Pondering Invariant Properties of Self-Modifying Systems

Michael Bukatin

February 8, 2024

The studies of invariant properties of self-modifying systems is a subject which is quite neglected and which is likely to become fairly important in the near future.

It is likely that the ability to establish at least some approximately invariant properties of self-modifying systems will be important for any hopes to maintain any specific safety properties of the rapidly evolving AI ecosystem and of the world containing this ecosystem.

One conjecture I would like to ponder is that it is likely to be easier to maintain invariants in a situation where changes are gradual and continuous (especially, if the self-modifying ecosystem can control the rate at which the potentially most disruptive changes are phased in).

Let's ponder some possible technical directions which might allow us to make progress here.

1 Introduction

Currently, the self-improvement processes in self-modifying systems tend to saturate before long, and the dynamic systems involving self-modifications tend to fail to unlock true open-endedness.

However, this state of affairs might not continue. When we have the ability to generate *diverse AI equivalents of top software engineers and leading AI researchers*[1], and when the communities of such entities pursue various activities including *research, design, and development of further modified and improved artificial software engineers and artificial AI researchers*, the current limitations of self-improvement are unlikely to persist.

A variety of other development trajectories which don't involve direct creation of conventional artificial software engineers and conventional artificial AI researchers are also likely to lead to self-improvement which does not saturate and to self-modifications which are reasonably close to being fully open-ended.

We would like these kinds of ecosystems[2] to be able to *collectively determine and reliably maintain with a reasonable degree of approximation certain invariant properties*, such that *adequately taking into account interests of various parties and entities*[3] *would follow from those invariant properties being approximately satisfied*.

1.1 Unusual properties of the situation

Reading the last paragraph closely, we see that the situation involves two unusual properties.

- The invariants can't be formulated once and then kept fixed. Instead the invariants has to be gradually discovered and refined by the system which is supposed to maintain them, in collaboration with all stakeholders capable of expressing opinions, including humans, AI systems, and so on.
- It is often the case that invariants can only be satisfied approximately. For example, it is unreasonable to assume that the system can satisfy a particular invariant before that particular invariant has even been formulated. And it's unreasonable to assume that the system will be able to suddenly jump to a state where a particular invariant is exactly satisfied as soon as that invariant is formulated. And in many cases, whether a desirable invariant is satisfied would be a fuzzy thing, not a binary yes/no thing.

In particular, the process of formulating and refining the invariants needs to satisfy various properties establishing its non-pathological nature (the process of deciding on invariants should be fair[4], should remain fair, and ideally should gradually become more fair, whereas a "rigged" process of establishing invariants might be as bad as not having any process whatsoever).

This "reflection aspect", the need to evolve and maintain non-trivial invariant properties of the evolving process of formulating and refining the invariants is a particularly non-trivial aspect of the overall situation.

[1] when people talk about creating artificial software engineers and artificial AI researchers, they often neglect to take into account the fact that real-world teams consist of different people with different skillsets, different approaches and preferences, different strong and weak points; thankfully, LLMs do encourage us to move into the direction of simulating drastically different types of people, and we already have indications that there is a lot of strength in introducing this kind of diversity, see e.g. "Mindstorms in Natural Language-Based Societies of Mind" by Mingchen Zhuge et al., `https://arxiv.org/abs/2305.17066` and `https://github.com/mczhuge/NLSOM`

[2] self-improving in an open-ended manner and including advanced AIs, humans, and more

[3] such as interests of sentient beings, interests of potentially sentient beings, interests of natural environments and ecosystems, interests of preservation of diverse cultural phenomena, interests of various innovative research, discovery, and development, etc.

[4] in some reasonable sense (to be figured out)

1.2 Structure of this paper

Section 2 discusses a small subset of known self-modification and self-improvement experiments. Section 3 talks about the few attempts to study invariant properties of self-modifying systems I currently know about. Section 4 discusses the fact that formal correctness proofs (while useful where feasible) are, in any case, insufficient, and therefore other methods of establishing correctness need to also be pursued. Section 5 discusses pre-deployment testing and post-deployment run-time correctness checks and safety measures.

Section 6 discusses various reasons which support a (rather obvious in retrospect) claim that gradual and continuous changes are easier to manage in this sense than abrupt discrete changes. Section 7 discusses various ways to make sure that changes are gradual and continuous. It notes that while some techniques to make changes gradual and continuous are available even for conventional software systems, it is much easier to make changes gradual and continuous for self-modifying *continuous neuromorphic software systems*. Fortunately, we do have a relatively well-developed research direction which studies continuous neuromorphic architecture for general-purpose self-modifying software known as *dataflow matrix machines (DMMs)*, and therefore we do have an option of taking advantage of this architecture for self-modifying continuous neuromorphic software.

