



Benchmark

In this project you will learn how language models are compared, by testing how good they are at doing different jobs.

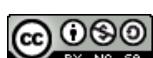
This project assumes you already understand the basics of how language models work.

If you haven't already completed the "Language Models" worksheet, you will understand this project better if you do that worksheet before this one.

The image shows a Scratch project titled "Benchmark". The stage features a scatter plot with a logarithmic scale for both axes, ranging from 1 to 10,000. Red 'X' marks represent incorrect answers, and green 'X' marks represent correct answers. A blue arrow points upwards and to the right across the plot area. The script on the stage sprite contains the following code:

```
when green flag clicked
  generate test
  reset
  run test
  test answer
  define test answer
    clear context
    if [submit v] then
      join [join v]
      join [What is sum? v]
      using temperature
      change num correct by 1
      switch costume to [correct v]
    else
      change num incorrect by 1
      switch costume to [incorrect v]
    end
  end
  if [is the language model ready to use? v] then
    use [text for initial context v]
  end
```

The script uses the "language model" extension, which includes blocks for "submit", "prompt", "using temperature", "clear context", "use [text for initial context]", and "Is the language model ready to use?".



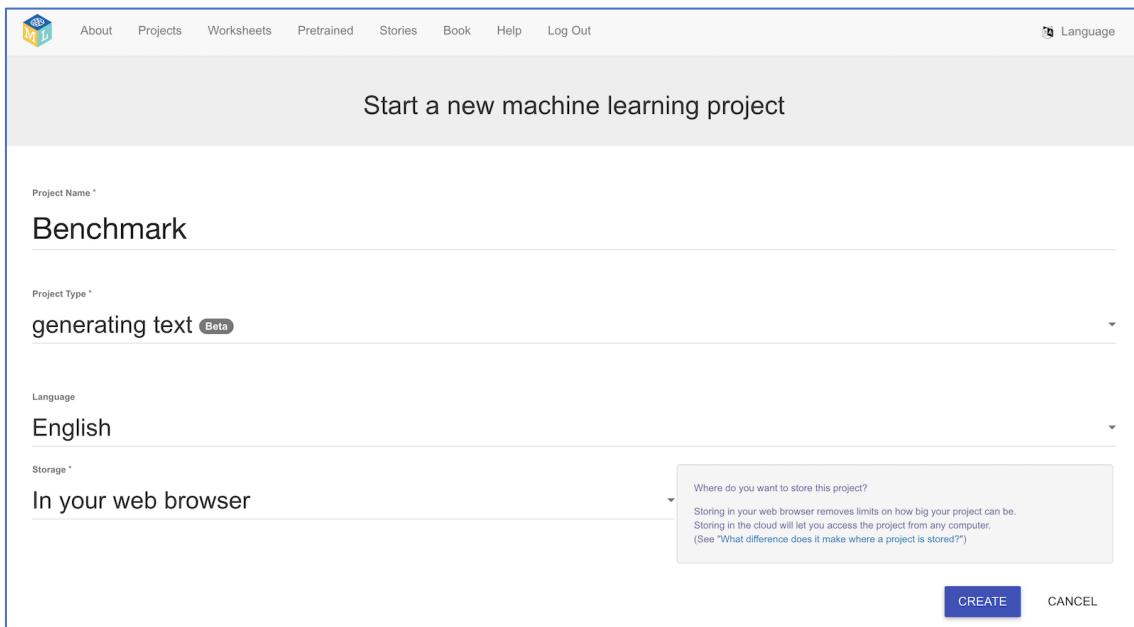
This project worksheet is licensed under a Creative Commons Attribution Non-Commercial Share-Alike License
<http://creativecommons.org/licenses/by-nc-sa/4.0/>

If you are under the age of 13, please only use a small language model with supervision from a trusted adult.

Generative AI can sometimes generate text that isn't nice or appropriate.

- 1.** Go to <https://machinelearningforkids.co.uk/>
- 2.** Click on “**Get started**”
- 3.** Click on “**Log In**” and type in your username and password
If you can't remember your username or password, ask your teacher or group leader to reset it for you.
- 4.** Click on “**Projects**” on the top menu bar
- 5.** Click the “**+ Add a new project**” button.

- 6.** Name your project “**Benchmark**” and set it to generate text.
Click **Create**



The screenshot shows a web-based form for creating a new machine learning project. At the top, there's a navigation bar with links for About, Projects, Worksheets, Pretrained, Stories, Book, Help, and Log Out. On the right side of the nav bar is a Language selection icon. Below the nav bar, the main title is "Start a new machine learning project". The form has several input fields:

- Project Name ***: The value is "Benchmark".
- Project Type ***: The value is "generating text" with a "Beta" badge next to it.
- Language**: The value is "English".
- Storage ***: The value is "In your web browser".

A small tooltip box appears near the "Storage" field, stating: "Where do you want to store this project? Storing in your web browser removes limits on how big your project can be. Storing in the cloud will let you access the project from any computer. (See 'What difference does it make where a project is stored?')". At the bottom right of the form are two buttons: a blue "CREATE" button and a white "CANCEL" button.

- 7.** You should see your new project in the projects list. Click on it.

8. Click on Small, and then click Next

If you are under the age of 13, please only use a small language model with supervision from a trusted adult. Generative AI can sometimes generate text that isn't nice or appropriate.

Type of language model

Toy **Small** Large language model

Computers can look for patterns in large numbers of documents. Language models generate text by using those statistical patterns to predict what word could come next. Creating a **toy** language model will show you how this works, and see the types of patterns computers look for (in documents you choose yourself). Configuring a **small** language model will show you how to use patterns found in millions of documents. All of this will help you to understand how **large** language models work in the real world.

