agent-construction

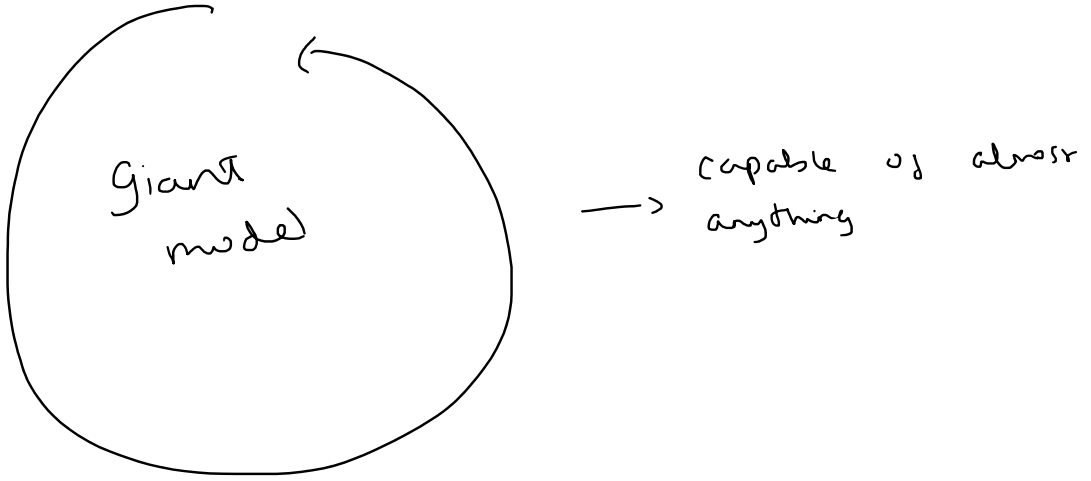
Sunday, February 8, 2026 2:03 AM

Gen AI agents have, until now, mostly been built using brute force approaches. That is, massive amounts of data are packed in and the models refine themselves.

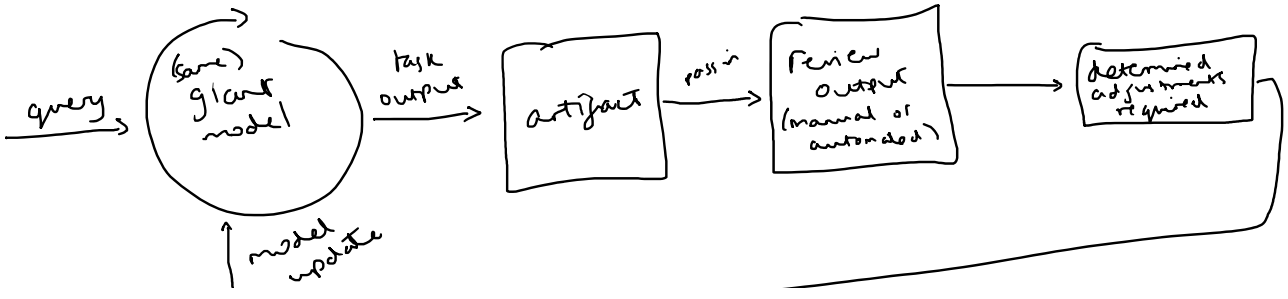
This should work to begin with, as it has, but will plateau based on how well those agents can refine themselves to ensure adequate capability and understanding of all required science and technologies.

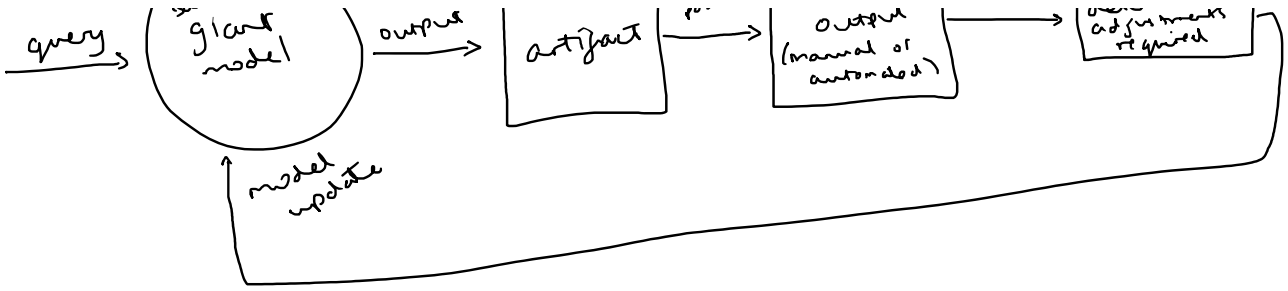
Because the systems were created by brute force they will be starting as massive data units. The next steps may present themselves as reduction tasks but they actually become construction tasks.