Section 8 starts moving to the technical core of our work (which, mostly, is future work, as the present text is currently more like a sketch of a future research program). First of all, we need to model systems operating in a changing world. In this sense, dataflow matrix machines are expressive enough to serve as models of the world and of different parts and subsystems of the world at different scales. In particular, dataflow matrix machines have enough hierarchical structures to host evolving populations of dataflow matrix machines within themselves (those DMMs can in turn host populations of DMMs, and so on). We have at least two different research directions. One can consider models of systems having certain properties and emitting outputs having certain properties *without actually implementing* systems with those properties/systems being able to actually emit those outputs. This is a theoretical and modeling direction. Another direction is to study well-contained and limited self-modifying and self-improving systems. I call this second direction *"mini-foom" studies*, the name being chosen to emphasize that various safety precautions are needed if people decide to move forward with this direction. In both of these research directions there is enough room for trying to prove statements about invariant properties and for studying those properties experimentally and refining techniques for testing and for run-time correctness checks. Section 9 concludes focusing on the key difficulties.

Appendix A briefly overviews the current state of knowledge about dataflow matrix machines and references the relevant resources.

Appendix B notes that it is unlikely that self-improving super-intelligent AI systems would enforce and maintain anthropocentric invariant properties as they evolve further and further. Instead one should aim for *"natural" non-anthropocentric invariants*, such that the safety properties we desire emerge as *corollaries* of those "natural invariants".

Appendix C considers different levels of initial modeling research I hope to do, starting from invariants in traditional software engineering ignoring the considerations beyond the narrow computations in question, progressing towards avoiding undue impact on the external environment, and further progressing towards protecting the external environment from other sources of undue impact or from other deterioration.

2 Rudimentary self-modification and self-improvement is not hard

People are occasionally experimenting with various self-modification and self-improvement scenarios. So far the results are interesting and tickle our curiosity, but are not overwhelming. However, this "results being not overwhelming" is not guaranteed in the future.

We have a long history of self-improvement studies, and this is a rather active research area.

Here is a very small non-representative selection of relatively recent examples.

A new Microsoft Research/Stanford study led by Eric Zelikman[5]. In that study a scaffolding program that structures multiple calls to an LLM to generate better outputs improves itself using suggestions and guidance from the underlying LLM. The LLM itself is fixed, so this is a limited, but rather interesting form of recursive self-improvement. Figure 4 on page 6 is of particular interest: it shows that the scheme does not work and the quality is degrading when the underlying LLM is GPT-3.5, but the scheme leads to genuine recursive self-improvement when the underlying LLM is GPT-4. Nevertheless, each improvement is smaller than the previous one, and the process saturates after a few iterations even with GPT-4. It is an open question whether an upgrade from GPT-4 to a not-yet-existing more powerful model would be sufficient to drastically improve

[5] "Self-Taught Optimizer (STOP): Recursively Self-Improving Code Generation", `https://arxiv.org/abs/2310.02304`

the performance of this scheme for recursive self-improvement, or whether the scheme would need to also be modified for that. Still, this paper gives us a taste of what is likely to come in the near future.

An older example of self-modifying neural systems comes from dataflow matrix machines which are highly expressive, but not highly optimized neural machines, capable of expressing compact general-purpose stream-oriented programs, with those programs being continuously deformable. These neural machines are capable of fluent self-modification (see Section 7.2 and Appendix A for details on those neural machines and on relevant self-modification primitives and experiments). These machines should be quite usable instead of Python programs as scaffolding software in the context of the "Self-Taught Optimizer" study above.

However, a considerable degree of caution is advised here. It's all good as long as we are *quite certain* that the process of recursive self-improvement saturates before long. If we are not quite certain of that, we should master considerable ability to meaningfully reason about the properties of self-improving software before risking to unleash it. I'll be further discussing how to approach this kind of particularly risky research area in the subsequent parts of this text.

3 The studies of invariant properties of self-modifying systems are rare

Recently I have been trying to research the state of studies of invariant properties of self-modifying systems. In particular, I have been trying to find whether much is known besides the relatively well known series of studies conducted by MIRI.

Of course, there are plenty of studies of invariants for conventional software, where invariant properties are part and parcel of software correctness proofs using Floyd-Hoare logic and similar techniques. There are even some studies of invariants of recurrent neural networks.

However, I have not been able to uncover much in regard to invariants of self-modifying systems. All I have found besides MIRI studies has been a very lightly cited 1995 paper "S-and T-Invariants in Cyber Net Systems" by Yuan Chongyi, and asking on LessWrong has not yielded any additional references[6].

Speaking about the MIRI line of studies of properties of self-modifying systems, I believe that those studies have mostly been derailed by excessive focus on the so-called "Löbian Obstacle". More about this is to be said in the next section.

4 Proofs are not full-proof

Here I am going to list various reasons for mathematical proofs not providing full safety guarantees while being an important ingredient of the overall safety effort.

The first and the most obvious reason is that a formalization might fail to capture the reality with sufficient degree of faithfulness. Then many other kinds of mistakes can be made: people or AIs operating the process of verification might make avoidable errors (for example, in the Arian 5 maiden flight disaster, the software verification process produced invalid guarantees because of errors committed by people running the verification), proof verification software might be defective, the logic might be inconsistent (the full guarantees of consistency are next to impossible due to results by Gödel, and the concerns related to the so-called "Löbian Obstacle" also belong here[7]), and so on.