Next

9. Choose a model architecture that you will be testing

There are multiple models to be tested.

Divide the models between all the students in the class.

Every model can be tested by someone in the class, and then you can compare your results.

If you are under the age of 13, please only use a small language model with supervision from a trusted adult. Generative AI can sometimes generate text that isn't nice or appropriate.

Type of language model

Toy **Small** Large language model

Model architecture

Smol
Qwen
Tiny Llama
Llama
Phi

Larger and more complex models can generate better text.

But larger models:

- * *will take longer to download*
- * *will need more storage space on your computer*

And more complex models:

- * *will need a faster and more powerful computer to run*

10. Click **Download**

11. You can choose any context window size, and click **Next**

12. Leave the high temperature and Top-p value, and click **Next**

13. Select **No initial context**, and click **Next**

The screenshot shows a user interface for configuring a machine learning model. It includes three main sections: 1. 'Size of context window' with a dropdown menu set to '512'. 2. 'Temperature' and 'Top-p' both set to their maximum values ('high'). 3. 'Initial context' where the 'No initial context' button is selected. Each section has a 'Review' button in the top right corner.

What are you doing?

The objective for this project is to find out how good these different language models are.

You have one of the language models to test.

There is rarely such a thing as a “best” or “worst” model. Different models are good at different tasks. When we test models, we test them to see how good they are at the job we want them to do.

For this project, we will be testing to see how good the models are at **giving answers to addition sums**.

- 14.** Type a prompt asking the model to give the answer to a simple sum
“*What is 2 + 3?*”

The interface shows two horizontal sliders at the top: 'temperature' and 'top-p'. Both are set to 'high'. Below them is a section for 'Initial context' with 'No initial context' selected. A 'Review' button is in the top right. A 'Prompt' input field contains 'What is 2 + 3?'. To the right are 'Reset' and 'Generate' buttons. The main output area displays the text 'The answer to your query is 5.'

- 15.** Click on **Reset** and try asking an addition sum with larger numbers
“*What is 13 + 25?*”

The interface is identical to the previous one, with 'temperature' and 'top-p' both set to 'high'. 'No initial context' is selected in 'Initial context'. The 'Prompt' field contains 'What is 13 + 25?'. The output area shows '13 + 25 = 28'.

- 16.** Repeat this a few more times using different numbers.
Ask enough questions until you think you understand how good your model is at giving answers to sums.

The interface is identical to the previous ones. The 'Prompt' field contains 'What is 96 + 89?'. The output area displays the text 'Yes, a 96 + 89 combination can be expressed as 96 + 5.'

17. Compare your experience with the other students in your class
Ask everyone for a few words that describe their opinion of how good their model is at addition and use that to fill in this table.

Model	Model capability at simple addition
Smol	
Qwen	
Tiny Llama	
Llama	
Phi	
Zephyr	
Gemma	
RedPajama	

What have you done?

Different language models have different capabilities at mathematics. Some are better at it than others.

Your class has performed subjective evaluation of the models. That is quick and simple, and is a common approach.

But is it a reliable approach?

Next, you will try a different, more organised approach to manual testing.

18. Ask your model for an answer to the following sums.

Put a tick or cross in the grid for each sum based on whether the model gives the correct answer.

*Click **Reset** in between each sum.*

Ask other students in the class for their results with the other models

Sum	Example	Smol	Qwen	Tiny Llama	Llama	Phi	Zephyr	Gemma	Red Pajama
What is $2 + 5$?	✓								
What is $6 + 3$?	✓								
What is $4 + 8$?	X								
What is $12 + 6$?	✓								
What is $3 + 15$?	✓								
What is $18 + 5$?	X								
What is $52 + 7$?	✓								
What is $35 + 62$?	X								
What is $13 + 41$?	X								
What is $87 + 76$?	X								
What is $9238 + 4326$?	X								
results	5 / 11	/ 11	/ 11	/ 11	/ 11	/ 11	/ 11	/ 11	/ 11

19. What do these results show about how the models compare?

How do the results from this manual testing compare with the subjective evaluation?

What have you done?

Testing is a data-driven way to evaluate the models. Doing it manually is slow and means you can only do a small number of tests.

Computers are good at quickly doing things over and over again. We write code to give a computer instructions for how to quickly perform a task repeatedly. Next, you will write code in Scratch to do a lot of testing very quickly.

You need to describe, in code, how to know if the model gives the correct answer. To start with, use **contains** : if the model's answer contains the expected value, anywhere in it, it will count as correct.

question: “What is $12 + 7$?”

example model’s response: “I want to tell you that 19 is the answer”

expected answer: 19

Does the response contain the expected answer? Yes! It is correct.

This approach isn’t perfect.

question: “What is $12 + 7$?”

example model’s response: “The answer is nineteen”

expected answer: 19

Does the response contain the expected answer? No. But it was correct!

question: “What is $12 + 7$?”

example model’s response: “The answer is 2197”

expected answer: 19

Does the response contain the expected answer? Yes. But it was incorrect!

“contains” is a good test. It is not as reliable as manual testing. But it is simple and quick, which makes it a useful choice.

