HYPERCOMPUTATION:

Introduction
It has long been assumed that Turing Machine is a sufficient model for computing. Anything that is computable can be simulated by a Turing Machine. Around middle of twentieth century, Alan Turing came up with the notion of computability. He formally described it in the form of a Turing Machine. At around the same time, Alonzo Church came up with the concept of Lambda Calculus. A few other models of computations were also developed like recursive functions, register machines, etc.

Alonzo Church thesis established the equivalence of all these models of computation. In layman’s term, anything which can be expressed by Turing Machine can also be described by Lambda Calculus and other aforementioned models of computations. On the contrary, there does exist some models of computation that are not as powerful as Turing Machines. The flip-side is that they are simpler to describe. Two of the most commonly used such models are Finite State Machine and Context Free Grammar. Another yet powerful model is that of Context Sensitive Grammar.

Due to underlying simplicity of a Turing Machine and legacy reasons, it was chosen as the de facto model of computation. Hence, it is used extensively computer scientists and complexity theorists alike. Every modern computer in today's world is essentially a mimic of Turing Machine albeit in a limited fashion.

This artwork tries to explore beyond the realms of a Turing Machine.

History
The field of Computer Science and Mathematics experienced three major revelations in the 20th century which led us to question the established norms of computations. Several powerful negative results were concluded about the limits of computation. They are briefly described as follows :-

1. Godel's Incompleteness Theorem

Godel showed that there are propositions in the language of arithmetic that are either true but unprovable or false but provable. He also showed how everything in mathematics can be transformed into arithmetic essentially implying unprovable in mathematics. In a way, no formal systems of axioms can prove everything. As an implication naïve set theory gave way to axiomatic set theory the most common of which is ZFC (Zermelo-Frankel set theory).

2. Turing Machines

The incompleteness of mathematical theorems was very intimately tied down to the notion of a formal system. Since Entscheidungsproblem and Hilbert's Tenth Problem were unresolved till date, this led mathematicians to suspect that there doesn't exists a solution to them. However, proving the absence required a precise formulation which gave rise to Turing Machine. The premise description of a Turing Machine is omitted as it is too mathematical to describe. Godel praised Turing Machine as “precise and unquestionably adequate definition of a general concept of formal system”.
Further more Turing went on to show that Halting Problem is indeed not computable on a Turing Machine which further bolstered Godel's Incompleteness Theorem. Till now, even after Godel's Incompleteness Theorem, it was widely speculated that each statement of arithmetic was computable is some system. Unpredictability of Halting Problem gave a serious dent to this notion and showed that undecidability in mathematics is even more widespread.

3. Algorithmic Information Theory

The field of Algorithmic Information Theory was developed by Andrei Kolmogorov and Gregory Chaitin and asks questions about information of a string. Formally, the complexity of a string is defined as the length of the minimum length program to describe the string. Here also, analogous to Halting Problem, computing complexity of a string is undecidable.

All these problems warrant us to look beyond computation, to dive into the world of hypercomputation.

Formal Definition of a Turing Machine
Before discussing further, it's worthwhile to study the formal structure of a Turing Machine once as it will be used time and again in various hypercomputation ideas.

The Turing Machine is visualized in the given figure. It consists of a finite control, which can be in any of a finite set of states. There's also a tape divided into cells each of which can store any one of a finite number of symbols.

The input, a finite length string, is placed at some location on the input tape. All other cells, extending infinitely in both directions contain a special symbol called the blank symbol. Then's there's a tape head always positioned at one of the tape cells. Initially, it's at the left most cell holding the input.

A move of the Turing Machine changes the state of finite control and the tape symbol scanned. In one move, Turing Machine will:

1. Change state of finite control
2. Write a tape symbol on cell of the tape head
3. Move the tape head left or right

Formally, we describe a Turing Machine by a 7-tuple:

$$M=(Q,\Sigma,\Gamma,\delta,q_0,B,F)$$

where
$Q$ : Set of states of the finite control
$\Sigma$ : Finite set of input symbols
$\Gamma$ : Finite set of tape symbols. $\Sigma \subseteq \Gamma$
$\delta$ : Transition function. $\delta(q,X) = (p,Y,D)$. In this case, TM is presently in state $q$with tape symbol $X$ and the transition takes the TM to a new state $p$, writes $Y$ onto the tape and $D$ is the direction in which the tape moves
$q_0$ : Start state
$B$ : Blank Symbol. $B \in \Gamma$ but $B \notin \Sigma$
$F$ : Set of final of accepting states

A Turing Machine accepts a string in a language if it arrives at a accepting state. Else, it runs forever. It there is also a set of rejecting state, it might reject a language.

What doesn't constitute hypercomputation
Before delving into proposed hypercomputation models, it will be worthwhile to take a look at what hypercomputation is not rather than what hypercomputation is. Some of the falsely perceived notions of hypercomputation are :-

1. **More than two characters as alphabet** This may reduce the running time of Turing Machine. However any such configuration can uniquely be expressed by using binary digits only via a suitable encoding. Suppose the Turing Machine deals with Ternary Alphabet. It can be uniquely mapped to a binary alphabet by adopting the following convention: 0 → 00, 1 → 01, 2 → 1.