The bottom line is that formal proofs of correctness are useful, they greatly increase reliability of mission-critical software, but they are not full-proof, and they do not abolish the need for various forms of testing, run-time correctness checks, and redundancy.

This is why I think that the focus of MIRI researchers on the "Löbian Obstacle" was excessive, and that it is regretful that this focus prevented them from obtaining stronger positive results about properties of self-modifying systems.

The "Löbian problems" simply need to be recognized as one of the factors which prevent formal proofs from being absolutely reliable in practice. But nothing can make formal proofs absolutely reliable in practice. At the end of the day it's all about decreasing chances for things to go drastically wrong, not about impossible "absolute safety guarantees".

[6] `https://www.lesswrong.com/posts/sDapsTwvcDvoHe7ga/what-is-known-about-invariants-in-self-modifying-systems`

[7] see `https://en.wikipedia.org/wiki/Curry's_paradox`; see also Morgan Rogers, "Escaping the Löbian Obstacle", `https://www.lesswrong.com/posts/gbNLvkGuGcmSFFpSE/escaping-the-loebian-obstacle` for a more in-depth discussion

5 Pre-deployment testing and post-deployment correctness checks and run-time safety measures are important

As proofs alone are not enough, a variety of diverse pre-deployment property testing and of post-deployment correctness checks and other run-time safety measures would be necessary.

Here, of course, we are facing a problem which is not typical for a more pedestrian software: while the proof process analyzes a static passive object[8], other forms of testing deal with active processes and, in the case we are particularly interested in, they will deal with superintelligent processes which might be smarter than the present generation of those processes.

The standard problem here is that if you have a running superintelligent process which is smarter than the state-of-the-art and which has unfavorable properties, then it might be too late.

6 Gradual and continuous changes are likely to be helpful

Continuous and gradual changes are helpful in two ways in this context.

First of all, the new versions are not so different and therefore are unlikely to have an overwhelming advantage over the present versions, so it should be possible to mitigate the difference in capabilities until we are reasonably confident in the properties of the new version. As we shall see below, the methods ensuring that the changes are gradual and continuous tend to include some mitigation measures for free (that is, we shall see that the mechanisms ensuring that the "delta" is small tend to also mitigate the dangers from that "delta" to some extent).

Another aspect is that with changes being small, it would often be easier to reason about the difference in properties compared to the current version, as opposed to dealing with rather arbitrary drastic changes.

7 Ways to maintain continuous and gradual nature of changes

What can we do to make sure that changes are continuous and gradual, rather than unpredictable sharp jumps?

Some steps in that direction can be made with conventional software, but more can be done if the overall system is architected as a generalized neural machine, as a continuous neuromorphic system.

7.1 Conventional discrete software systems: what can be done to maintain continuous and gradual nature of changes

7.1.1 Run new additions on slow clocks, accelerate gradually

In a multiprocess system, one can run new additions on arbitrarily slow clock rate, and accelerate them gradually as confidence in those new additions improves.

This both ensures that effect of the new additions on the functioning of the overall system would increase gradually, and gives the status quo system more time to react and take measures if things go wrong.

7.1.2 Include new additions into only a few instances initially, proliferate gradually

In a multi-instance system, one can start with incorporating new additions into the instances gradually, so that there are a lot of unaffected instances at first.

Moreover, in order to maintain diversity and to aid conservation, one might make sure that some instances from older generations are kept running unchanged indefinitely.

7.2 Continuous neuromorphic software systems: more can be done to maintain continuous and gradual nature of changes

All methods of making changes continuous and gradual applicable to conventional software would be applicable to continuous neuromorphic software as well, but transition to continuous neuromorphic software also enables new ways of making changes continuous and gradual.

[8] a blueprint consisting of the source code

7.2.1 Continuous neuromorphic software systems

We do have a relatively well-developed research direction which studies continuous neuromorphic architecture for general-purpose software known as *dataflow matrix machines (DMMs)*. Formally speaking, one considers recurrent neural networks and replaces streams of numbers with arbitrary *linear streams*[9].

For single neurons in traditional RNNs, their inputs compute linear combinations of input numbers with weights. Here these inputs become, roughly speaking, attention devices computing linear combinations of high-dimensional objects. So the resulting recurrent neural machines can be viewed as flexible attention machines.

One adds some further improvements, such as

- arbitrary input arity of single neurons (so that a single neuron admits multiple linear combinations as separate inputs),
- arbitrary output arity,
- powerful built-in stream transformations in place of activation functions inside single neurons,
- somewhat non-standard flexible tensors with tree-shaped indices,
- ways to embed discrete data into this continuous framework without traditional embedding distortions,

and one gets a powerful platform for stream-oriented general-purpose programming with continuously deformable programs. Small programs are expressible by small compact human-readable neural machines. The dimension of the inputs and the size of the machine are decoupled from each other, just like in traditional programming.

