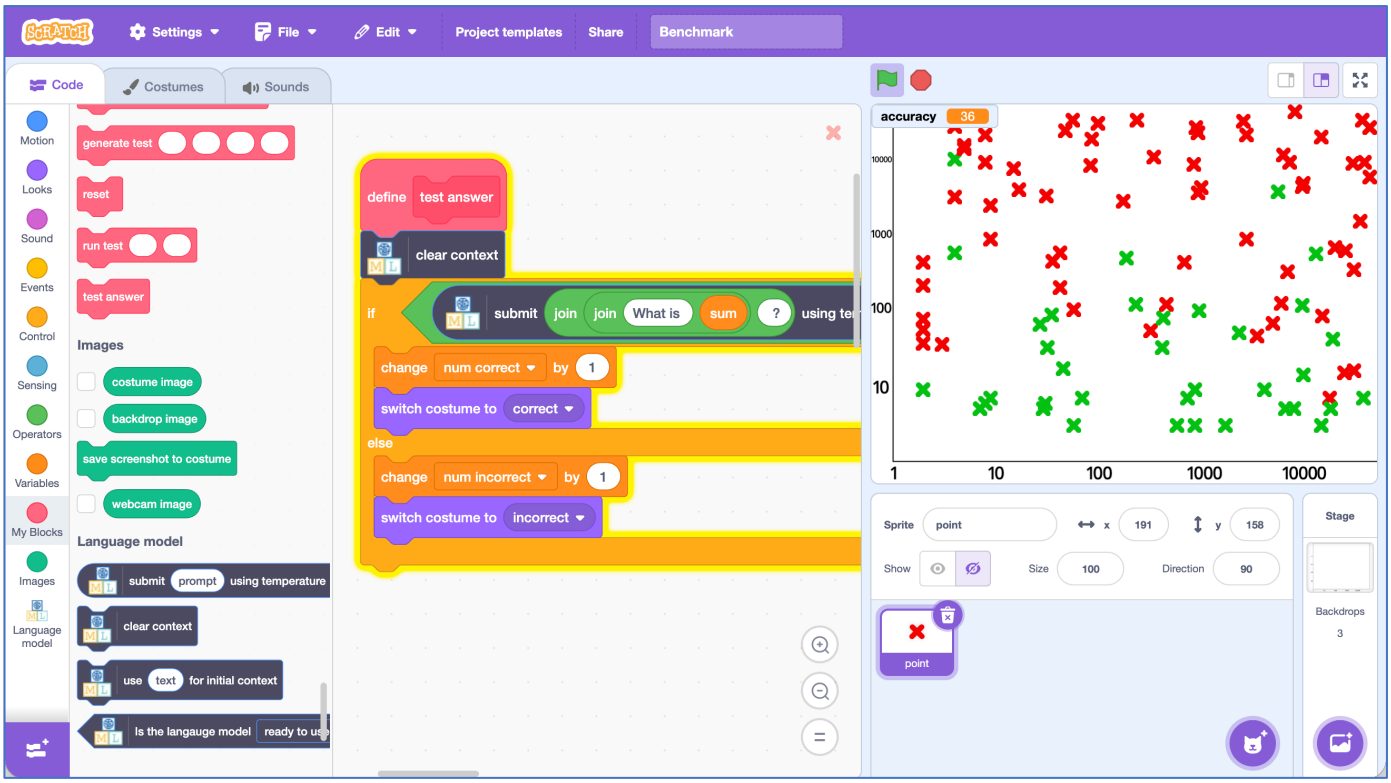
Benchmark

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.

You will understand this project better if you do the "Language Models" worksheet before this one.



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/

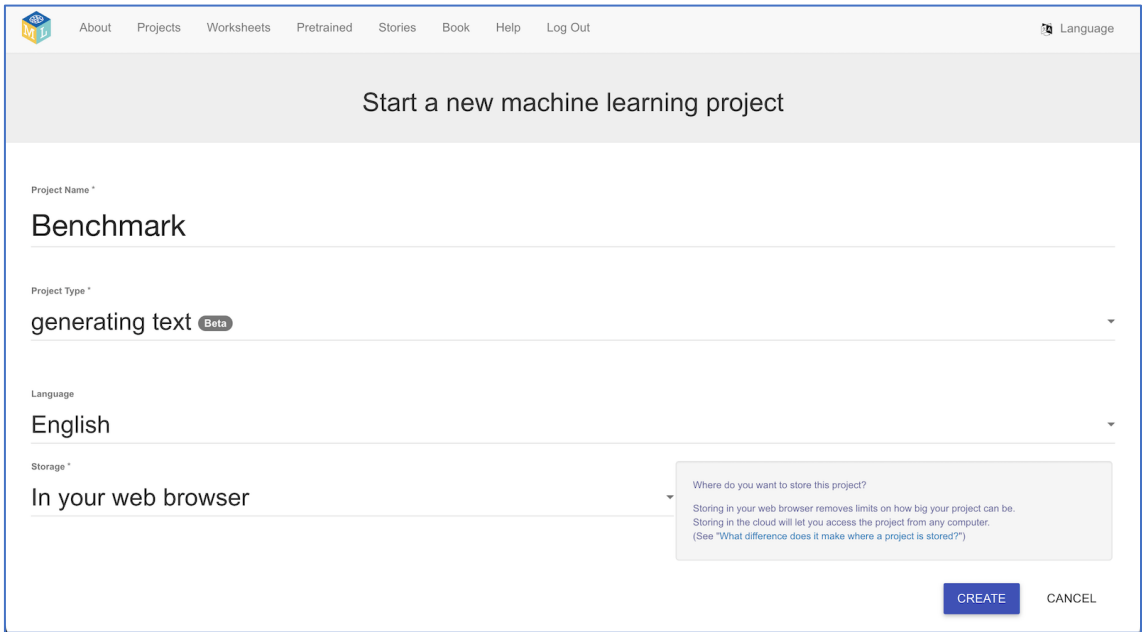
2. Click on "**Get started**"