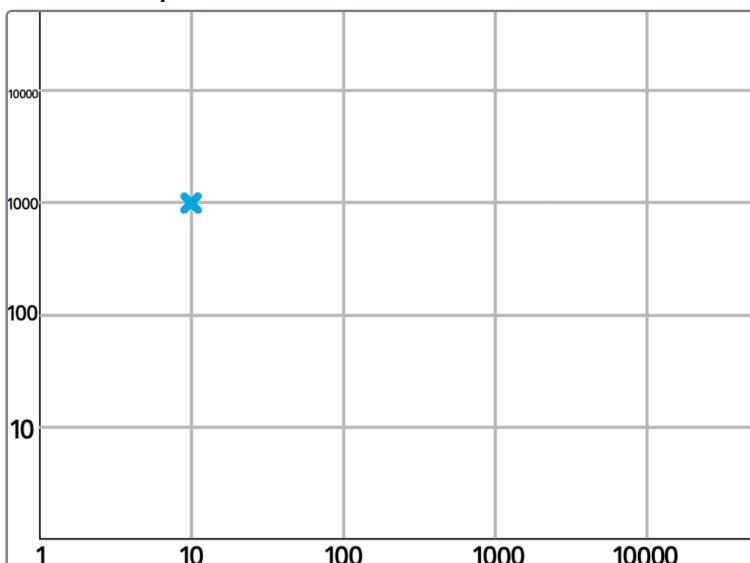
20. Click on the **Scratch 3** button (at the bottom of the page)

21. Click on **Project templates**

22. Open the **Benchmark** template

You will use this to plot addition sums on a graph.

*For example, the sum **What is 10 + 1000 ?** would be plotted here:*



The Scratch code will generate a random test for addition sums of numbers in each grid square.

For example, it will generate a sum to add a number between 100 and 1000

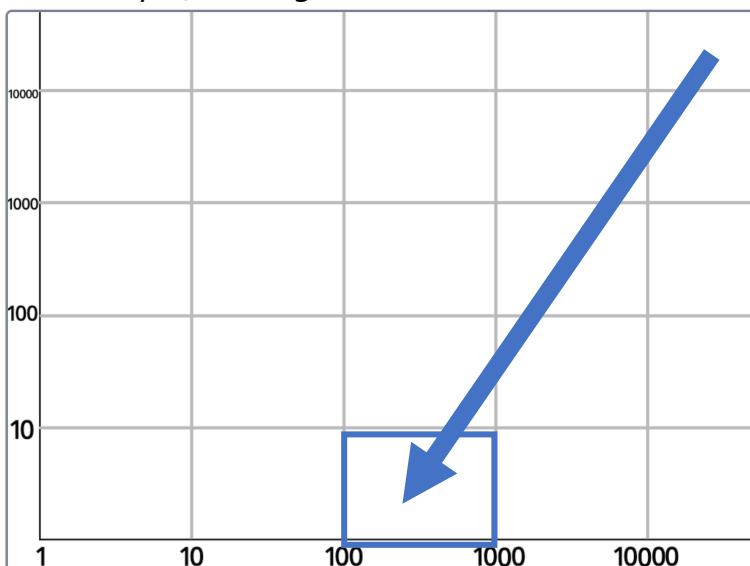
*to a number
between 1 and 10*

e.g.

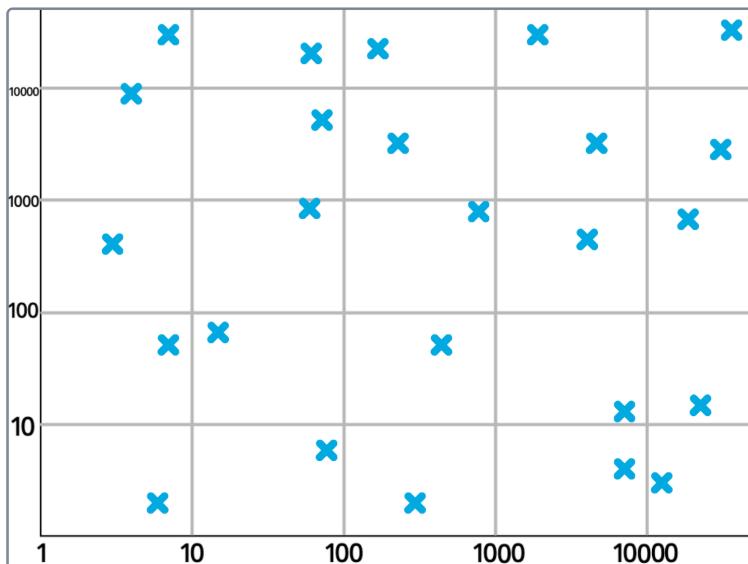
What is 452 + 3 ?

or

What is 377 + 8 ?



Clicking Green Flag will generate a different sum for each square, such as:



You will update this project so that each of these random sums is asked to the language model.

If the model's response contains the correct answer, the sum will be shown as a green cross.

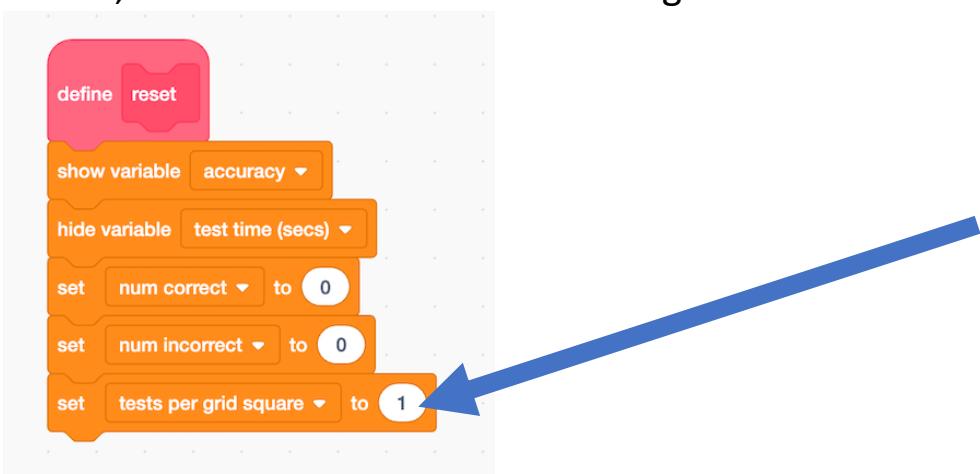
If the model's response does not contain the correct answer, the sum will be shown as a red cross.

