2: 7 --> 1       // mistakes out of 340
grade 3: 25 --> 12     // mistakes out of 808
grade 4: 49 --> 19     // mistakes out of 301
grade 5: 31 --> 18     // mistakes out of 146
```

```
grade 6: 139 --> 94    // mistakes out of 514
```

The differrent kinds of problems are interesting and important.

Improved:

```
Subtraction: 29 --> 7
Sum: 10 --> 0
Multiplication: 8 --> 2
Floor-Division: 6 --> 5
Common-Division: 14 --> 7
Comparison: 22 --> 0
TVQ-Final: 3 --> 0
Surplus: 22 --> 6
Algebra-1: 21 --> 16            // note, largely out of zone, only partially calculational
```

Regressed:

```
LCM: 9 --> 13                   // note, largely out of zone, mostly word puzzle curiosities
GCD: 11 --> 13                  // note, largely out of zone, mostly word puzzle curiosities
Sequential-Operation: 3 --> 8   // note, largely out of zone, only partially calculational,
```

About the same:

```
Addition: 18 --> 16             // note, remaining are largely date/time calculations
Ceil-Division: 1 --> 2          // note, largely out of zone, mostly word puzzle curiosities
Ratio: 12 --> 7                 // note, ratio reduction involves LCM/GCD which isn't a calc
Algebra-2: 44 --> 35            // note, largely out of zone, only partially calculational
```

Notes:

- The big improvements lie in the calculational heart: subtraction, summation, multiplication, comparison, surplus and some division problems.
- In contrast, some areas such as LCM and GCD have been a little impaired. These problems are largely non-calculational mathematical reasoning and are likely vanishingly rare in real-world chat (except for students doing homework puzzles!). However we should continue to investigate the reasons that performance is impaired on this kind of problem, and what can be done to restrict the technique from attempting to work on this kind of problem.

As an example of the kind of problem that the equip does not improve, consider:

> Andrew's 4 friends decided to bring food as well. If each them brought 4 slices of pizza, how many slices of pizza do they have in total?

Both with and without equip the model answers "20" because it counts Andrew as an additional friend, the correct answer is 16. This kind of word-interpretation mistake is not solved by the technique here.

Details: Financial, Table, Unit, Date and other calculations

See the linked data sets for the kinds of problems we explored. As few examples are shown below:

Table calculations:

Example:

Month	Date	Interest	Principal	Ending Balance
1	4/2023	$500	$343	$199,657
2	5/2023	$499	$344	$199,313
3	6/2023	$498	$345	$198,968
4	7/2023	$497	$346	$198,622
5	8/2023	$497	$347	$198,275
6	9/2023	$496	$348	$197,928
7	10/2023	$495	$348	$197,579
8	11/2023	$494	$349	$197,230
9	12/2023	$493	$350	$196,880
10	1/2024	$492	$351	$196,529
11	2/2024	$491	$352	$196,177
12	3/2024	$490	$353	$195,824

`What is the total interest paid in these 12 months?`

Unit calculations:

Example:

`Convert 7482 joules to calories. Answer to two decimal places.`

Finance calculations:

Example:

`Suppose a firm has Cost of equity 4.2%, Equity $1934143, Cost of debt 4.2% and Debt $50,042`

Date calculations:

Example:

`How many days are there between April 25th and May 11th, excluding the last day?`

> Note, we estimate the date-time problems are an area we should proactively try to **not** apply this technique without much more confidence that results improve and accuracy is achieved. This is discussed in the report.

Assessing variations of calculation code

Taking the mathematical word puzzles, we tried variations on the prompt that eliminated some characteristics of the generated calculations. This really gives a measure of the proportion of word puzzles sensitive to this aspect of calculation.

```
Without equip:                    253 failures 2050/2303
With equip:                       145 failures 2158/2303
With equip (emitChecks):          140 failures 2163/2303
With equip (noEliminateDateTime): 153 failures 2150/2303
With equip (noEmitComparisons):   164 failures 2139/2303
With equip (noEmitUnits):         166 failures 2137/2303
```