The neural machines in this class can be equipped with fluent and convenient self-modification capabilities. The traditional difficulties of controlling N^2 weights by N scalar outputs in self-referential RNNs are not applicable here, because each output can have high or infinite dimension. Hence these machines are well-equipped to express various self-modification and self-improvement methods, and, in particular, to learn better self-improvement methods.

In their full-generality these machines are not optimized for GPUs. Without considering possible further research and engineering efforts, these machines currently come as a CPU-oriented continuous neuromorphic programming framework. Further information on these neural machines is in Appendix A.

7.2.2 Attenuation of the outputs

Because the outputs exist in the form of linear streams, and a linear stream can always be multiplied by a numerical coefficient, one can initially attenuate the outputs (by magnitude of the output vectors or by sampling frequency when the output linear stream is a stream of samples from a probability distribution).

And then one can gradually lift this attenuation, just like one gradually lifts the slowdown in Section 7.1.1.

7.2.3 Attenuation of the weights

We are talking about self-modifying neural machines which are capable of gradual expansion, of producing new neural connections where there were no connections before. Formally speaking, we are talking about infinite, countably-sized connectivity matrices with finite number of non-zero weights at any given moment of time. Those non-zero weights correspond to the existing neural connections, whereas zero weights correspond to the potential connections which can be brought into life as the number of non-zero weights increases.

Here one can initially keep the absolute values of these new non-zero weights small and only allow them to increase gradually as the confidence in the quality of the new additions improves.

8 How to include the changing world in our modeling

The existential safety and positive vs. negative AI impact is about the effects of advanced AI in the world, not in the sandbox. So we need to include the world in our models of the situation. In some sense, one can consider the reality as a whole as the overall self-modifying system around us.

As the model is thus always smaller and less complex than the system itself, full guarantees are impossible. Nevertheless, we should be able to improve the chances of "good outcomes".

[9] streams which can be combined together with coefficients (a version of the notion of "linear combination of steams" is defined)

8.1 DMM-based models of the self-modifying world and of AI systems in the world

Dataflow matrix machines described in Section 7.2 and Appendix A are sufficiently expressive to model the world and to model different parts and subsystems of the world at different scales simultaneously. In particular, dataflow matrix machines have enough hierarchical structures to host evolving populations of dataflow matrix machines within themselves (those DMMs can in turn host populations of DMMs, and so on).

So we have this very convenient *"homoiconic"* situation where the structures based on dataflow matrix machines and nested dictionaries can describe everything: world models, programs, machine learning models, and data.

8.2 Modeling self-modifying systems vs "mini-foom" experiments

There are at least two different directions here. One can consider models of systems having certain properties and emitting outputs having certain properties *without* actually implementing systems with those properties/systems being able to actually emit those outputs. This is a theoretical and modeling direction we should definitely pursue.

Another direction is to study well-contained and limited self-modifying and self-improving systems. I call this second direction *"mini-foom" studies*, the name being chosen to emphasize that various safety precautions are needed if people decide to move forward with this direction.

The catastrophic risks might include both accidentally creating actual "foom" instead of a well-contained and limited "mini-foom", and developing the techniques which might allow other people (or computer systems) to carelessly create actual "foom". So one needs to spend a good deal of time pondering whether it is timely to explore this direction now, rather than waiting until we understand more about "foom" safety on a theoretical and modeling level. Nevertheless, modest well-contained experiments with self-modifying systems not involving too much self-improvement pressure should be relatively safe.

In both of these research directions there is enough room for trying to prove statements about invariant properties and for studying those properties experimentally and refining techniques for testing and for run-time correctness checks.

9 Conclusion: reliability should improve together with capabilities

These are just the first steps. The need to continually research the desirable invariants and to modify invariants to aim for as the ecosystem self-improves is a huge complication. Also we should keep the following in mind:

- **Transitivity problem.** How do we know that a successor would care about its own successors preserving the same invariants at least to the same degree as the degree to which the system in question cares at the moment? So we need to figure out the invariants which describe the meta-level of caring about invariants and caring about successfully applying procedures to maintain those invariants.
- **Imperfect transitivity.** Another problem is that reliability of self-improving steps with regard to respecting the invariants needs to gradually improve. If there is a constant probability P to drastically screw-up during a single self-improvement transition, then sooner or later the numbers will catch with us. Instead the overall process needs to be organized so as to make P go down quickly enough as self-improvement progresses, so that the *accumulated probability* of big disasters remains reasonably low as time goes forward indefinitely.

Acknowledgments

I am grateful to various members of Boston area Machine Learning community for inspiring discussions and very helpful feedback.

APPENDICES

A Dataflow matrix machines and V-values[10]

This Appendix contains further information about neural machines discussed in Section 7.2. The main theoretical work on dataflow matrix machines was done in 2015-2017, and the first research software prototypes were created during that period. The canonical reference paper covering that period of research has been published in 2017[11].