This convention is inspired from Huffman's Code.

2. **Turing Machine with many tapes** Ideally only one tape is sufficient to describe a Turing Machine. However, for simplicity there can be more than 1 tape associated with a Turing Machine. This in no way, increases the computational power of Turing Machine.
3. **Turing Machine with a semi-infinite tape** As a matter of fact, a Turing Machine with semi-infinite tape is equivalent of a normal Turing Machine with infinite tape in terms of computational power. This can be shown by showing a bijection between the set of integers and the set of natural numbers.
4. **Turing Machine and Pushdown Automata** Pushdown Automatas and Turing Machines have a very intimate relation with each other. Any language that is not accepted by Pushdown Automata with one stack can be accepted by a Turing Machine. It turns out if we give the Pushdown Automata two stacks, then it can accept any language that a Turing Machine accepts. However, increasing the number of stacks doesn't increase its computational power. In fact, we can even store a counter in place of stack which gives rise to Counter Machines.

Hypercomputation

By the end of the twentieth century, many computer theorists wondering about what's beyond Turing Machine? Can it be tweaked to make it more powerful? They came up with following approaches to increase computation power :-

- **Zeno's Machine** The most classic of all is Zeno's Machine. As evident by its name, it is inspired by Zeno's Paradox. The Zeno Machine performs its first computation in, say, 1minute, the second step in ½ second, the third in ¼ second and so on. In a way, even if there are infinitely many steps of computation, it would halt within 2 seconds as the sum of the infinite geometric series 1+½+¼ +.. is 2. It exploits the fact the every step of computation has to be over in 2 seconds even though there may be infinitely many steps. Such a machine is also called Accelerated Turing Machine.
- **O-Machines** Oracle Turing Machines, abbreviated as O-Machines, was proposed by Turing as an extension to Turing Machines. Informally speaking these Turing Machine in addition have an oracle sitting on top of them which has the power to decide some particular language. Note that it can be any particular language, not even computable. However, the oracle must be set a prior. Another obvious problem is that of oracles itself.
- **Turing Machines with Initial Inscriptions** Another variation to Turing Machine proposed is to allow it to begin with information already on its tape. Adding constant number of bits doesn't increase the power. However, we can add infinite number of bits, the machine's capabilities are increased. This is, in fact, akin to just another just another formulation of Turing Machine.
- **Accessing real variables** A few paragraphs before, we discussed that we just showed that increasing the set of alphabet doesn't increase computational power. However, if we start admitting the set of reals, a real computer can perform hypercomputation because a real has infinitely many digits so the whole of the description can essentially be encoded in finitely many steps. However most of the modern computers can do calculations only up to a precision. If possible, this will be bizarre physical implications. For example, Chaitin's Constant will then be computable.
- **Coupled Turing Machine** This is a Turing Machine with one or more input channels,providing input to the machine while the computation is in progress. Some theorists claim that interaction and algorithms are not analogous. This is a classic example of interaction. However, if we supply the whole input all the once, the above model is as good as a classic Turing Machine. To go beyond the notion of classic Turing Machine, randomization is introduced and content of different input channels is dependent on the previously returned values. This is given way to a

whole different field of Interactive Proofs in Complexity Theory.

- **Error Prone Turing Machines** Here, we allow the Turing Machine to do error, albeit a very small amount. It returns 0 when it should return 1 and vice-versa. In such a formulation, even halting problem and other non-recursive functions can be solved/computed, depending on the error functions.
- **Probabilistic Turing Machine** A simple extension of Error Prone Turing Machine is the Probabilistic Turing Machine where the next state at any transition is chosen with equal probability. Such a Turing Machine is said to compute a function if chances of getting the correct output is greater than ½.
- **Infinite State Turing Machine** This happens when the automata associated with the Turing Machine has infinitely machine states. However, this is as good as hardcoding each of the strings in the language into the Turing Machine.
- **Inductive Inference Turing Machine** As evident from the name, such a models enables non-recursive languages to be "learned in the limit". Suppose the machine stabilizes on an output for a relatively long period of time. Then this value is returned. However, correctness can only be guaranteed by running the Machine forever.
- **Fair Non-Deterministic Turing Machine** A non-deterministic Turing Machine is a TM in which the transition function leads to more than one choices. A string is accepted if it is accepted in at least one of the branches and rejected only if non of the branches accepts it. A non-deterministic TM can be thought of using parallel processors for computation. However, this is no way, increases the computation power of TM. A computation of a TM is said to be unfair if it lands in a state infinitely many number of times and one of the transitions from there is never chosen. A non-deterministic TM is fair if it produces no such unfair computations. A simple manipulation suggests an ingenious way to solve halting problem : Take a fair non-deterministic TM and given a string, write a number n in unary at the end of the input. Run the TM for n number of steps. If the TM accepts within n accepts, return 1. Else increment the counter n and keep doing so. It is clear that the TM will halt if the string is in the language. What happens when computation doesn't halt? Surely, the counter is incremented infinite number of times and avoids halting each time. Surely this would be an unfair computation and this is not possible. In this strange way, a fair non-deterministic TM can compute the halting function.