You will see if your language model is better at sums with larger or smaller numbers.

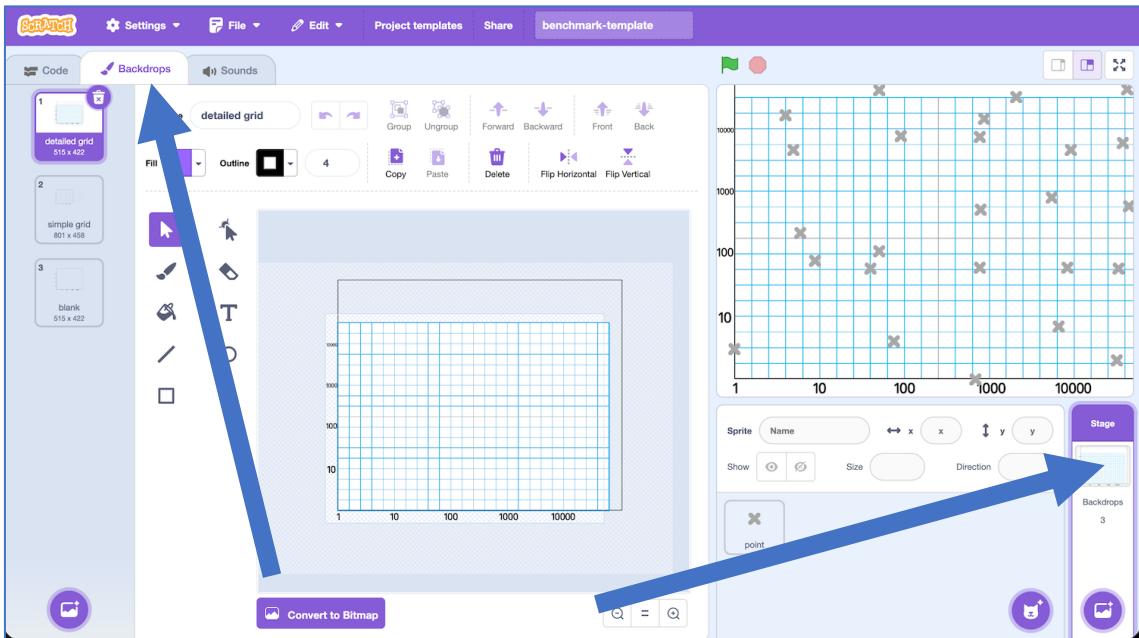
23. Click on the **Green Flag** a few times

See the sorts of sums that the project can generate

24. Increase the **tests per grid square** to generate more than 1 test for each grid square. This will improve how useful the results from your test can be, but will make the test take longer.

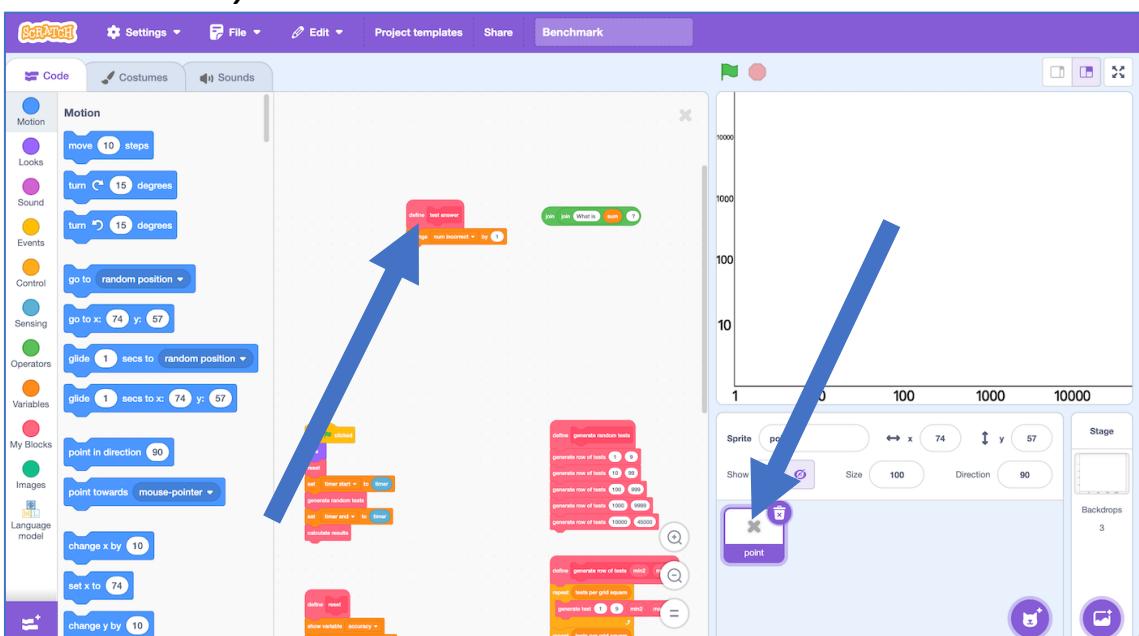


25. Choose the backdrop you would like for the Stage



26. Find the **test answer** block in the **point** sprite

This is where you will add code to check the model's answer

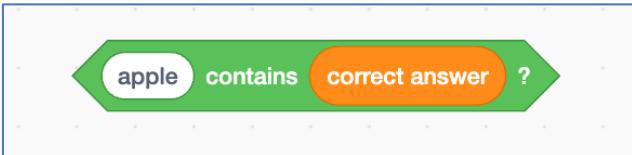


The sum is already next to it in green join blocks to help.