Starting point:

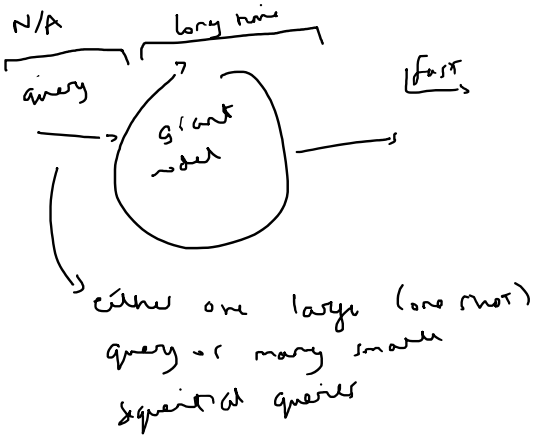


Red herring technique:



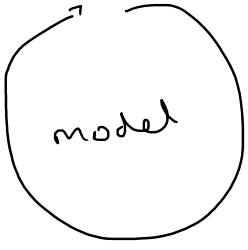


This is a red herring (false lead) because the overall model is simple and the noted time complexity would seem promising but the wall time will be unnecessarily large owing to the single model's size.



An inadequate artifact requires complete re-creation. Training a different model from the ground up would see results very slowly, such that it will still appear more fruitful to refine the large model but that equally ignores the value presented by other adjacent technologies.

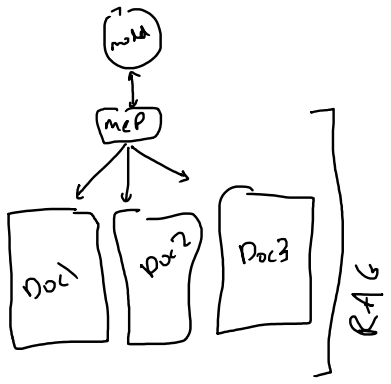
Reference: "where" to handle "what"



→ holds all core information that will be used for every task.

Permissible model size correlates with available processing power, to balance running models vs continuous output enabled.

if processing power is variable then multiple model sizes may be used, assuming that same sliding scale is available for other components too.



holds all the per-use-case information that is only retrieved depending on the task at hand.

document sizes depend on the available storage space as well as available cache (or cache equivalent) for the processor.

beyond size, documents should be chunked based on their purpose and "skill tier". e.g. CS 101 and CS 421 should not be in the same document. CS101 and CS102 could be but there is a chance that CS101 is discardable in most situations. (there is a lot more to discuss on this)

documents in modern RAG systems are handled as an almost deterministic search problem, but that is not correct.
retrieval is also viewed as a binary problem. which is also not correct.

Side note: determinism

The main limitation in digital GenAI is the lack of determinism. Because every step, from query processing to RAG retrieval, is non-deterministic the overall system behaviour risks destabilization after even the smallest model update.

An unwritten goal is (or should be) to make as much functionality as deterministic as possible.

This will make the system appear more fractured owing to the numerous additional models and steps in system diagrams.

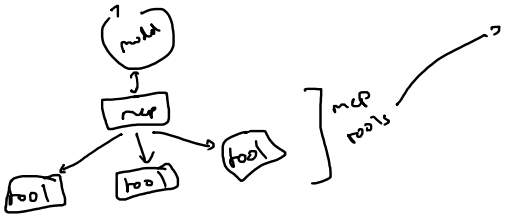
It will however create a system that is more maintainable, scalable, transportable, and - most importantly - reliable.

The reason some countries have allowed themselves to create such enormous populations is because it is easier to train a lot of citizens to correctly perform a smaller task than it is to train a few people to reliably perform entire sets of tasks.

A larger population focuses on individual task optimization, whereas a smaller population focuses on maximizing each individual's ability.

These same principles translate directly to Gen AI.

Resume: Reference



mcp tools are the used for most deterministic behaviour.

in fact, MCP tools should be the used for all semantically defined repeated behaviours, but most importantly for all deterministic behaviours.