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

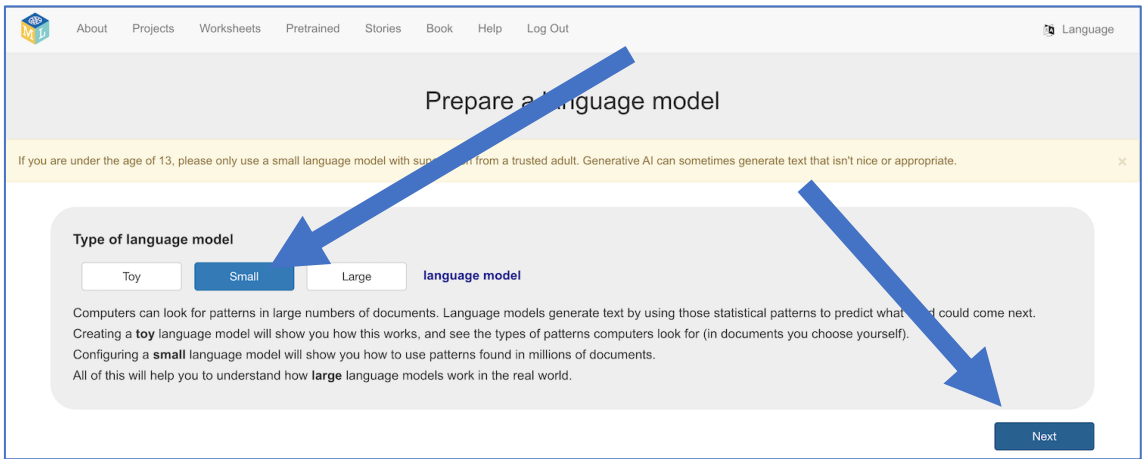
5. Click the **"+ Add a new project**" button.

6. Name your project "**Benchmark**" and set it to generate text.
Click **Create**



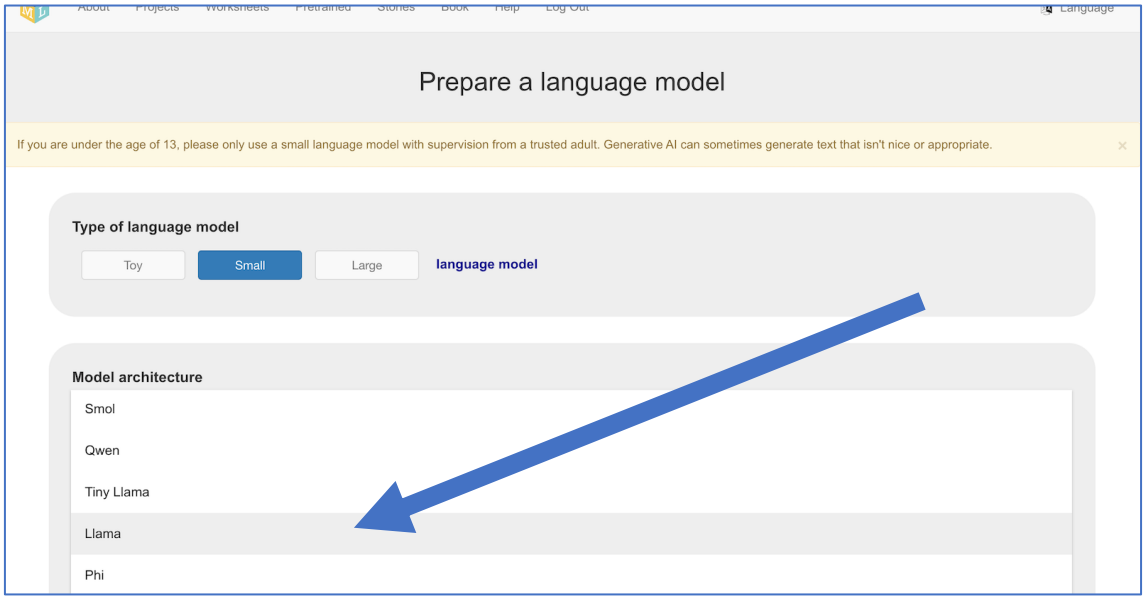
7. You should see your new project in the projects list. Click on it.

8. Click on **Small**, and then click **Next**



9. Choose a model architecture that you will be testing

There are several models for your class to test.
Divide the models between all the students in the class.
Every model will be tested by someone in the class, and then you can compare your results.



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*

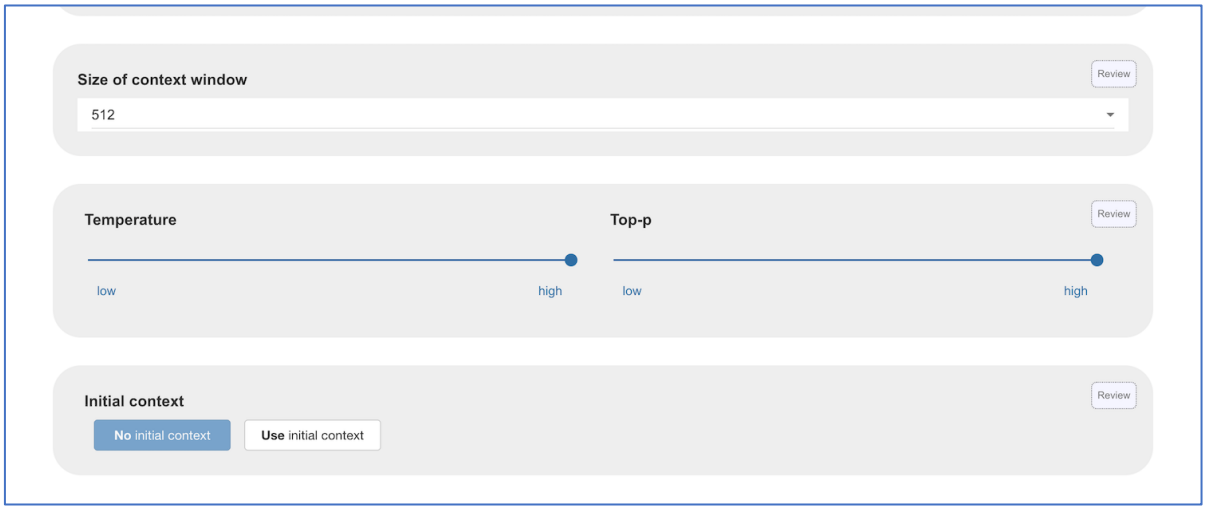
It is okay to leave out the biggest models if the download would take too long.