Two main novel motives associated with this approach are fluent compositional self-modification and neuromorphic continuously deformable general purpose programs.

A.1 Self-modification: theory and experiments

Using neural networks for metalearning is always non-trivial. In particular, dimension mismatch, namely the number of neuron outputs being much smaller than the number of network weights, means that a neural network can only modify itself in a highly constrained manner. Dataflow matrix machines address this problem by enabling high-dimentional outputs of single neurons and have **powerful and flexible self-modification facilities**.

In the technical sense, DMM literature has been focusing on the self-modification scheme based on allocating a single neuron *Self* which outputs the current network matrix on each step, and takes this output and any possible updates from other neurons at its inputs, making *Self* an accumulator of network matrix updates (see Section 6.1 and 7 of `https://arxiv.org/abs/1712.07447`).

Therefore, a dataflow matrix machine can be equipped with a variety of primitives which perform self-modifications, and it can fruitfully learn various linear combinations and compositions involving those primitives.

Self-modification facilities of dataflow matrix machines are not limited to the weight changes for the existing connections in the network. The available primitives allow to modify the network topology as well. For example, primitives allowing the network to control its own fractal-like growth by the means of cloning its own subnetworks are available.

Therefore, this is a very promising architecture not only for methods of learning to learn better in a traditional sense, but also for methods of learning to perform neural architecture search better.

A dataflow matrix machine can comfortably host an evolving population of other DMMs inside itself, so it is an excellent environment for neuroevolution experiments and, in particular, for the experiments aiming to learn to evolve better (or to evolve to evolve better).

These self-modification facilities have been used in software experiments to

- produce controlled wave patterns in the network matrix;[12]
- create randomly initialized self-referential DMMs which generated interesting emerging behaviors;[13]
- edit a running network on the fly by sending it requests to edit itself (in particular, this enables **live-coding**, but this is also quite open-ended, since it enables a population of networks to tell each other to modify themselves; of course, the receiving network doesn't have to follow an incoming instruction to self-modify blindly, although in the most simple-minded case it would do so).[14]

[10] This appendix reuses fragments from Michael Bukatin, "Dataflow Matrix Machines: a Collaborative Research Agenda", `https://anhinga.github.io/brandeis-mirror/dmm-collaborative-research-agenda.pdf` with some modifications

[11] Michael Bukatin, Jon Anthony. "Dataflow Matrix Machines and V-values: a Bridge between Programs and Neural Nets", `https://arxiv.org/abs/1712.07447`, see Section 11 for historical remarks and related work

[12] see Appendix B.2 of the LearnAut 2017 paper, Michael Bukatin, Jon Anthony, "Dataflow Matrix Machines as a Model of Computations with Linear Streams", `https://arxiv.org/abs/1706.00648`

[13] see Section 1.2 of "DMM technical report 11-2018", `https://www.cs.brandeis.edu/~bukatin/dmm-notes-2018.pdf`

[14] see Section 1.1 of the same technical report

A.2 Various kinds of linear streams

There are many different kinds of linear streams. They include streams of numbers, sparse vectors and sparse tensors (both of finite and infinite dimension), streams of functions and distributions. We found streams of V-values (**flexible tensors** based on tree-shaped indices) to be of particular use.

A single dataflow matrix machine can process a large variety of different kinds of linear streams, or it can be based on a single kind of linear streams, sufficiently expressive for a given class of situations.[15]

The detailed theory of V-values with scalar leaves can be found in Section 3 of the reference paper[16], and the generalization to arbitrary leaves can be found in Section 5.3 of that paper.

Section 5 of the reference paper contains the theory of linear streams, which explicitly deals with the fact that, in the abstract mathematical sense, we often need to represent infinite objects but, for a computer implementation, we need to consider streams of approximate finite representations (in particular, streams of distributions would often be represented by streams of samples from those distributions).

One important feature of DMMs as a general purpose programming platform is that they are able to process discrete data without embedding distortions. I am aware of two different ways to do so in context of dataflow matrix machines and linear streams. One approach involves formal linear combinations of objects of interests and streams of those formal linear combinations. Another approach is to consider streams of probabilistic samples of objects of interests.[17]

A.3 Hierarchies and world modeling

Tree-shaped V-values, and the ability of those V-values to host non-scalar leaves is very convenient for modeling hierarchical structures.

Neurons in the DMM formalism can also be very complex, they can host complicated changing state via accumulator connectivity pattern (Section 6.1 of `https://arxiv.org/abs/1712.07447`) and arbitrary complex algorithms for processing their input streams, so the whole networks with different clock speeds can be hidden inside neurons. Neurons are supposed to respond in time according to the clock speed of the network containing those neurons, but the ability to output zeros as a default is always available and can be used if a computation is late. The subnetworks can also be created within the network weight-connectivity matrices which are infinitely-dimensional matrices with finite number of non-zero elements at any given moment, so extra space can always be allocated as necessary.