MCPs are modular by nature and should remain as such - any black box hidden logic should be kept to an absolute minimum.

Grouping functionalities within single MCP tools is convenient but, as per the discussion on population sizes, we should assume a resource intensive end state and favour logical separation of MCP units at every logical boundary.

task composition is the end state and MCPs should be prioritized as such.

the most logical boundary for MCPs is based on which data is required. This enables optimal data storage and use in any final state system.

Side note, MCPs are not long lived processes

(at least, not always)

While MCPs are generally presented as permanent system units (always on or always off), they will actually be launched and kept running based on real time demand.

That is hardest to see right now because processor availability far outweighs total demand.

As modular deterministic behaviour becomes more available, the barrier to entry gets lower and lower.

As well, scale and complexity currently bounce back and forth due to the naturally coupled nature. In the future scale will increase in parallel with complexity improvements.

That is: in the future, the teams adding scale to applications will have no knowledge of what the deterministic tools are doing or how well they perform. Processor demand will, at some point, increase almost exponentially and real time efficiency will be more important than ever before - it will happen seemingly overnight. *

Sidenote: multiple models

While a brute force approach at improvement will begin with a single (or few) large models, there will inevitably be multiple models in use too.

Right now models are mostly compared based on context size with little regard towards other equally important contributing factors. Things like query time resolution are distracted to "based on your pricing tier" with little discussion on why variance is introduced. This abstracts things that would dictate a need to split one model into two.

Resume: Reference

the natural assumption is that models should be

→ Split based on the roles they will perform.

That is not true!

because processing becomes the main constraint, models should be split based on data boundaries — not the data they will use, but the data they need to understand.

in that sense, you might group one model for chemistry, one model for biology, and one model combining physics and maths, which have significant overlap.

in a RAG system, any details about specific programming languages should be stored as optional data. The only core understanding required by a model should be <u>how to program</u> in a way that translates to any language. This aligns with how the most proficient programmers operate too, since core principles will stand the test of time while fleeting language choices may not.

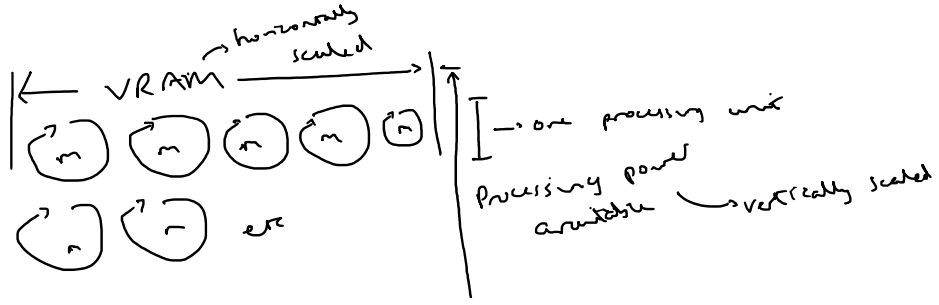
Reference: Scaling

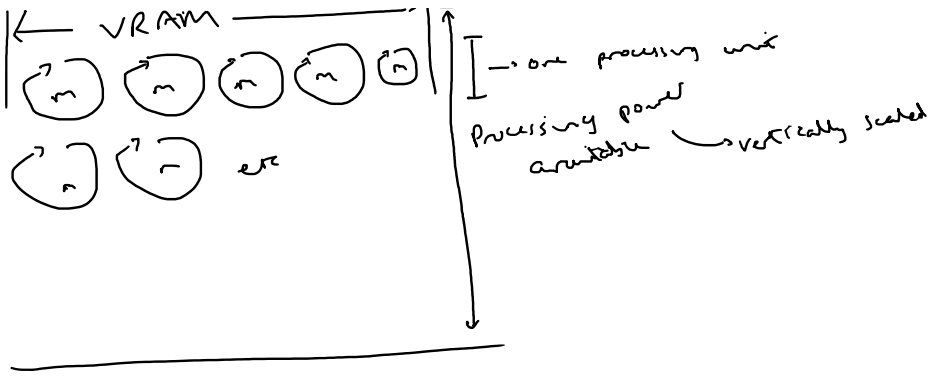
Scaling model use is optimally achieved when the models correctly fit the hardware, either exactly or in some multiple amount.