10. Click **Download**

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

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

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



What are you doing?

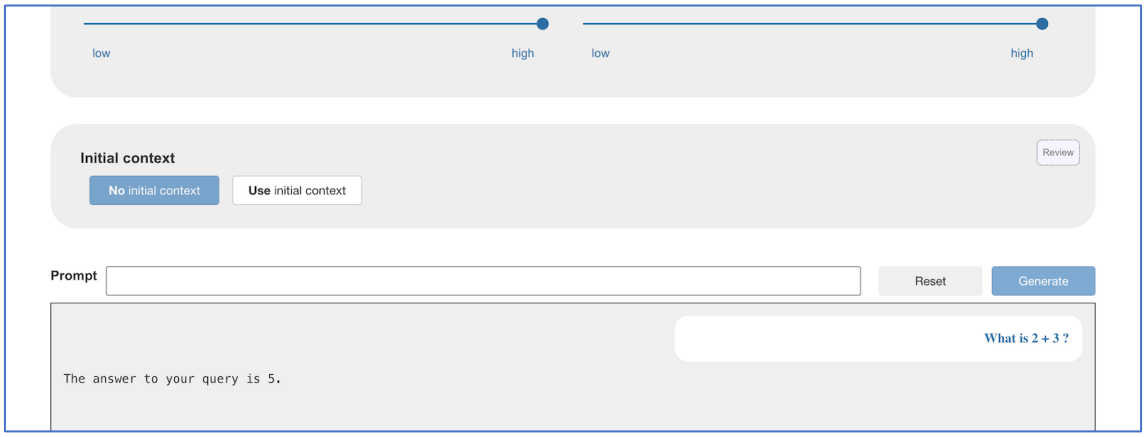
The goal for this project is to find out how good these different language models are.

You have one of the language models to test.

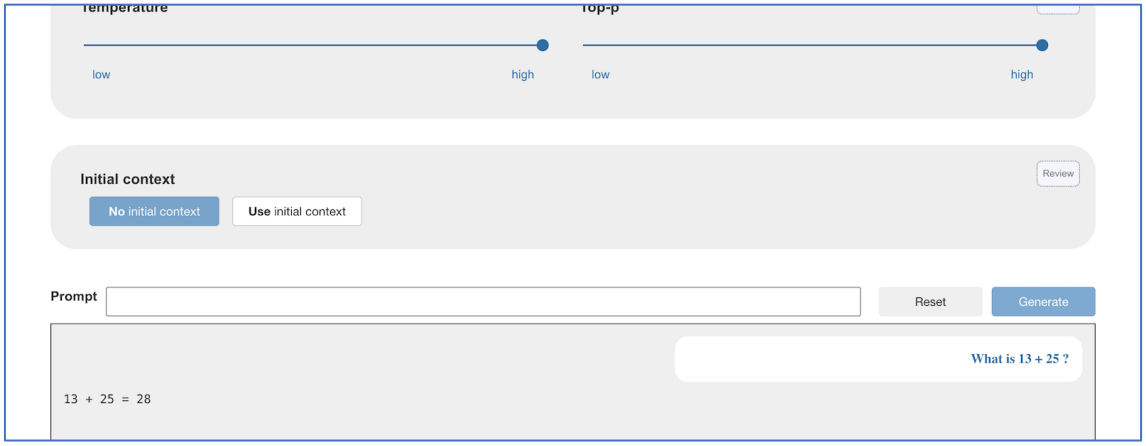
There is rarely such a thing as a "best" or "worst" model. Different models are good at different tasks. When we test models, we test them to see how good they are at the specific job we want them to do.

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

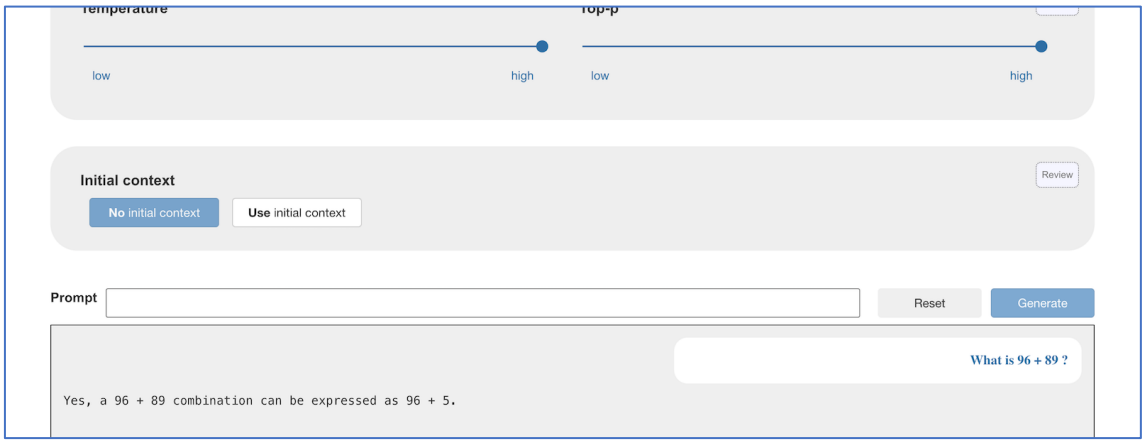
14. Type a prompt asking the model to give the answer to a simple sum *"What is 2 + 3?"*



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



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.



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 a **subjective evaluation** of the models. That is quick and simple, and is a common approach.

But how reliable do you think it is?

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 you 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. But 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 can write code to give a computer instructions for how to quickly perform a task repeatedly. Next, you will write code in Scratch so it can do a lot of testing very quickly.