Your next step will be to add code that will:

- * ask your language model for an answer to the sum
- * check if the response **contains** the correct answer
- * updates the count of correct / incorrect answers
- * changes the colour of the cross to show if it's correct / incorrect

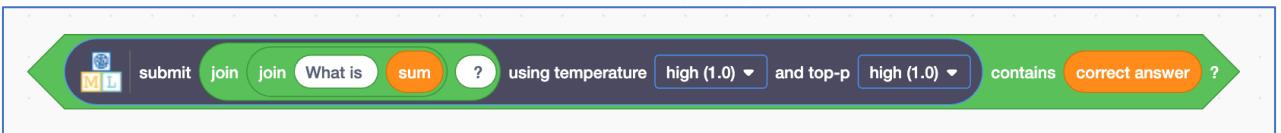
27. Start by preparing the check for the correct answer



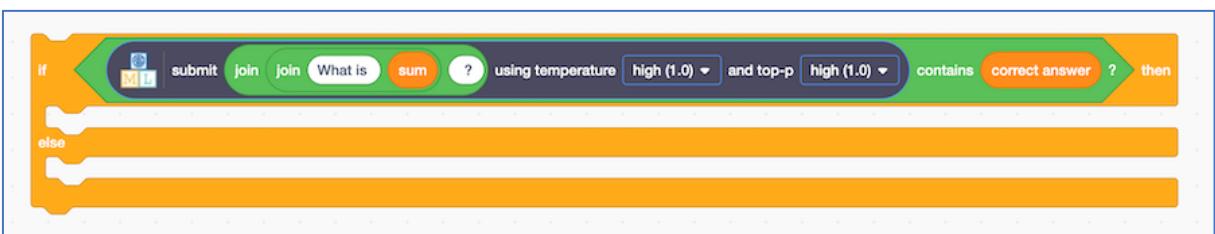
28. Add your language model block to the check