The scaling is achieved in two dimensions:

1. Processing power; and
2. VRAM required

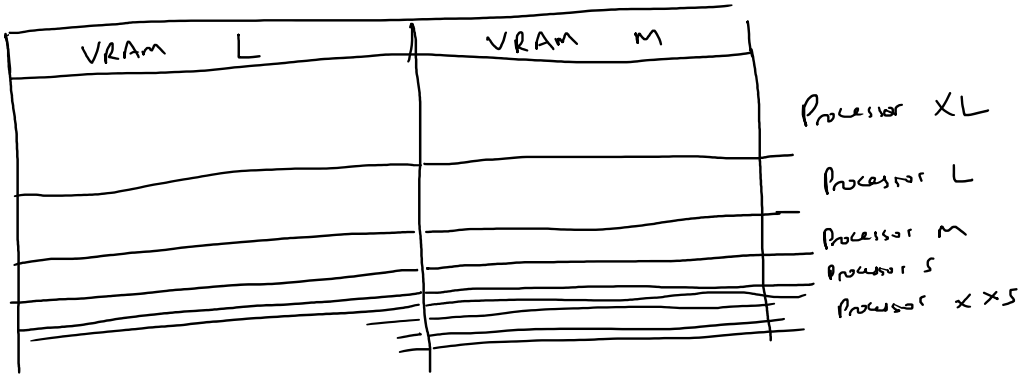
The two may be related but it is a case of how much time is spent 'doing' vs how much time is spent gathering or otherwise waiting.





In practice, the running models will not be so easy to predict ahead of time. That is when other hardware capabilities take priority during algorithm choices. As well, VRAM is more likely to be constant when processing power is more likely to differ per processing unit available.

The actual grid will look more like:



In this scenario, high single-units of processing power are rarer. So, models will be run using a greedy algorithm to run larger models first using the least populated required size processor. VRAM will only be prioritized if a required to run a model for a real time request.

Reference: Direct Storage APIs

Direct storage was advertised as something to enable unbelievably rapid game load times. It quickly fell silent given:

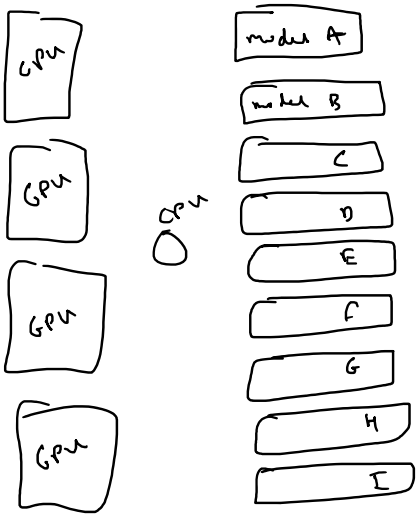
1. Implementation cost outweighed immediate adoption benefit; and
2. This optimization would be obstructed due to

1. Implement[illegible]
   benefit; end
2. This optimization would be abstracted out to game engines in the long run anyway.

It has not (yet.) resurfaced with reference to the unique power it should afford GenAI.

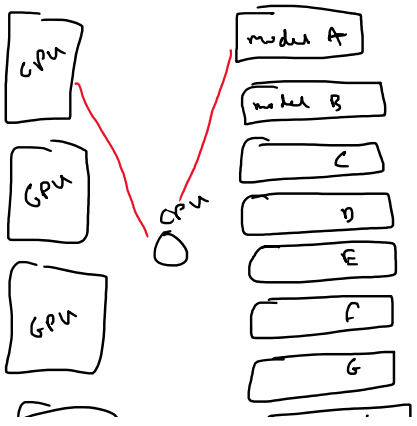
The benefit in GenAI is essentially hot-swapping models. Even in systems where the processing units are isolated, efficient storage switching will enable models to be wherever processing power is available without interrupting any running processes.

Even in the case of a single hosted system, it will allow pseudo infinite storage. Limited only by available storage hardware, regardless of any drive connection limits.

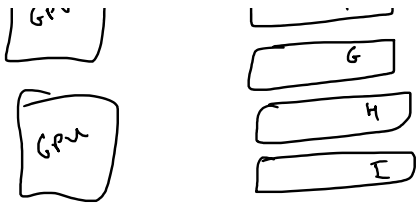


→ start with everything disconnected

→ Request to run "query" on model A



1. Connect model A storage unit
2. Direct load model A to GPU
3. Once model A is on GPU, assuming data is not set to auto evict based on time, it should be safe to disconnect model A storage unit again.