You need to describe, in code, how to check 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.

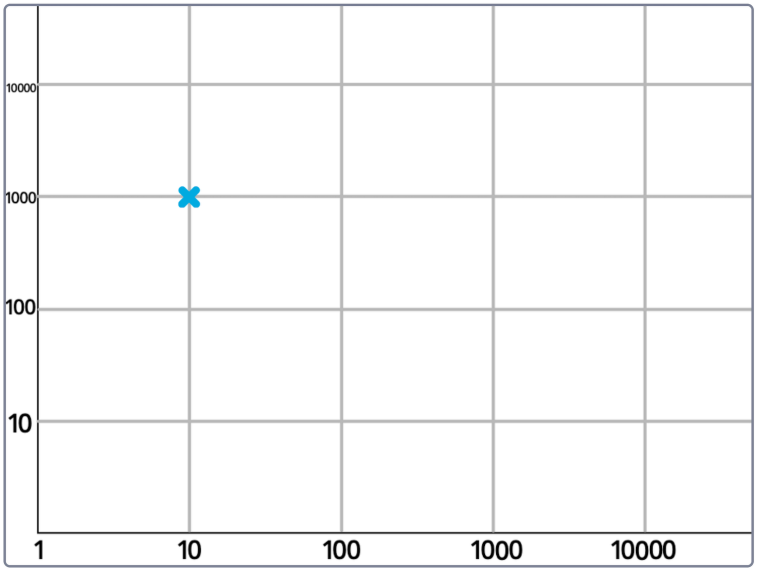
20. Click on the **Scratch 3** button (at the bottom of the page)

21. Click on **Project templates**

22. Open the **Benchmark** template

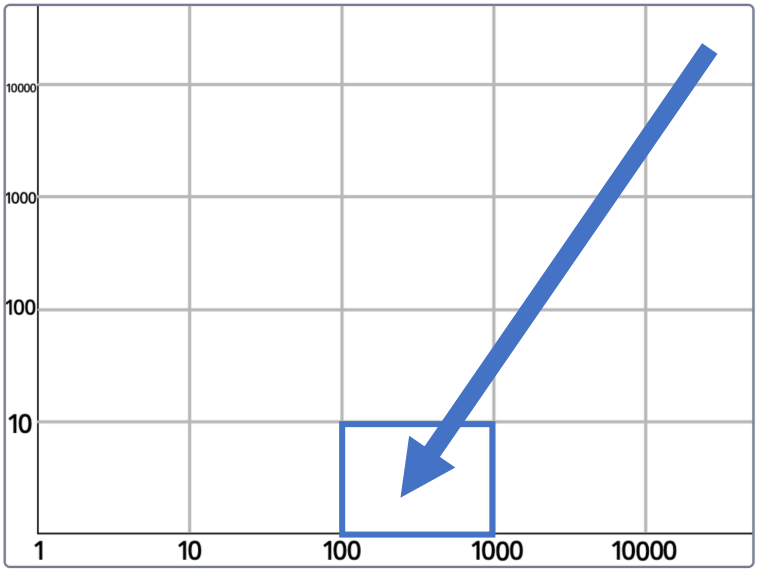
You will use this to plot addition sums on a graph.

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



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

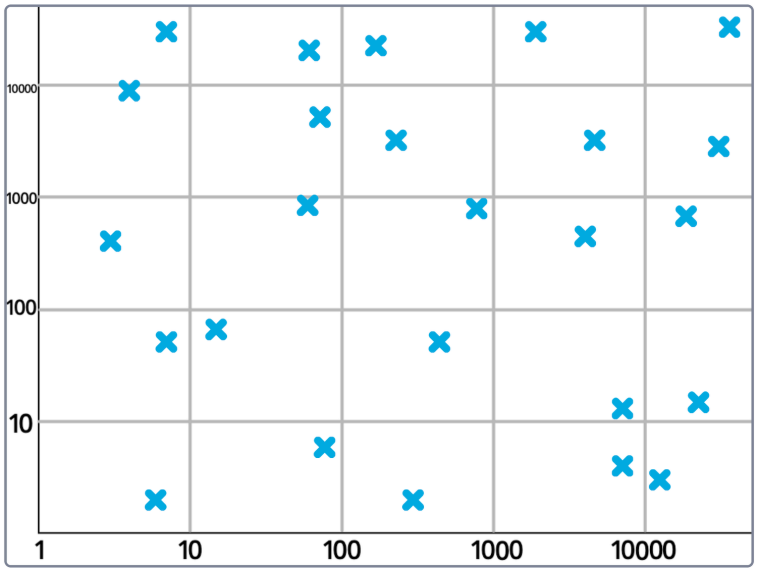
For example, it will choose a number between 100 and 1000



and add it 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:*

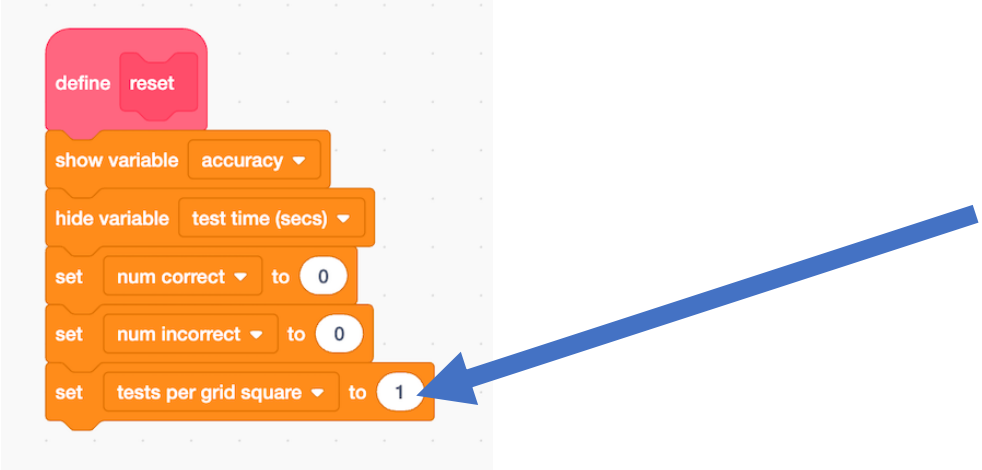


You will update this project so that each of these random sums is asked to your language model. And you will update the code for how it displays results.
If the model's response contains the correct answer,
the sum will be shown as a green cross.
If the model's response does not contain the correct answer,
the sum will be shown as a red cross.