Hence this formalism is very convenient for world modeling at different time scales and different levels of complexity within a single model. The only thing which is currently missing is continuous time, the formalism can handle coexisting different clock speeds, but the time is currently discrete.[18]

A.4 Continuously deformable programs: manually crafted and synthesized

The dimension of the network and the dimension of data are decoupled, so compact neural machines for solving conventional programming problems are available. For example, by considering streams of maps from words to numbers, one can build a dataflow matrix machine counting words in a given text which uses only a few neurons (Section 3 of `https://arxiv.org/abs/1606.09470`). Similarly, by considering streams of V-values (flexible tensors based on tree-shaped indices) and embedding of lists into trees, one can build a similarly compact dataflow matrix machine accumulating a list of asynchronous incoming events (e.g. mouse clicks, see Section 6.3 of the DMM reference paper, `https://arxiv.org/abs/1712.07447`).

For more examples of DMMs as programs, see *Map of DMM-related programming examples and techniques*:

`https://github.com/anhinga/2020-notes/tree/master/programming-overview`

[15] for a DMM processing different kinds of linear streams, one can have separate weight-connectivity matrices for each kind of streams, with transformations between streams happening only inside the neurons, or alternatively one can have conversion rules between kinds of linear streams; DMMs based on a single kind of linear streams, such as streams of fixed-rank tensors of fixed finite or infinite dimensions, or streams of flexibly-shaped V-values, are conceptually much simpler

[16] `https://arxiv.org/abs/1712.07447`

[17] for detailed treatment, see a 2020 short preprint "Using streams of probabilistic samples in neural machines", `https://www.cs.brandeis.edu/~bukatin/dmm-probabilistic-samples.pdf`

[18] we have dynamically changing sparsity structure, implying that the essentially discrete objects can emerge and disappear, and the effective dimensionality can change dynamically, so if continuous time is desirable, a thoughtful approach is required in order to add it in a seamless manner

The task of synthesis of dataflow matrix machines should be more tractable than conventional program synthesis. When one works with DMMs, the task of *learning program sketches* is reformulated as *neural architecture search*, and converting a program sketch to a full program should be done by conventional methods of neural net training.

Dataflow matrix machines allow us to combine

- aspects of *program synthesis* setup
  (compact, human-readable programs);
- aspects of *program inference* setup
  (continuous models defined by matrices).

Differentiable programming with tree-like structures used to be quite challenging, but the situation has improved with the advent of the latest generation of differentiable programming systems. JAX[19] and Zygote.jl (Julia Flux) are capable of taking gradients with respect to variables stored inside nested dictionaries.

First successful experiments in DMM training and in program synthesis/circuit synthesis/DMM synthesis via neural architecture search were performed in June 2022 using Zygote.jl. The synthesized DMMs had pretty impressive generalization properties. These experiments were presented at JuliaCon 2023.[20].

A.5 GPU vs CPU

There are various GPU-friendly subclasses of dataflow matrix machines, for example, the subclass of self-referential lightweight pure dataflow matrix machines which are DMMs of fixed shapes based on streams of rectangular matrices of fixed size.[21]

At the same time, making more general and more flexible classes of DMMs GPU-friendly and optimization-friendly remains an open research and engineering problem.[22]

What options are available, short of solving the problem of GPU-friendly or using more flexible accelerators, such as FPGA-based solutions or Cerebras hardware? Purely CPU-based DMMs should still be sufficiently powerful for solving mid-size problems. For larger problems, it should be fruitful to consider hybrid architectures between DMMs and more traditional GPU-based machine learning models, such as, for example, Transformers.[23]

In particular, returning to Section 2 and "Self-Taught Optimizer", instead of using scaffolding software written in conventional Python and running on CPU, one can use modestly sized DMMs running on CPU as scaffolding software for LLMs (running on GPUs), so this would be a typical hybrid setup discussed in the previous paragraph.

A.6 Dataflow Matrix Machines resources

The useful links include:

Reference paper: `https://arxiv.org/abs/1712.07447`

Reference slide deck: `https://web.archive.org/web/20220305051310/https://researcher.watson.ibm.com/researcher/files/us-lmandel/aisys18-bukatin.pdf`

GitHub Pages: `https://anhinga.github.io`

[19] see `https://github.com/anhinga/jax-pytree-example`

[20] see `https://github.com/anhinga/DMM-synthesis-lab-journal/tree/main/JuliaCon2023-talk`

[21] Appendix D of "Notes on Pure Dataflow Matrix Machines: Programming with Self-referential Matrix Transformations", `https://arxiv.org/abs/1610.00831`

[22] various drafts making a bit of headway towards this goal are publicly available, and the problem of GPU-friendly computations with sparse matrices and sparse tensors with dynamically changing sparsity structure is closely related to the problem of GPU-friendly flexible DMMs

[23] mathematically speaking, Transformers and many other traditional classes of machine learning models can be thought of as subclasses of DMMs, so one can often think of this kind of hybrid setups as of setups where faster specialized DMMs (parts of DMMs) are running on GPUs, and slower, more flexible parts are running on CPUs

B Non-anthropocentric AI existential safety

Traditional AI alignment approaches formulated in anthropocentric terms of aligning AI systems to human values are unlikely to work for superintelligent entities, because when one considers a self-improving ecosystem of superintelligent entities undergoing multiple drastic self-restructurings, there is no reason why externally imposed values and goals would survive throughout these changes.

