Language models, D, and so on

Max Haughton

Survey

- Do you (consciously) use some kind of “AI”?
- Pay for it? Have an API key?

Some terms

- LLM - large language model, big blob of maths to predict what comes next in a sequence.
- Somewhere between a USB stick and a hard drives worth of numbers.
- Talk isn’t about their design, rather what can we get from them

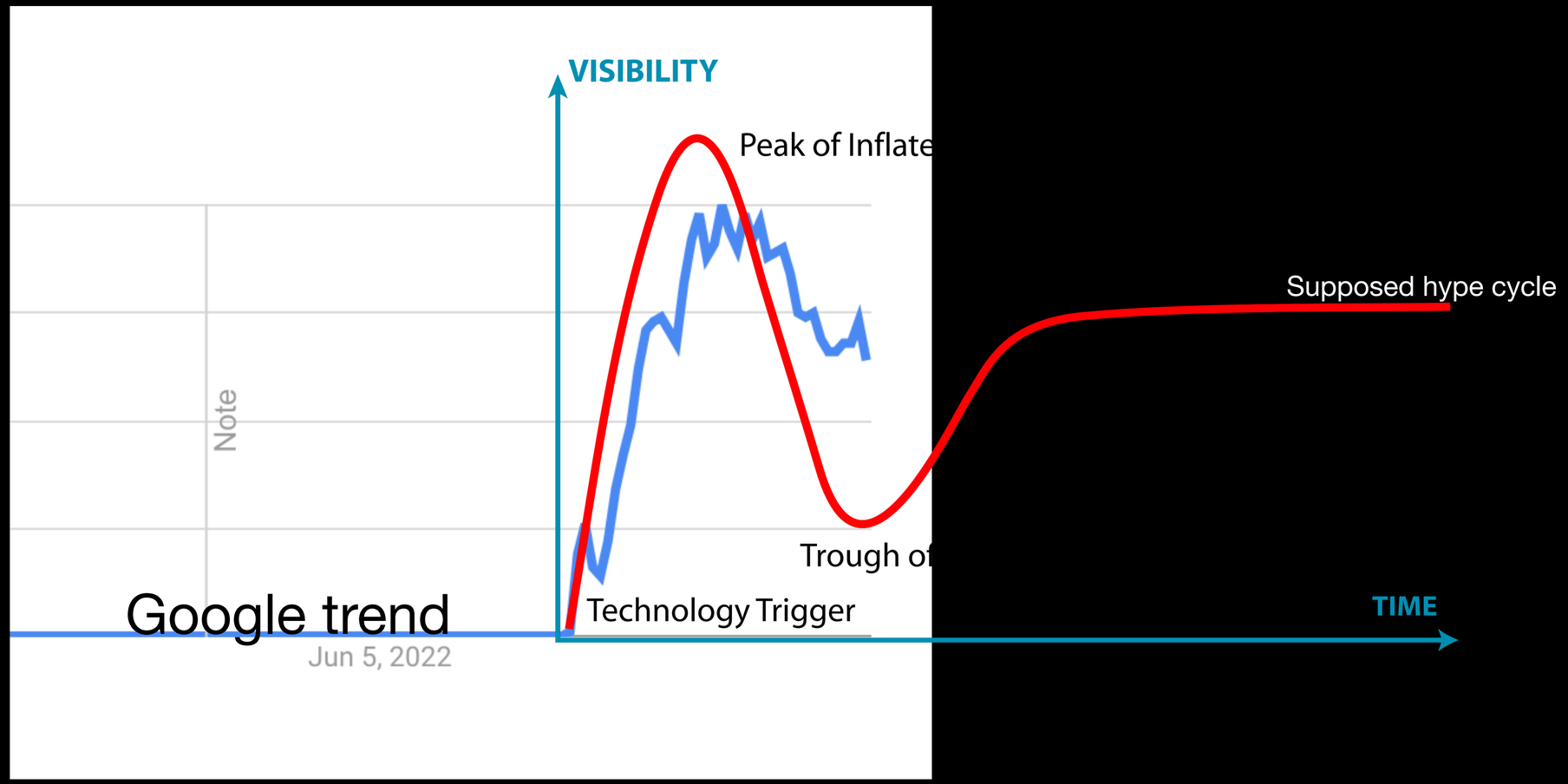
Framing – what do we care about?