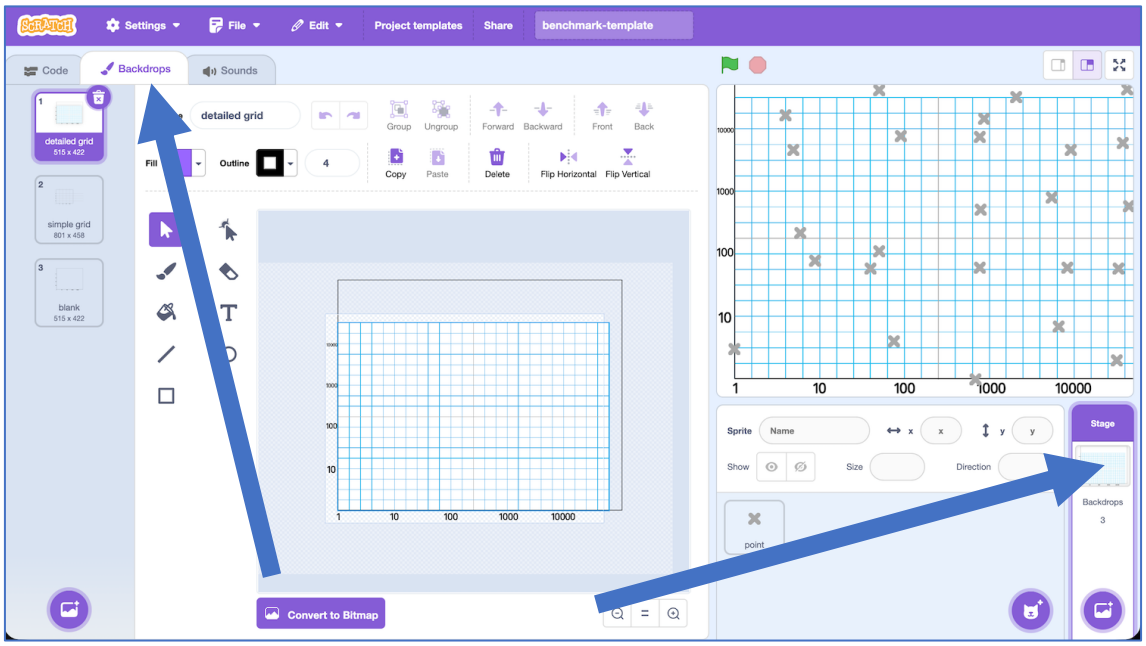
This will show you if your language model is better with larger or smaller numbers.

23. Click on the **Green Flag** a few times
See the sorts of sums that the project can generate

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

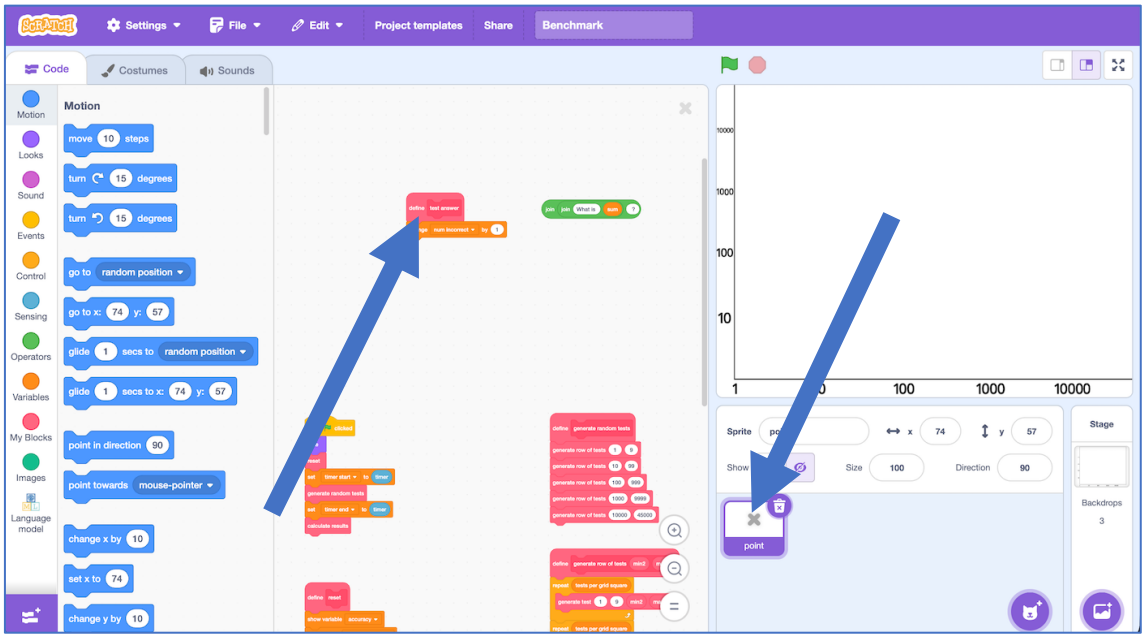


25. Choose the backdrop you would like for the Stage



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 the green **join** blocks.*

Your next step will be to add code that will:

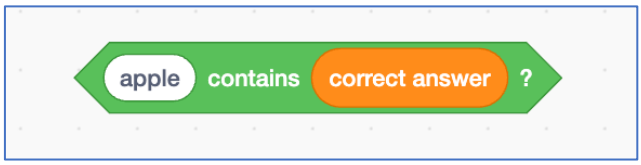
** ask your language model for an answer to the sum*