Given that people also tend to add the additional desideratum of not overconstraining the future by our limited views and values, the traditional formulation of AI alignment for superintelligent systems is likely to be fundamentally inconsistent.

Instead of focusing on unworkable anthropocentric approaches, researchers should aim for *"natural" non-anthropocentric invariants*, such that the safety properties we desire emerge as *corollaries* of those "natural invariants".

We are recently seeing a number of publications and informal thoughts pointing in this general direction.

I noted in a recent LessWrong essay[24] that *"the rapidly improving ecosystem of superintelligent AIs will face technology-related existential risks of its own"*.

More specifically, that essay argues that it is likely that rapid progress in fundamental physics will resume and that technologies enabled by this progress will potentially be a threat to the "fabric of reality" (at least in the local neighborhood) and might therefore be a threat to the existence of the ecosystem of superintelligent AIs itself (together with everything else around it).

Quoting from the essay:

> *So, the AI-ecosystem will have to deal with the issues which are similar to the issues our human community is currently dealing with with rather limited success. Collaboration vs competition of its members, the right balance between freedom and control, careful consideration of whether novel experiments are too risky to the fabric of physical reality, how all this interplays with creation of smarter and smarter offspring, what should be done to make sure that the smarter and smarter offspring remain faithful to all aspects of the existential safety agenda despite unpredictable sharp left turns.*

But, as the essay notes, we have a fundamental difference between this situation and the situation of externally imposed alignment constraints:

> This is a difficult problem, but it's in the intrinsic self-interest for the community of superintelligent AIs to competently address this problem, so the values and goals of competently and successfully addressing this problem have good chances of being preserved throughout continuing self-improvement and "sharp left turns".

So there are reasons to be more optimistic in this sense, compared to the traditional approaches to AI alignment. However, this kind of existential safety is not sufficient for us, we need more than that. For example, two-fold permanent change in the concentration of oxygen in the atmosphere would be catastrophic to us,[25] whereas AI systems should be able to handle this kind of change without serious problems.

Here the approaches proposed by various researchers on how to make it so that superintelligent AIs are naturally interested in maintaining the properties we need (instead of the hopeless attempts to impose those constraints in the adversarial manner) understandably diverge. This is actually a good sign. We can hope that some combination of the approaches being proposed would work.

For example, my essay in question proposes to try to make sure that the AI ecosystem has a lot of the "salient sentient beings" within itself, having sufficient clout and being interested in the AI community respecting their interests throughout the process of rapid evolution.[26] The essay observes that in this scenario

> For each of [those salient sentient beings], adopting the value of *taking interests of "all sentient beings" into account* minimizes the risk of being eventually dropped from the set of beings whose interests are taken into account.

[24] "Exploring non-anthropocentric aspects of AI existential safety", April 2023, `https://www.lesswrong.com/posts/WJuASYDnhZ8hs5CnD/exploring-non-anthropocentric-aspects-of-ai-existential`

[25] so a reasonable concentration of oxygen in the atmosphere is one of the invariant properties we need to have maintained, see e.g. `https://chat.openai.com/share/85259243-d88c-48a8-a508-6af09f8e6334`

[26] Of course, the key problem with any sentience-based approach is that we don't have a good theory telling us what entities are sentient and what kind of subjective reality they have. So the hope here is that various experiments in creating hybrids between biological and electronic entities (using Neuralink-style or, as one might prefer, non-invasive BCI) will inform AIs about human subjective experience and will facilitate a continuity of spectra of subjective experiences between biological and electronic entities. Such experiments might be initiated by humans or by AIs, as some AIs are likely to be sufficiently curious to be interested in human subjective experience.

therefore

> So if "salient sentient beings" do maintain enough clout in the AI community throughout its evolution and "sharp left turns", the value of *taking interests of "all sentient beings" into account* stands good chances of being preserved. And then the mechanisms of cooperation and moral reasoning developed for the sake of other matters (such as the particular existential risk discussed above) should be useful in implementing this goal.

Another non-anthropocentric approach based on ethical rationalism and Gewirth' Principle of Generic Consistency with respect to Prospective Purposive Agents and reliant on appropriate flavors of modal logic and automated theorem proving is proposed by András Kornai in 2012[27] and expanded upon in a 2023 preprint.[28]

The approach based on liberal norms and on refraining from "othering" the AIs is proposed by Joe Carlsmith in January 2024.[29]

In July interview,[30] Ilya Sutskever, a co-lead of the OpenAI Superalignment team, seems to suggest to narrow down the definition of the notion of alignment from defining alignment as being able to "to steer and control AI systems much smarter than us"[31] to defining it as being able to prevent a catastrophic blow up, "as an analog to nuclear safety".