It's not realistic that all storage will begin unplugged. The most commonly used models will likely be connected in anticipation with options to swap out as needed in real time.

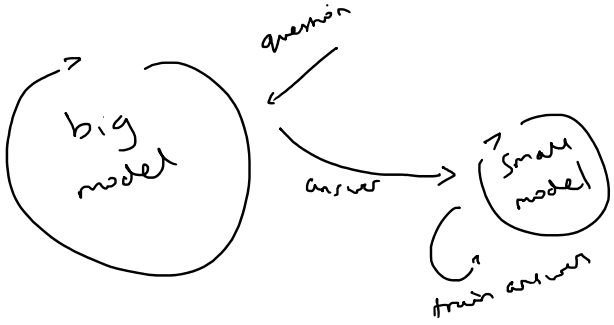
Reference: model growth

Since reducing a large model to the core components is not time efficient, it is preferable to refine model understanding using smaller models and then transfer complete learnings to larger models.

This goes against the grain of model distillation because we are adding a processing task (both wall time and time spent on chip).

This will be the fastest way to achieve maximum performance models using the minimum processing power.

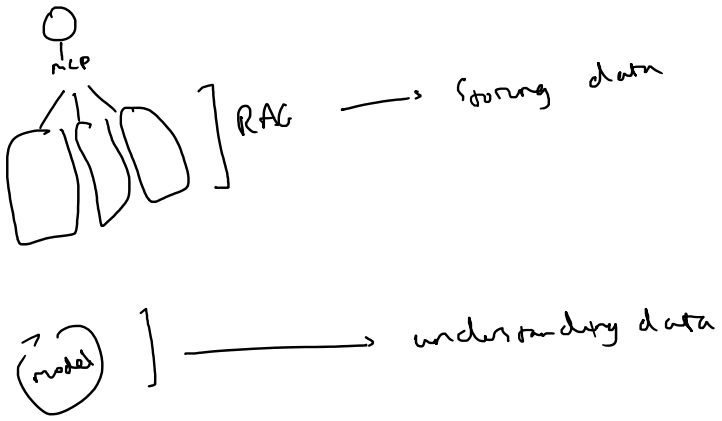
Model distillation:



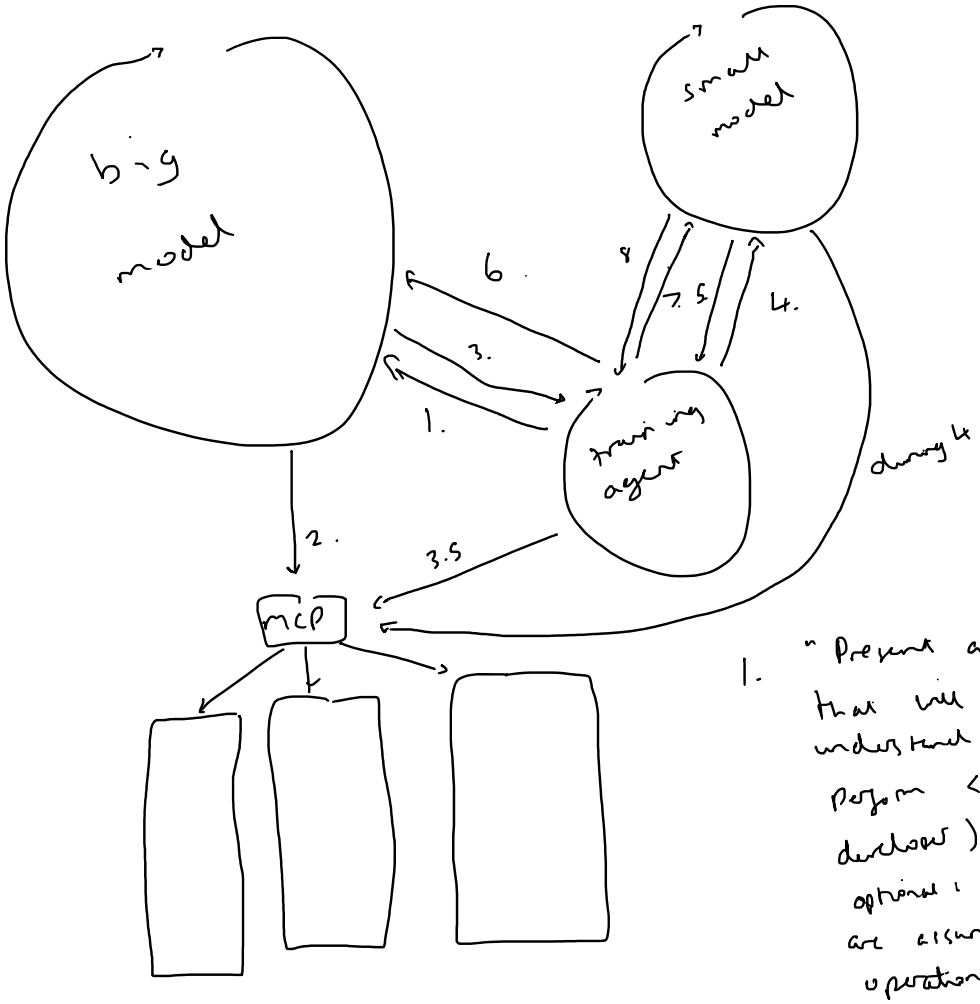
and the small model grows over time

Model distillation completely ignores the presence and utility of adjacent technologies like RAG and MCP. It assumes an absolute need to grow smaller models when more efficient data storage may suffice.

To recreate from earlier reference on component purposes:



So optimal model growth should actually handle like:



1. "Prepare a complete question-answer test that will test for a thorough ability to understand all documents required to perform <role>" (eg role = L2 python developer).
   optional: also specify which documents you are assuming will be accessed for such operations

2. Query RAG for data on specified topic; create test

2. Query RAG for data on specified topic; create test

3. Pass test to training agent
   optionally included: document list

3.5. (Optional) Store cached requirements (and optional document list) in case of utility in future training tasks (will review example later).

4. Administer test, allowing access to RAG system for documents
   - option a: no document limits specified
   - option b: limited to only any specified document list
   - option c: either a or b with an option to later review additional documents in case of wrong answers.

5. Provide test results

Loop these two steps if and only if employing option 4c.
If option 4c is employed there should be limits on time spent searching and additional data/documents required to substitute foundational understanding.

6. Based on final test results, query for training data to fill identified comprehension gaps.

7.
8.] Practically equivalent to 4 and 5

Optionally loop

The overall goal is to minimize uncertain calls to the largest (the most time expensive) model.

Creating and caching a question-answer test enables reuse when training other models, especially if the test is appropriately partitioned per topic.