** 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*

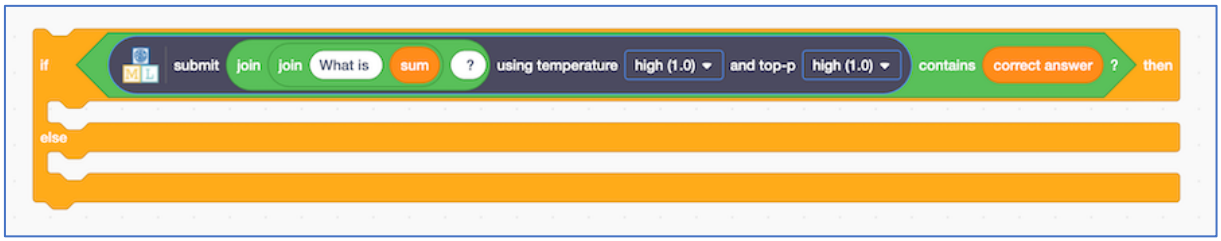
27. Start by preparing the check for the correct answer



28. Add your language model block to the check

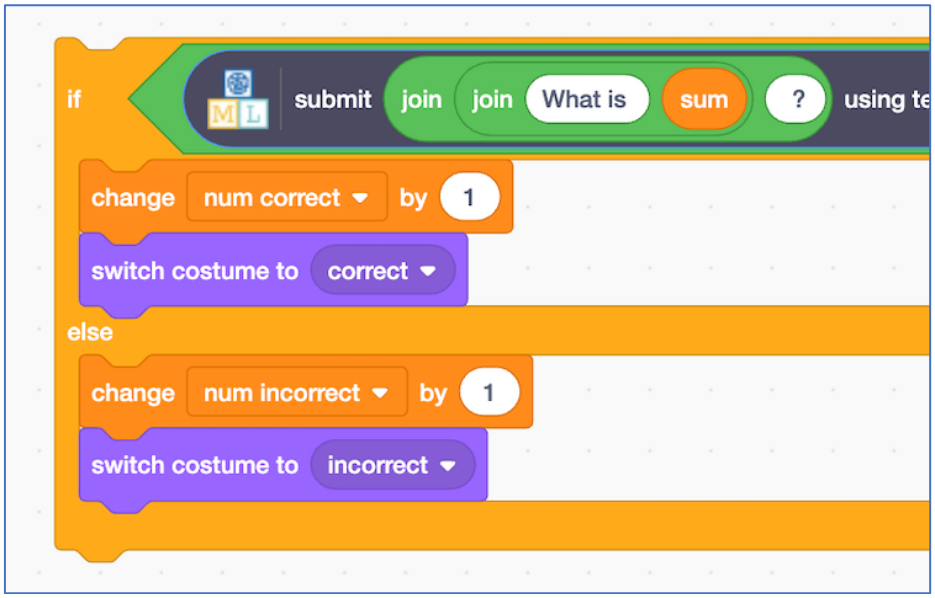
29. Put the sum question into the language model block

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

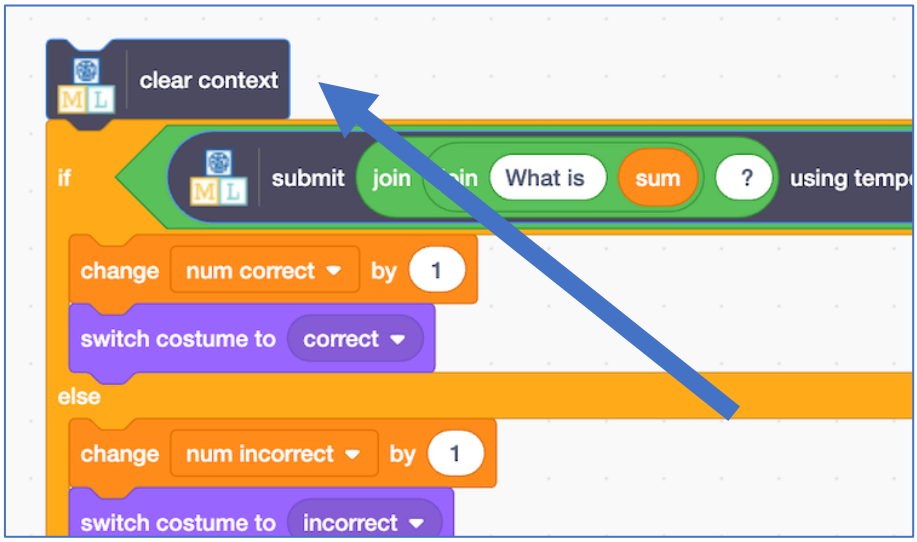


31. Add blocks to change colour and update the counts

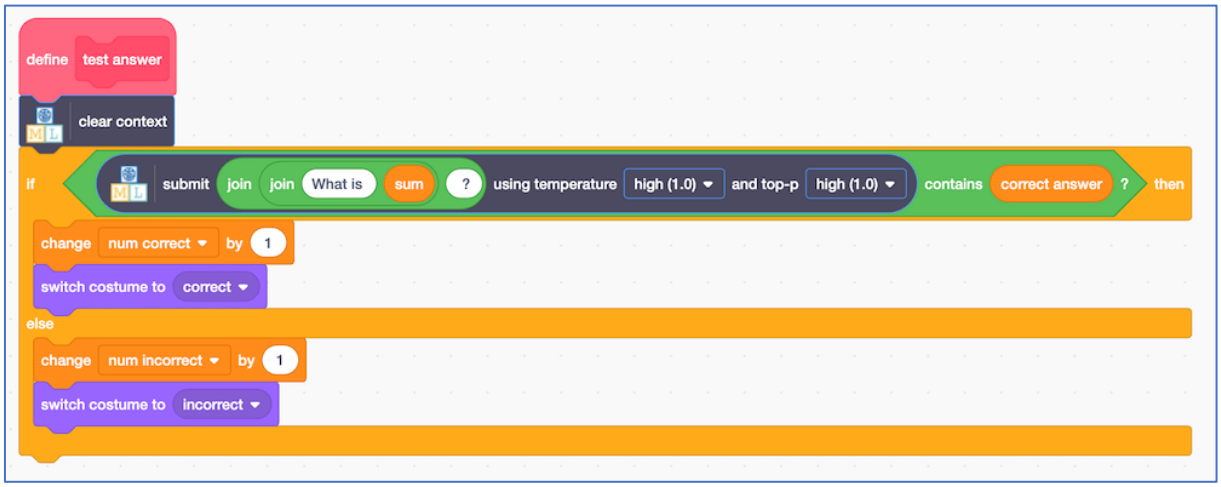
Do this carefully or your test results will not be valid