Shortcomings of previous work

Though so many different hypercomputation techniques have been proposed, not even a single so far has had a practical implication. Even other tweaks to Turing Model of Computation proposed such as Quantum Computation are relatively better off and have been, albeit a very little extent, developed practically. On the major problem stagnating the development of almost all the proposed hypercomputation techniques is the apparent infeasibility of some step in that hypercomputation proposal.

For example, Accelerated Turing Machine talks of doubling the speed of every next step. This idea, though simple to mind, is just incomprehensible practically. This surmounts to developing, in a way, a perpetual motion machine who realizability again seems physically impossible. For techniques resting on an oracle, the existence of a oracle itself is really questionable. Secondly, can an oracle for complexity problems solve computational problems? Yet another set of proposed approaches demand infinite precision for example taking real numbers as the set of alphabets. This again is something not realizable as every machine has some precision no matter how small it is. Furthermore, yet another class of problems demands some initial input or external help to perform the computation task. Why not ask the decidability of the problem itself? There certainly seems to lie a caveat in all these ideas.

Other approaches such as the probabilistic one seems to dilute the notion of computability. The collapse here is analogous to the collapse of an NP-Problem to a P-Problem in probabilistic scenario characterized by probabilistic complexity classes. Admitting probabilistic solutions leaves a window for error which can never be closed completely no matter how small the error is. Other solutions suggested such a returning an output when the Turing Machine stabilizes over a long period of time is also in a way probabilistic solutions.

Other solutions can be thought of more as an mind game rather an achieving anything tangible.

As a matter of fact, full computational powers of a Turing Machine can be achieved in very simple ways. Specifically speaking, we have what we fondly call as Esoteric Programming Language which is used to test the boundaries of a computer programming language design as a proof of concept. These languages are also Turing complete and are way to encode Turing Machines in a way as succint as possible. The term Turing Tarpit is used to define a Turing-complete programming language which minimum number of programming constructs.

Proposed solutions
I am going to propose a few solutions, in fact, areas which are relatively unexplored and warrants to study to further unravel the mysterious field of hypercomputation.

1. **Biological Models**

Biological models of computation is a field that has been adopted time and again in Computer Science. The most classical example of all is that of Artificial Neural Networks in Machine Learning where Minsky and Papert in their book titled Perceptron prove that a ANN with 3 hidden layers is Turing-complete. In the recent past, there have been studies on biological hypercomputation. Hence, if we can't come up with models on our own, can they actually be based in nature itself? If yes, it might be worthwhile to explore these models further.

2. **Non-Deterministic Language**

In a deterministic framework, one can always predict the next state given the present state of computation. This is not true for a non-deterministic language which supply a built-in randomization instruction. In such a setting, getting a reliable result for even trivial input is a monumental task. However, it can be used to explore large spaces where exhaustible search is impractical. This can have a impact in the theoretical investigation of hypercomputation.

Philosophical Implications
The computational theory of philosophy of mind looks at the possibility that the computations performed in the brain gives rise to mind. If is often claimed that if mind is indeed computational in nature, we can simulate it on a Turing Machine which also appeals to the Churh-Turing Thesis. However, Copeland argues that brain is much more powerful than a Turing Machine and studying its philosophy can unearth the hypercomputational processes in our brain.

Hypercomputation also seems to have significant implications in epistemology. Classical computation keeps a cap on the number of valid conclusions that be deduced from a set of premises but hypercomputation generalizes this with the amount of valid inference growing with the computational resources available. This not only relativizes provability and inference, but makes the actual amount of inference that is possible in the real world dependent upon the true nature of physics and a proiri as it is generally believed.

Another deciding factor while deciding the relevance of a certain theory is its simplicity which can be described by its Prefix Complexity in which scenario hypercomputation becomes relevant. To measure the complexity of a theory, we must do it relative to a computational model and the one giving the minimum complexity is likely to be a hypercomputation model.

Though the implications of hypercomputation is still not crystal clear, it should definitely not be overlooked. Several areas of philosophy involve computational concepts and the widespread lack of hypercomputation can cause important observations to be overlooked.

Conclusion and Future Work

Models of computation more powerful than a Turing Machine can be possible at some point in the future. This would move on the debate to 'uncomputable' from 'uncomputable by a Turing Machine'. Since, the proof of incompleteness also relies of some model of computation and halting problem in uncomputable by a Turing Machine, hypercomputation might show us that great negative results obtained in 20th century might actually not a that negative. If this is indeed possible, we may need a further refinement of Godel's Incompleteness Theorem and need to define finer boundaries between what's actually computable/provable and what's not. Though Incompleteness Theorem will still be valid in a hypercomputational model, it might actually compute the halting function. The boundaries of incompleteness will be further pushed and as such, it would be interesting to explore if there exists some problem which is not computable even by a hypercomputation machine.