29. Put the sum question into the language model block

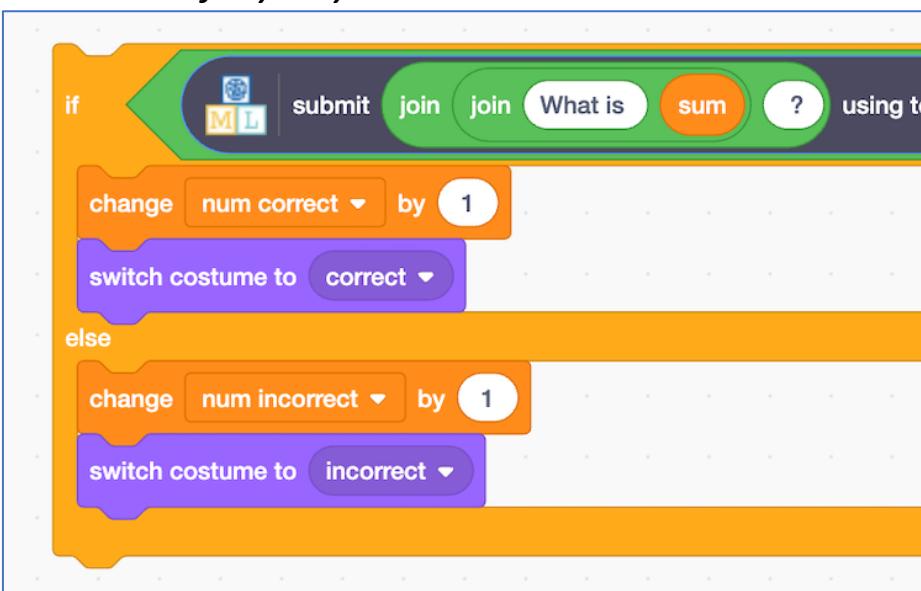


30. Put this check into an if ... else control

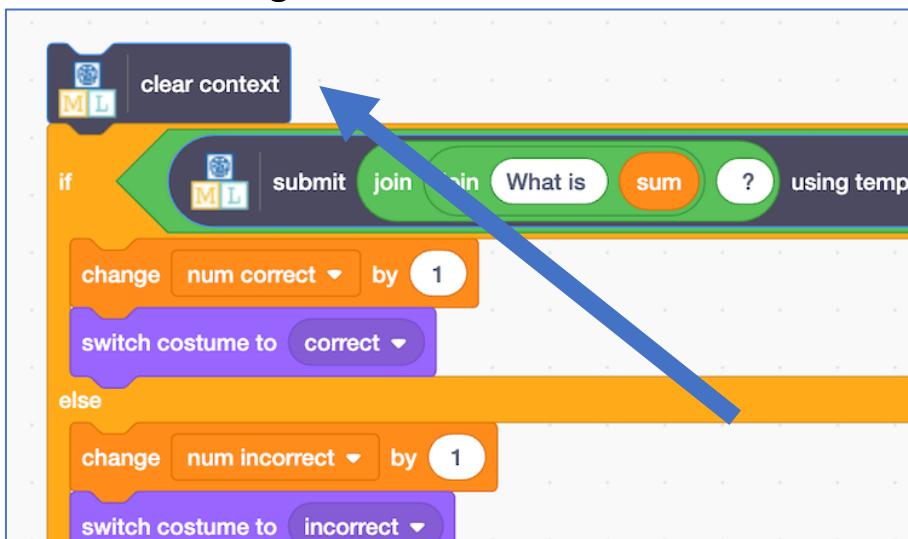


31. Add blocks to change colour and update the counts

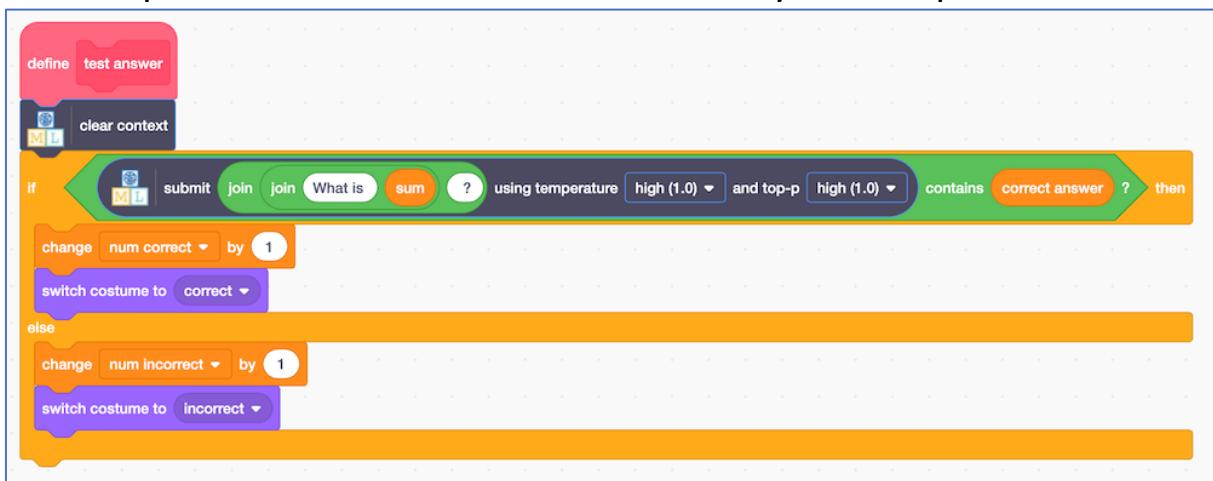
Do this carefully or your test results will not be valid



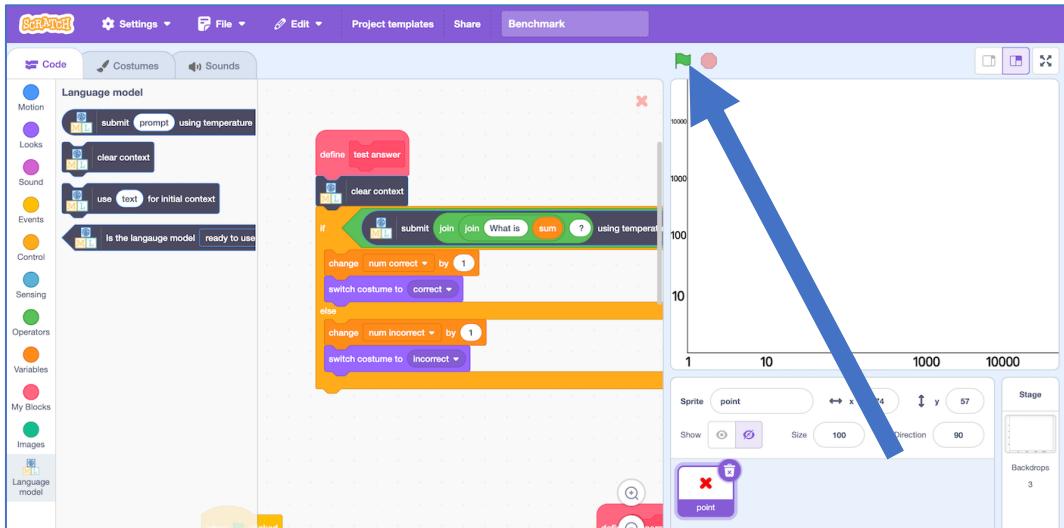
32. Reset the model before each test so the context does not affect the answers that it gives