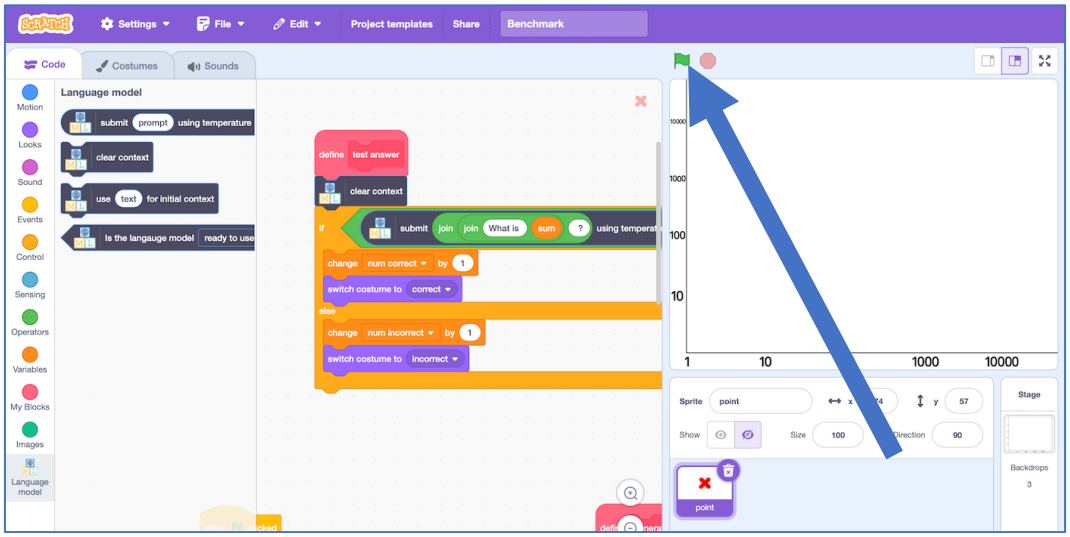
32. Reset the model before each test so the context does not affect the answers that it gives