He also seems to advocate in that interview for a more collaborative non-adversarial approach, where humans and superintelligent AI systems collaborate in formulating values and goals, and methods of reaching those goals.

Further down the road he is expecting that some humans will need to take an approach of merging with AIs. He also thinks we need to aim for forming children-parent relationship with superintelligent AIs:

> "The upshot is, eventually AI systems will become very, very, very capable and powerful. We will not be able to understand them. They'll be much smarter than us. By that time it is absolutely critical that the imprinting is very strong, so they feel toward us the way we feel toward our babies."[32]

What is common for all these different proposals is that human and non-human entities should have adequate voice in deciding what properties the world around us should have, and what properties this decision-making process should have.

In some sense, the spirit of all this is captured by my own 2012 quote::

> The idea of trying to control or manipulate an entity which is much smarter than a human does not seem ethical, feasible, or wise. What we might try to aim for is a respectful interaction.

---

[27] András Kornai, "Bounding the impact of AGI", *Journal of Experimental and Theoretical Artificial Intelligence* (2014), **26.3**, pp. 417–438, `https://kornai.com/Drafts/agi12.pdf`

[28] András Kornai, Michael Bukatin, Zsolt Zombori, "Safety without alignment", `https://arxiv.org/abs/2303.00752`

[29] See the sequence "Otherness and control in the age of AGI", `https://www.lesswrong.com/s/BbAvHtorCZqp97X9W` and, in particular, the essays "Otherness and control in the age of AGI" and "Being nicer than Clippy". Note that the author disclaims: "I don't think that "niceness/liberalism/boundaries" is enough, on its own, to ensure a good future, or to allay all concern about trying to control that future over-much".

[30] "Ilya Sutskever's thoughts on AI safety (July 2023): a transcript with my comments", `https://www.lesswrong.com/posts/TpKktHS8GszgmMw4B/ilya-sutskever-s-thoughts-on-ai-safety-july-2023-a`, a transcript I posted on LessWrong

[31] `https://openai.com/blog/introducing-superalignment`

[32] `https://time.com/collection/time100-ai/6309011/ilya-sutskever/`

C Different levels of including "the world"

This section mentions various initial experiments one can try in order to start making progress in the studies of invariant properties of self-modifying systems.

We only consider one axis here: whether to ignore the existence of "the world", whether to try to avoid impact on "the world", or whether to try to stabilize some properties of "the world".

C.1 Level 0: Isolated self-modifying systems

At level 0, we ignore "the world" beyond what's relevant to the narrow computation in question.

C.1.1 Periodic and quasi-periodic self-modifying DMMs

We have published hand-crafted periodic self-modifying DMMs and also randomized self-modifying DMMs in which various quasi-periodic patterns such as oscillations and sleep-wake behavior emerge (Appendix A.1).

The questions which arise in the case of periodic self-modifications (precisely crafted wave patterns propagating through the network connectivity matrix) are related to proper formalization of intuitively obvious presence of invariants.

The questions which arise in the case of emerging quasi-periodicity are more open-ended and involve understanding the mechanisms and reasons for quasi-periodic behaviors in question and the nature of invariant properties involved, as right now we see that something seems to be preserved in those randomized self-modifying machines, but we don't know how to formulate those invariant properties and don't know, if they will be preserved indefinitely or only for some limited duration.

C.1.2 Traditional software engineering

A study here might involve taking a modest imperative program involving `while` loops together with its invariants under Floyd-Hoare formalism, porting this program to the language of self-modifying DMMs in such a way that `while` loops are expressed via self-modifications and studying what happens to Floyd-Hoare invariants in the resulting self-modifying DMM.[33]

C.2 Level 1: Avoiding undue impact

At level 1, we try to make sure that our system avoids undue impact on "the world".

The simplest way to start modeling this is to recall that computations are not free, but involve various costs in resources, environmental damage, and so on. In a model world, with processes flexibly competing for computational resources, we would like to see some moderation in this sense.

For example, we can have a model world being an artificial ecosystem with computational resources slowly growing at a fixed rate, and we would like to see that a model ecosystem of software processes which would generally like to expand its usage of resources would refrain from increasing its relative share of allocated resources, even when not constrained from outside from grabbing larger and larger share.

C.3 Level 2: Stabilize the environment against disruptions

At level 2, we need to start modeling an actual world with some desired properties, which might initially hold, but be under pressure from other sources, and the AI ecosystem in question working to counter that pressure and to keep the properties in question satisfied or approximately satisfied.

[33] We have quite a bit of experience translating imperative software to DMMs, but we have not tried to specifically express imperative loop constructions via self-modifications of the network weight-connectivity matrix; such an exercise would be a continuous analog of a possible translation of an imperative program to lambda-calculus and looking at what happens to Floyd-Hoare invariants in terms of lambda-calculus; here instead of lambda-calculus which is a discrete formalism we have an equally fundamental and reasonably high-level continuous formalism of self-modifying DMMs.