33. Replace the test answer contents with your completed code

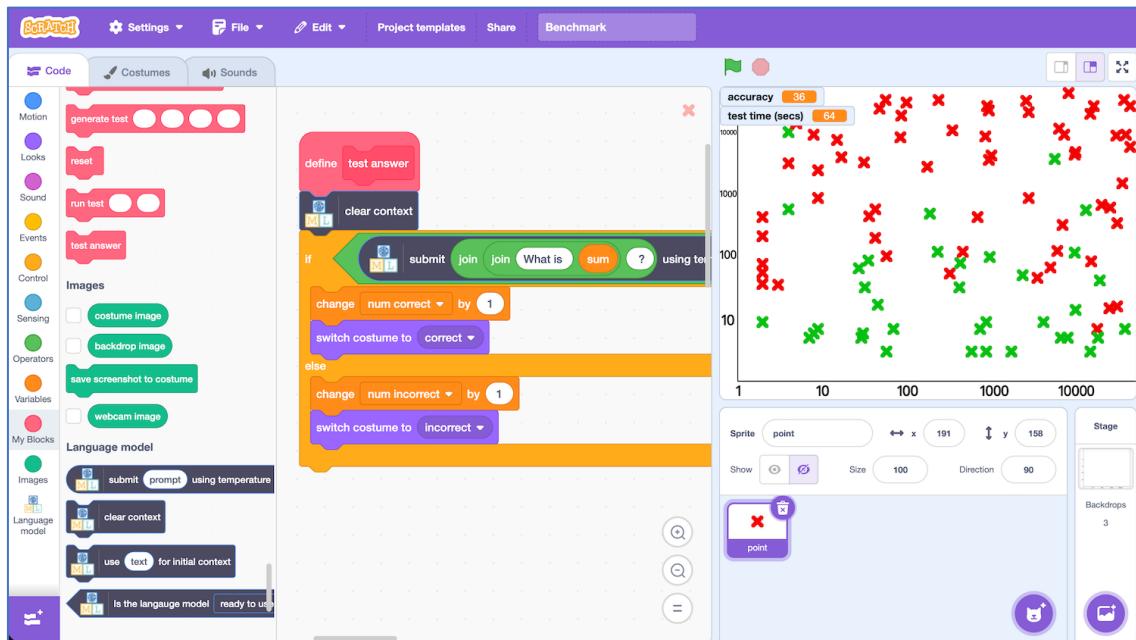


34. Time to test! Click the Green Flag



35. Wait for the test to complete

If you increased the number of tests in step 24 and this is taking too long, you can stop the project, decrease the number, and start it again.



36. Record your results

Make a note of the accuracy and the time taken

Capture a screenshot of the stage as well if you know how to do that.

Remember!

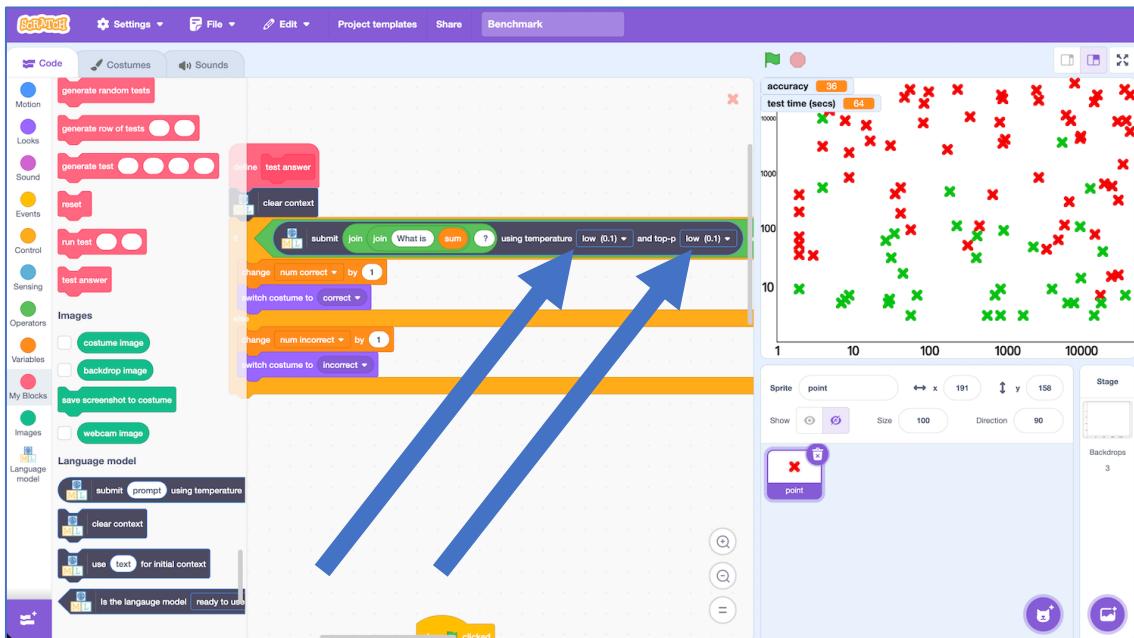
Higher Top-P and Temperature values increase the randomness and result in language models outputting less likely words. This is very useful for generation tasks that value creativity.

Decreasing these values results in more repetitive generation, but this is useful for tasks that prioritise factual output.

When asking a sum, you don't want a creative or unlikely answer.

When asking a sum, you want the most likely, most typical, most common response.

37. Change your language model to use a low temperature and low top-p



38. Click the Green Flag

Run the test again using the low temperature and low top-p values

39. Make a note of the results this time as you did before Record the accuracy, time taken, and if you can, a screenshot

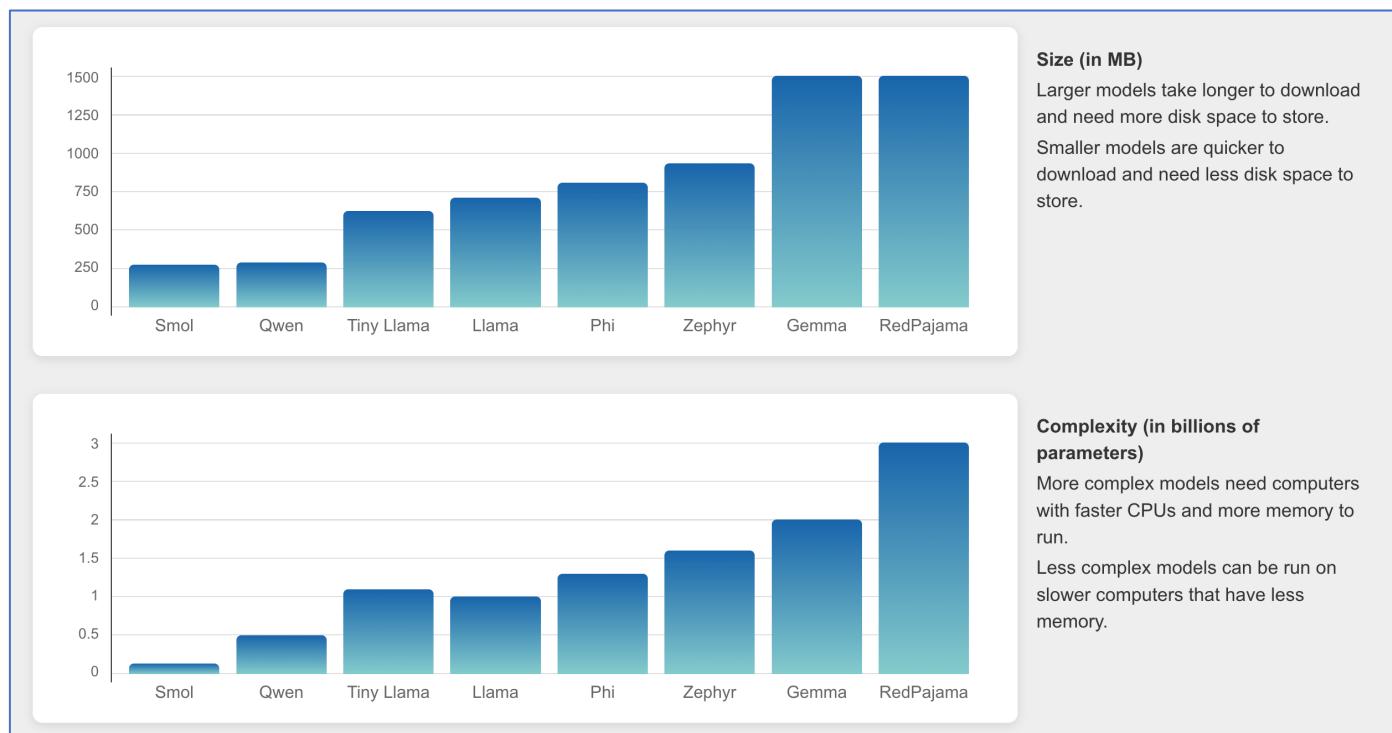
40. Compare your results with the results the other students in your group got for other models