33. Replace the test answer contents with your completed code

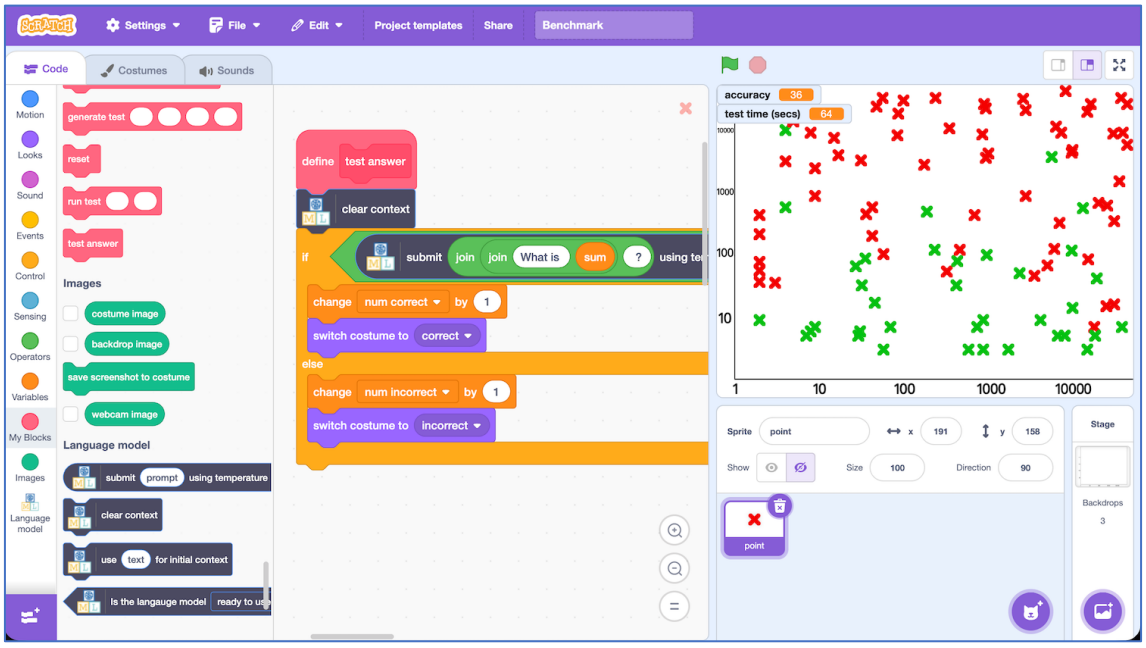


34. Time to test! Click the **Green Flag**



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

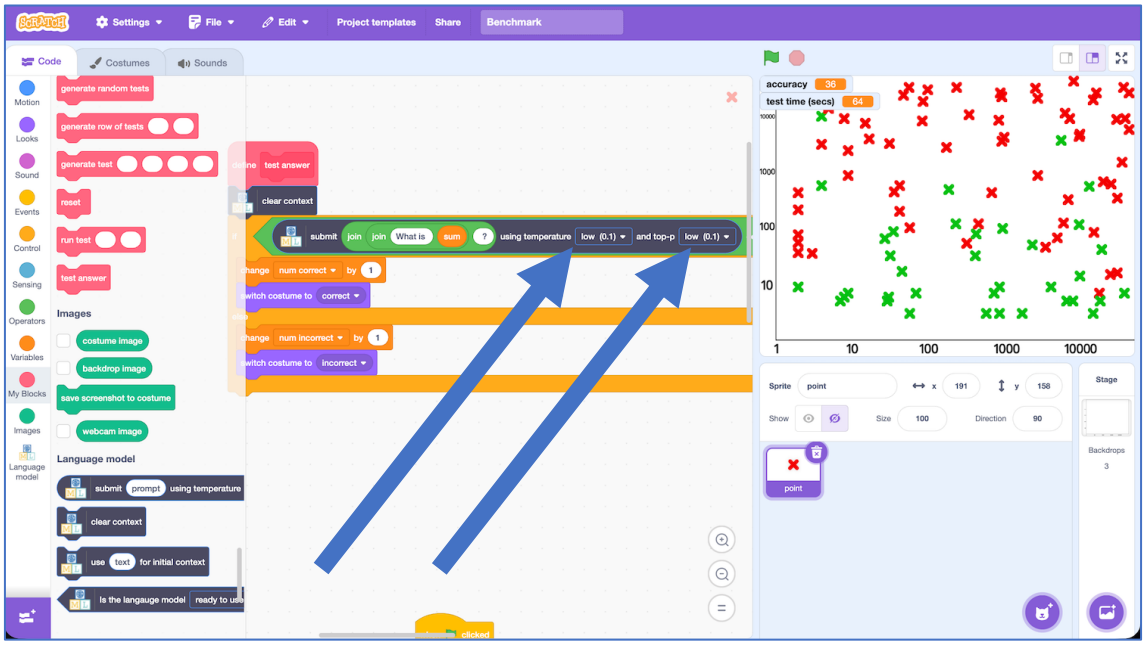
Higher Top-P and Temperature values increase the randomness and result in language models outputting less likely words. This is useful for language generation tasks that need creativity.

Lower Top-P and Temperature values cause 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

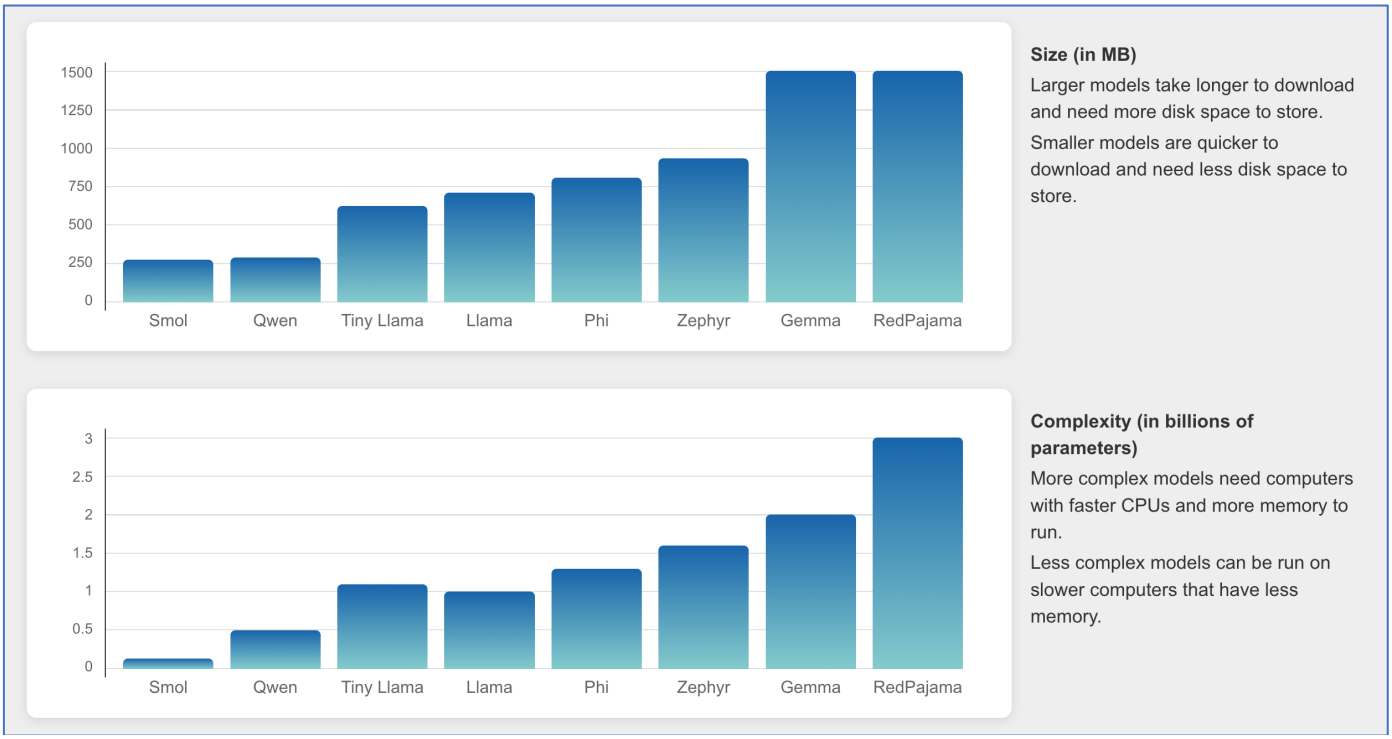
40. Compare your results with the results the other students in your group got for other models

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

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

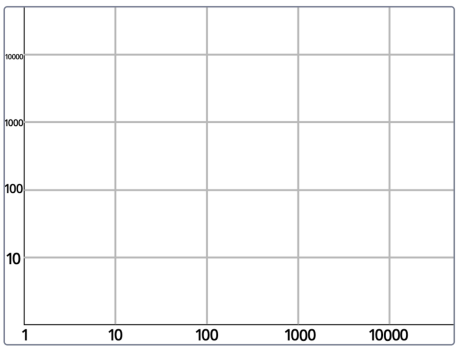
Are larger models always better at answering addition questions?

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.

What impact does this have on the models that you have tested?



What pattern of green and red crosses would you expect for a language model that depends only on 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 a lot of 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**

You've seen that accuracy is not the only consideration. Today, **llm-prices.com** shows 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 for an AI project.

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.

What would be 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?