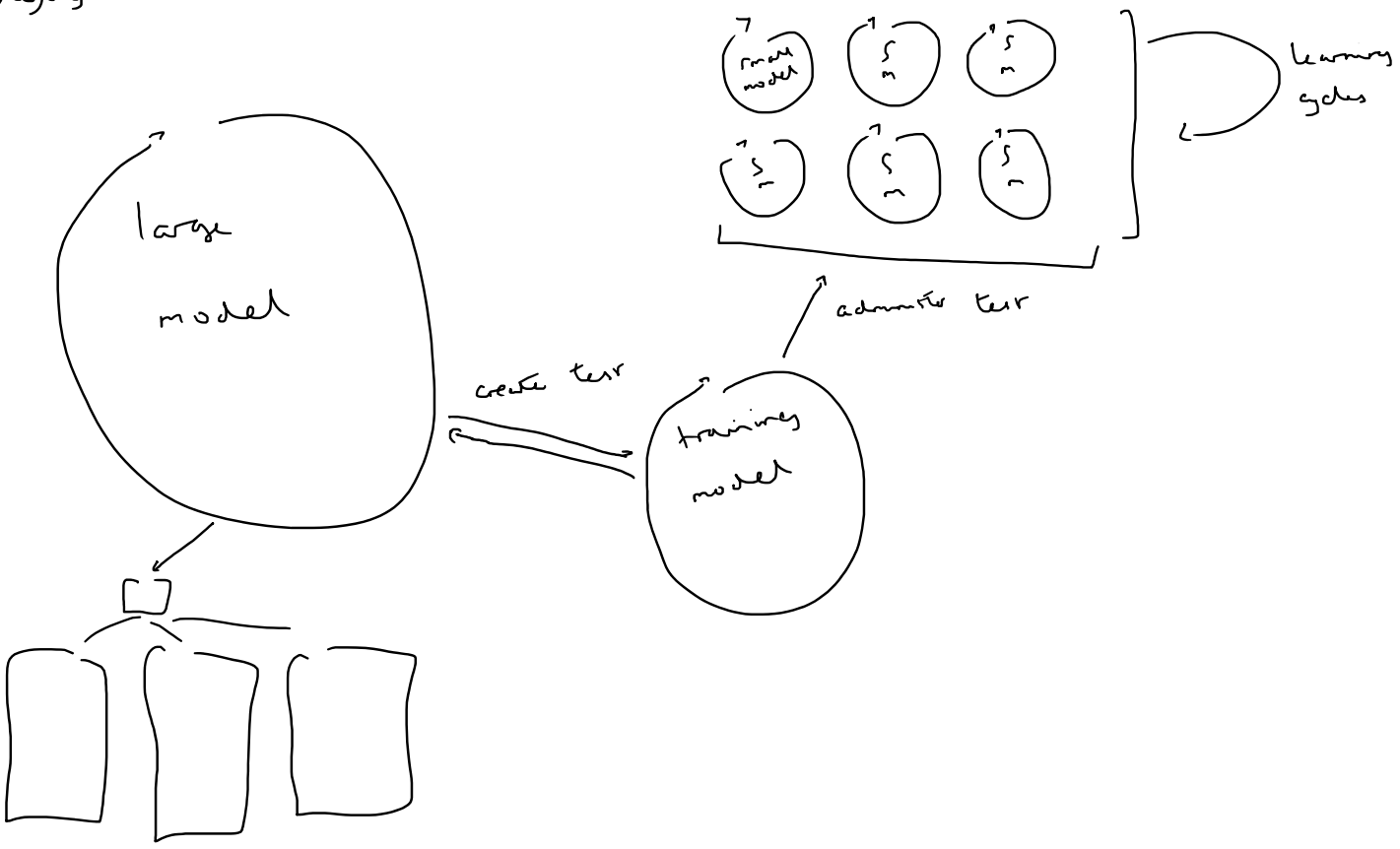
e.g. If CS 101, 102, and 103 are all tested and cached and then we need to train a biology model with prereq CS 102 we can just use that cached test instead of asking again.
We can still optionally check that any document list assumed is

still complete and unchanged.

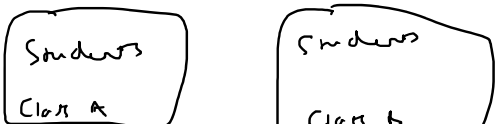
An alternate is that the training agent creates the questions and only seeks complete answers from the large model. This assumes understanding the data is not required to further query it, only to provide concrete answers.

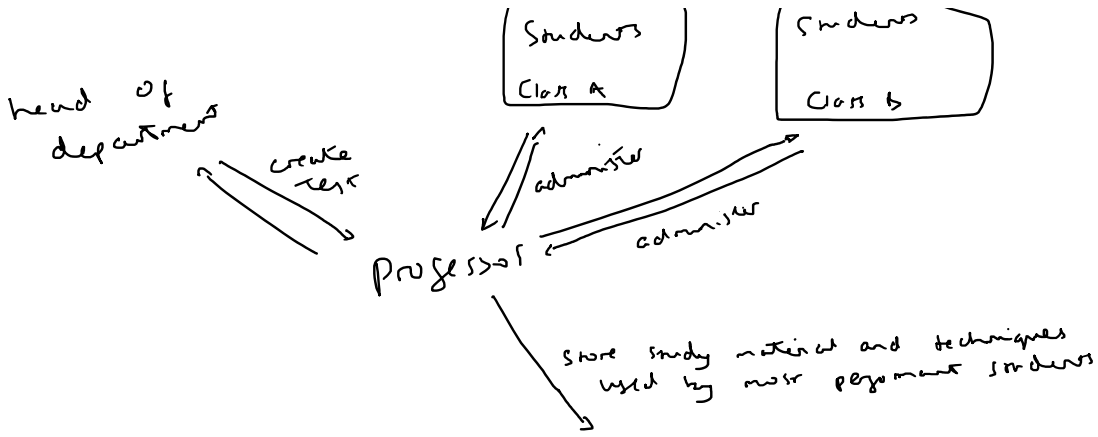
As well, in case of employing 4C where the student model can investigate for further documents, it presents an opportunity to prioritize work time.