- Are we AI researchers? I’m not
- Does it matter if they can actually think or not? Compression enough to be useful?
- (Does that question even make sense)

Maybe not for us?

- Nice for programmers but transformative for non-programmers.
- I got this wrong when I proposed this talk in its original form
- The most interesting uses don’t seem to fit cleanly into traditional “programmer” dominated jobs.
- Obviously popular with young people doing/~~cheating~~ on their schoolwork

Gartner Hype or just school holidays?



What do we want, what can we do

- Focus on things we might integrate into a program rather than what we might interact with e.g. not solely helping us write code.
- Lots of difficult tasks that are *almost* very possible.

Models we can use

- OpenAI - GPT3, GPT3.5, GPT4 (The famous ones)
- Open source models – still a bit crap. Good at talking (this is not useless), not so much at tasks.
- CodeLlama recently released, still takes a lot of local computing power to get anywhere near something fun.

Specific example:
C preprocessor macros

- Perpetual issue with tools - enums are easy, weird macros are not.
- GPT4 gets very close, not quite enough to trust to run in (say) a build system.

```
1   #define SET_BY_NAME(obj, field, set) obj.field = set
2
3   struct X {
4       int var;
5   };
6
7
8   void doesSomething(X& hello) {
9       SET_BY_NAME(hello, var, 1);
10  }
```

It (GPT4) gets very close

- Knows D better than you might expect, but needs help.
- Sometimes perfect, sometimes weirdly complicated.
- Requires feedback – ideally human, but compiler error messages are usually good enough. With feedback, generates a sensible string mixin

```
    static if (hasMember!(T, field)) {
        obj.tupleof[AliasSeq!(__traits(allMembers, T)).find!(x => x
== field)] = set;
    } else {
        static assert(0, T.stringof ~ " does not have a member named
" ~ field);
    }
```

What to do?

- Model output will improve, could give up and wait.
- Maybe dstep/dpp etc. process should be more interactive
- Subtly required anyway because this API is very much not free.
- Driverless cars – you could make the car better or change the road network.

Suppose you do have a model to run locally

- `import torch` takes 1s on my laptop.
- llama.cpp vibe shift: Extremely complicated big-tech-approach python stacks are replaced by a simple C library that covers most models people are interested in (llama, llama derivatives)
- Easy to access via D's C/C++ interop.
- Simple enough - bindings already exist, ImportC works in theory except for fp16 detection.

New dog, old tricks

- Recent trends in AI, or rather executing models once trained, start to look a lot like a compiler.
- Data flowing through a graph, sound familiar?
- Compiler optimisations in pytorch
- I think D can do extremely well here.

```
/**
 * Common Subexpression Elimination
 *
 * This transforms looks for specific operators (denoted by allowed_ops_),
 * and removes unnecessary repetition of that operator.
 *
 * Consider some operator of X, that reads from blob b_ written to by W.
 * X_a and X_b read the output of X. However, another operator Y, is the same
 * type as X, has the same arguments as X, and reads from the same input b_,
 * written to by W. It's output is the same as X. Y_a, Y_b, and Y_c read from Y.
 *
 * Then, we can eliminate the common subexpressions X and Y, and merge them to
 * Z, where X_a, X_b, Y_a, Y_b, and Y_c all read from Z.
 *
 *
 * TODO(benz): Fix the error to not match nodes that write to external output.
 */
class TORCH_API CommonSubexpressionEliminationTransform : public Transform {
```

- Types of problems now coming up (going fast, keep the problem small, nice code etc.) are things D is good at.
- Some prior art, libraries, these can be learnt from.
- Next DConf online perhaps.