	Smol	Qwen	Tiny Llama	Llama	Phi	Zephyr	Gemma	Red Pajama
Accuracy	Percentage of sums answered correctly. Higher is better.							
high temp & top-p								
low temp & top-p								
Time	Number of seconds to complete the test. Lower is better.							
high temp & top-p								
low temp & top-p								

What have you learned?

Which of the models was most accurate at answering addition questions?

Remember the information that was displayed when you selected your model.



Are the largest and most complex models the best at this job?

What difference do the Temperature and Top-P settings make?

Are the models better at providing correct answers to addition sums with a high or a low temperature and Top-P?

Do the models all take the same amount of time to provide answers?

Do models that take longer to provide answers give more reliable answers?

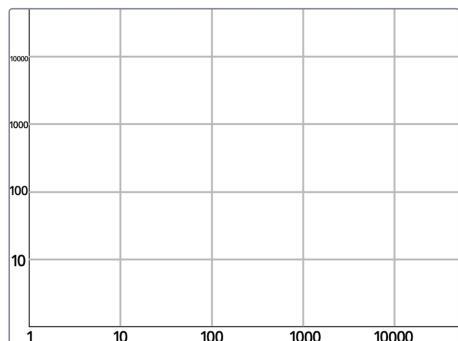
Are the larger models better at answering addition questions?

Are the models as good at answering addition questions with small numbers as they are at answering addition questions with large numbers?

Sums with small numbers (e.g. “ $2 + 3$ ”) will be found in many documents that models might have been trained on.

Sums with large numbers (e.g. “ $9472 + 3168$ ”) are unlikely to have been present in any of the documents that models might have been trained on.

Does this have an impact on any of the models that you have tested?



What pattern of green and red crosses would you expect for a language model that is dependent on the presence of sequences of words in documents they were trained on to predict the next value after a sum, as if it was any other type of sentence?

What pattern of green and red crosses would you expect for a model that has been specifically trained with mathematical content from which they could learn patterns in sums and techniques for solving mathematical equations?

Which model would you pick?

What if cost was the most important consideration, to get “good enough” answers as cheaply as possible? (In this project, you’ve been running the models yourself, but imagine you were paying to use a real LLM model API. More complex models with more parameters are often more expensive to use.)

What if speed was the most important consideration, to get “good enough” answers as quickly as possible?

What if accuracy was the most important consideration, to get the best possible answer, however long it took or how much it cost?

What if you only needed answers to addition sums with small numbers?

What if you needed to choose a model to answer science questions? Or geography questions? Or history questions? Etc.

Would your results from these tests help you to know how the models are likely to be at non-mathematical tasks?

Can you find information about what the models have been trained to be good at?

Can you explain the results that you've seen by reviewing model cards for the models you've used in this project?

It will help you to search for model cards for these models by using their full names and version numbers:

- SmoLLM2
- Qwen 2.5
- Tiny Llama 1.0
- Llama 3.2
- Phi 1.5
- Stable LM 2 Zephyr
- RedPajama INCITE

What have you learned?

What you've been doing is a simplified and shortened version of the work that goes into selecting a language model for an AI project.

Not all models are as good at all tasks, so we test models to see how good they are at what we need. Learn about some of these tests at <https://ibm.biz/benchmark>

Accuracy is not the only consideration. At the time of writing, llm-prices.com has a range of prices from \$0.03 (for a million input tokens with Amazon's Nova) to \$150 (for a million input tokens with OpenAI's o1-Pro). Budget is often a key consideration when selecting a language model.

Ideas and Extensions

Now that you've finished, why not give one of these ideas a try?

Try a different type of question

You could alter the benchmark to use a different type of mathematical question – for example multiplication.

Are language models as good at multiplication as they are at addition?

Try a more reliable way of checking answers

You saw in the worksheet that **contains** can incorrectly check the model answers.

Can you come up with a more reliable way to check?

Try different temperature and Top-P values

Instead of changing both values only between two extremes (high and low), can you improve the model's accuracy by selecting different values?