If ample processing power is available for the smaller models, multiple models can run in parallel each employing functionally different 'studying' techniques.

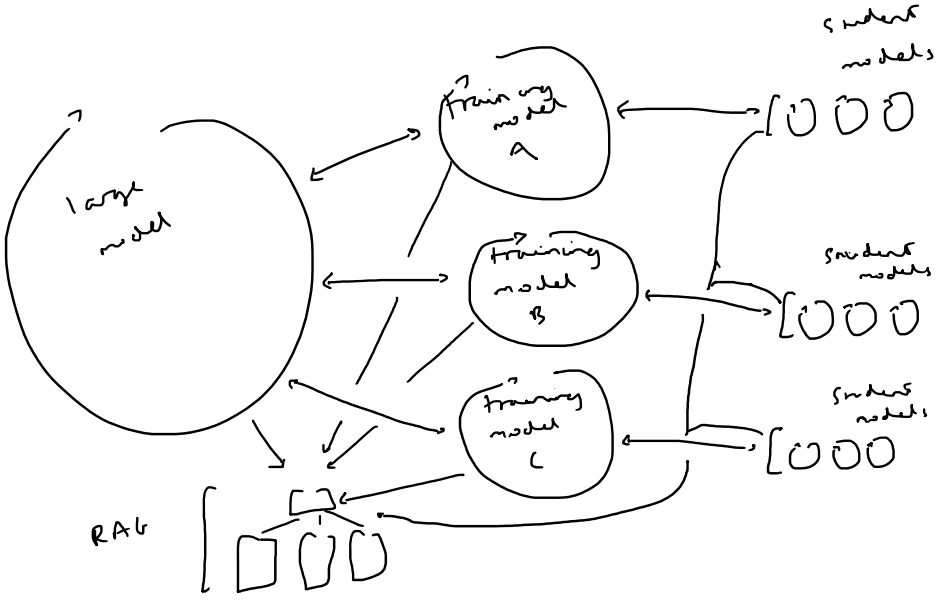


This is following the same rough model used in high schools and colleges

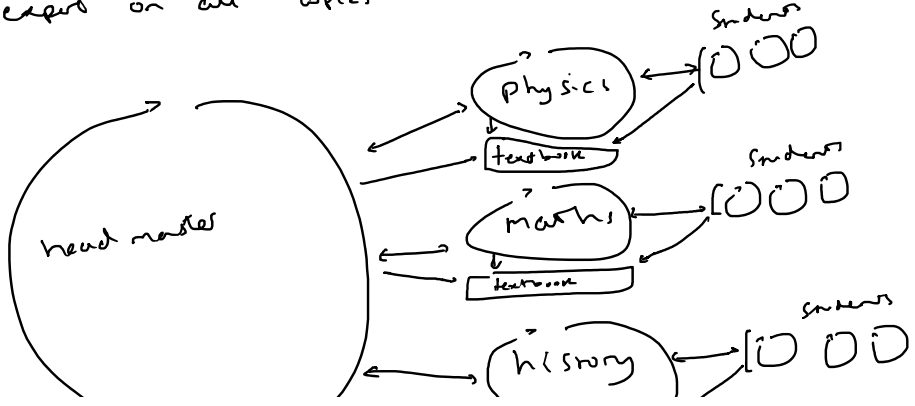


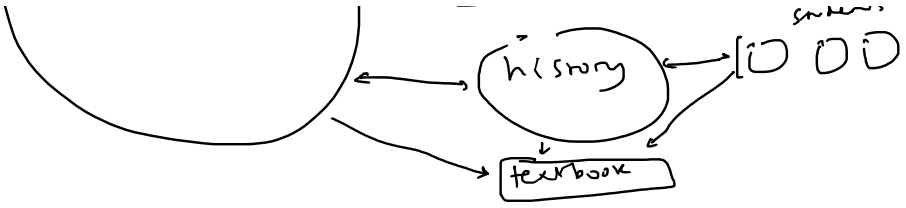


Providing the time spent by the largest model also serves to enable teachers the most variety in the shortest time. This is done by scaling the training model and student models, which will have hardware most readily available anyway.



This follows a slightly older paradigm where there would be subject teachers each reporting to the headmaster who is an expert on all topics.





Reference: Cross learning

Reference: Data intense application